

*Dave Mark Cartwright Reed*

# *Macintosh C Programming*

**P R I M E R**

*Volume I  
Second Edition*

*Inside the Toolbox Using THINK C™*

---

---

# **Macintosh C Programming Primer**

*Inside the Toolbox  
Using THINK C™  
Volume I, Second Edition*

**Dave Mark    Cartwright Reed**



**Addison-Wesley Publishing Company**

*Reading, Massachusetts    Menlo Park, California    New York  
Don Mills, Ontario    Wokingham, England    Amsterdam  
Bonn    Sydney    Singapore    Tokyo    Madrid    San Juan  
Paris    Seoul    Milan    Mexico City    Taipei*



The authors and publisher have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial capital letters.

### **Library of Congress Cataloging-in-Publication Data**

Mark, Dave.

Macintosh C programming primer / Dave Mark, Cartwright Reed. —  
2nd ed.

p. cm.

Includes bibliographical references and index.

Contents: v. 1. Inside the toolbox using THINK C.

ISBN 0-201-60838-3 (v. 1)

1. Macintosh (Computer)—Programming. 2. C (Computer program language) I. Reed, Cartwright. II. Title.

QA76.8.M3M368 1992

005.265—dc20

92-4299

CIP

Copyright © 1992 by Dave Mark and Cartwright Reed

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

Cover Concept: Doliber/Skeffington

Set in 10/12 Palatino by ST Associates, Wakefield, MA

123456789-MW-9695949392

*First printing, May 1992*

*This book is dedicated to Deneen and Kate.*

---

---

# Contents

*Source Code Disk for the Mac Primer*      ix

*Preface*      xi

*Acknowledgments*      xiii

## **1 Introduction      1**

The Macintosh Way      3  
About the Book      8  
Writing Macintosh Applications      11  
How to Use This Book      19  
What You Need to Get Started      20  
Ready, Set . . .      21

## **2 Setting Up      23**

Installing THINK C 5      25  
Accessing the Toolbox with C      29  
The Hello, World Program      40  
In Review      48

## **3 Drawing on the Macintosh      49**

Introduction      51  
Window Management      56  
Drawing in Your Window: The QuickDraw Toolbox Routines      62  
The QuickDraw Programs      69  
Hello2      69



Walking Through the Hello2 Code	79
Variants	83
Mondrian	88
Walking Through the Mondrian Code	92
Variants	98
ShowPICT	103
Walking Through the ShowPICT Code	109
Variants	113
Screen Saver: The FlyingLine Program	113
Walking Through the FlyingLine Code	119
In Review	126
<b>4 Events</b>	<b>127</b>
Understanding Events	129
The Event Manager	129
A New Structure for Macintosh Programming (Part 1)	133
A New Structure for Macintosh Programming (Part 2)	139
EventTracker	143
Walking Through the EventTracker Code	154
Handling mouseDown Events in EventTracker	165
Building the EventTracker Application	167
Updater: The Return of ShowPICT	168
Walking Through the Updater Code	177
Handling mouseDown Events in Updater	184
EventTrigger: Sending Apple Events	190
The Event Trigger Algorithm	192
Walking Through the EventTrigger Code	196
In Review	199
<b>5 Menu Management</b>	<b>201</b>
Menu Components	203
The Hierarchical Menu	206
The Pop-up Menu	206
Adding Menus to Your Programs	208
WorldClock	209
Walking Through the WorldClock Code	235
In Review	258
<b>6 Working with Dialogs</b>	<b>259</b>
How Dialogs Work	262
Dialog Items: Controls	263
Other Dialog Items	266
Modal Dialogs	267
Modeless Dialogs	267

---

Adding Dialogs to Your Programs	268
Working with Alerts	275
The Notification Manager	278
Using the Notification Manager	279
The Process Manager	281
Reminder	284
Resources	285
Running Reminder	328
Walking Through the Reminder Code	335
In Review	361
<b>7 Toolbox Potpourri</b>	<b>363</b>
Writing Out Resources	365
ResWriter	365
Walking Through the ResWriter Code	373
Scroll Bars! We're Gonna Do Scroll Bars!	379
Pager	382
Walking Through the Pager Code	393
The Scrap Manager	404
Walking Through the ShowClip Code	411
The Sound Manager	416
SoundMaker	416
Walking Through the SoundMaker Code	420
Working with Macintosh Files	424
The Standard File Package	425
The File Manager	427
Walking Through the OpenPICT Code	436
The Printing Manager	443
PrintPICT	444
Walking Through the PrintPICT Code	450
In Review	456
<b>8 Finishing Touches</b>	<b>457</b>
Building a Standalone Application	459
More Finder Resources	471
The Help and Edition Managers	478
Responding to the Required Apple Events	480
In Review	489
<b>9 The Final Chapter</b>	<b>491</b>
Macintosh Periodicals	493
The Essential <i>Inside Macintosh</i>	493
Apple Technical References	496
Apple's Developer Programs	497

Macintosh Developer Technical Support and Applelink	498
Software Development Tools	499
Source Code Bounty	501
To Boldly Go	502
<b>Appendix A Glossary</b>	<b>503</b>
<b>Appendix B Code Listings</b>	<b>521</b>
<b>Appendix C THINK C Command Summary</b>	<b>603</b>
<b>Appendix D The Debugger Command Summary</b>	<b>615</b>
<b>Appendix E Debugging Techniques</b>	<b>623</b>
<b>Appendix F Building HyperCard XCMDs</b>	<b>629</b>
<b>Appendix G Bibliography</b>	<b>639</b>
<b>Appendix H New Inside Macintosh Series</b>	<b>643</b>
<b>Index</b>	<b>647</b>



---

---

# Source Code Disk for the Mac C Primer

IF YOU WOULD like the source code presented in the *Macintosh C Programming Primer* on disk, please send in the coupon on the last page (or a copy of the coupon; we're not picky). If you like, you can order the disk by calling (215) 387-6002.

We hope you like the *Macintosh C Programming Primer*. If you have any comments or suggestions drop us a line on CompuServe. When you log on to CompuServe, type GOMACDEV, then stop by the Learn Programming area (Section 11) and say hello.

---

---

# Preface

ARE YOU INTERESTED in creating your very own Macintosh applications? Is the next great Macintosh software miracle tucked away in your brain? If so, you've come to the right place.

With this new edition of the *Macintosh C Programming Primer* in hand and a copy of THINK C on your hard drive, you've got everything you need to enter the wonderful world of Macintosh programming. The *Mac Primer* uses step-by-step approach that shows you how to add each element of the Macintosh user interface to your programs. You'll start by learning how Macintosh windows are created. Next, you'll learn how to draw text and graphics inside your windows. As you progress through the book, you'll learn about events, menus, dialog boxes, and much, much more.

The *Mac Primer* is chock-full of reusable sample code. Each program is discussed in detail. Nothing is left as an exercise for the reader. There's no better way to learn the art of Macintosh programming.

For readers of the first edition, this edition of the *Mac Primer* presents a boatload of brand-new material. All the source code has been rewritten and specifically designed with System 7 in mind. You'll learn how to create System 7-savvy programs using the latest versions of THINK C and ResEdit. You'll also get the inside scoop on the new additions to the *Inside Macintosh* family.

Get yourself a copy of THINK C, grab your *Mac Primer*, and we'll meet you inside.

Dave Mark  
Arlington, VA

Cartwright Reed  
Philadelphia, PA

---

---

# Acknowledgments

WE'D LIKE TO express our appreciation to the people who helped make this book possible, or at least coexisted with us harmoniously during its development.

First and foremost, we'd like to thank our wives, Deneen Melander and Kate Joyce, who know us and still smile.

A special thanks to Elizabeth Rogalin and Julie Stillman for loyalty, late nights, and much-needed moral support. Thanks also to Kathy Traynor, Diane Freed, and the rest of Addison-Wesley's finest! Thanks also to Jackie Cowlshaw, copyeditor extraordinaire.

A special note of thanks to Andy Richter of Intelligence At Large. Andy was a critical factor in making the code both clean and exciting. Thanks, Andy!

Thanks to Jim Reekes, Richard Clark, and Phil Shapiro for much-needed technical support. These guys kept us from going down for the compatibility count.

Finally, thanks to Apple and Symantec, whose products are the finest in the land.



---

# Introduction

*The Macintosh Programming Primer  
is a complete course in the art of  
Macintosh programming. With this  
book and Symantec's THINK C, you  
can learn to program the Macintosh.*

---

NO OTHER COMPUTER is like the Macintosh. Some computers look like it, others claim they work like it, but the Mac remains unique. Writing programs for the Macintosh is also a unique process, and the *Macintosh Programming Primer* is the shortest, best path toward becoming a good Mac programmer.

At the heart of the Macintosh is the Toolbox, a collection of over 1,400 procedures and functions that give you access to the Macintosh environment. The *Mac Primer* will teach you how to use the Toolbox; that is, how to add the power of pull-down menus, windows, and scroll bars to your programs.

This book serves as a bridge to the Macintosh way of programming. If you can't wait to code, and you've already installed THINK C on your hard drive, skip to the beginning of Chapter 3 to get started on your first Macintosh application. If you have the time, though, keep right on reading.

If you've read the first edition of the *Macintosh Programming Primer*, welcome back! One of the most important changes you'll find in this edition is the comprehensive coverage of System 7, Apple's exciting new operating system. *All of the sample programs presented in this book were written with System 7 in mind.* You'll learn how to send and receive Apple events, record and play sounds, use the Notification Manager, and much more. With the *Mac Primer* in hand, you'll be able to add System 7 savvy to your own applications.

If you're new to Macintosh programming, and wonder what "Apple events" are, or what the "Notification Manager" is, the *Mac Primer* is the perfect place to start. By way of concise source code examples, you'll quickly master the basics and the newest aspects of the Macintosh operating system. The concepts in this book are presented side by side with a complete set of source code examples. We don't skip the basics in bringing you up to speed.

---

## The Macintosh Way

---

Nowadays, the Macintosh line is successful, highly praised, emulated, and affordable. From the PowerBooks to the high-powered Quadra workstation, the Macintosh is on everyone's A-list. When the first Macintosh was introduced in 1984, however, people were perplexed: It was like no computer they had ever seen—a beige box with a little screen and a mouse. People called the Macintosh a toy because it had a graphical user interface, and pictures were not the way normal computers communicated.

We've come a long way since the '80s. Microsoft, Hewlett-Packard, and finally IBM have made their way to Apple's door. Graphical User Interfaces (GUIs) are a dime a dozen, mice are legion, and hardware standards espoused by Apple, such as NuBus and SCSI (Small Computer Systems Interface) have propagated throughout the industry. Apple naysayers have disappeared, replaced by company reps who claim their computers work "just like a Mac." They might be right, if the Mac were just another pretty interface.

The Mac's elegance, ease of use, and power stem from a combination of **interface**, **Toolbox**, and **resources**.

- **The Macintosh interface.** Anyone who's ever used a Macintosh is familiar with the Mac interface. Pull-down menus, movable windows, scroll bars, and graphics all combine to make the Macintosh one of the friendliest computers ever designed.
- **The Toolbox.** Comprehensive routines were defined in the Macintosh ROM that drove the interface and allowed software designers to write powerful, easy-to-use applications. Whether you're building a Sanskrit word processor, or just need to find out what time zone you're in, the Mac Toolbox designers have prepared the path for you.
- **The use of resources.** The building blocks for all software on the Macintosh, resources store program information in a series of templates in the program file, simplifying the creation and modification of Mac programs.

These three ingredients—interface, Toolbox, and resources—combine to make the Macintosh the most versatile computer ever made. Advanced capabilities, such as recording and playing back video (a kind of "MacTV"), virtual memory, and personal file-sharing are not only featured in the new Macs of the '90s, but are also available in Macs made almost a decade ago by installing new versions of the Macintosh operating system. The careful planning that went into the original Mac has paid off handsomely, as the Mac line continues to evolve and improve.

To write successful applications for the Macintosh, the would-be Mac programmer must understand how the interface, the Toolbox, and resources work together. First, let's look at the most visible of the three: the Macintosh user interface.

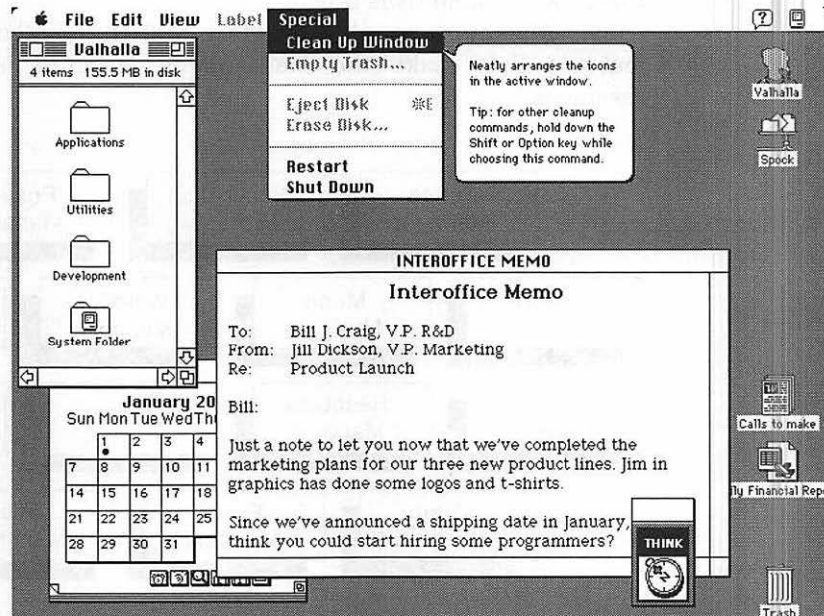


## The Macintosh Graphical User Interface

The Macintosh makes a great first impression on new users with its sophisticated user interface. Figure 1.1 shows some of the distinctive elements of the Mac “look.” Because neophytes understand and use the windows and menus of Mac applications intuitively, the Macintosh interface represents an impressive improvement over both “command-based” interfaces, such as MS-DOS, and windowed interfaces, such as Microsoft Windows, which is built on top of MS-DOS. Each element of the interface—windows, menus, dialog boxes, and icons—has a specific function associated with it, and extensive guidelines exist for the proper use of each element. With the implementation of System 7, the interface became more visually exciting, with brightly colored icons and friendly balloons of text that appear when you click the mouse on something you want to know about.

Of course, pretty pictures aren’t enough. The beauty of the Mac interface lies in how it is created. Each part of the interface is manipulated by a series of routines in the Macintosh ROM. For example, you can create an application’s window with one call to the Macintosh ROM.

The routines that underlie the interface—that build windows, control printing, and draw menus—are known collectively as the **Macintosh Toolbox**.



**Figure 1.1** The Macintosh desktop (circa 1992).

## The Macintosh Toolbox

The Toolbox can be thought of as a series of libraries that make it easy for you to create the features indigenous to Macintosh applications. For example, the Macintosh Toolbox routine `GetNewWindow()` creates a new window for use in your application.

Using Toolbox calls to create your applications gives the results a distinctive Macintosh look and feel. Operations common to most applications, such as cutting, copying, and pasting, are always handled in the same way, which makes it easier to learn a new application.

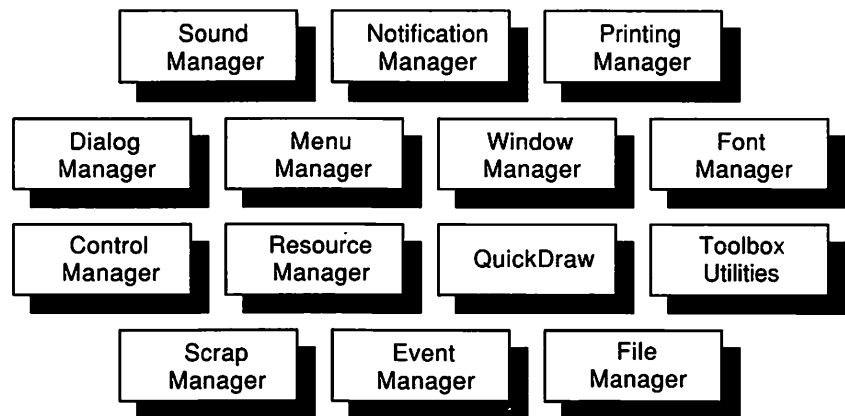
The Toolbox routines are grouped functionally into **Managers**, each of which is responsible for one part of the Macintosh environment (Figure 1.2).

The Macintosh Toolbox undergoes constant updating and modification; each new system revision gives you some shiny new tools as well as the old standbys to work with. As new routines are added to the Toolbox, Apple cleans up problems with older routines.

This brings us to System 7.

Apple has provided new Toolbox calls that revolutionize the programmer's ability to control text, audio, animation, and even real-time video on the Mac screen. These functions can all be incorporated into your program with just a few lines of code. At the same time, lower-level routines allow you to fine-tune any of these new features. Later in the book, we'll talk more about the revolution in Mac programming these new system tools provide.

The Macintosh graphic interface and the Toolbox are two of the features that make the Mac unique. A third is the concept of resources.



**Figure 1.2** Parts of the Toolbox.



Although the Macintosh line has expanded greatly, the basic compatibility of the different Macintosh models has been preserved. Yet, more powerful machines always provide more choices—and more decisions. When the only available Mac workstations were the Macintosh and the Mac Plus, software developers thought they had a certain flexibility in the way they followed the Mac programming guidelines provided by Apple. Now, in the midst of portables, workstations, and midline Mac CPUs that have widely differing capabilities, the successful developer hews closely to the Macintosh standards.

## Resources

If the Toolbox is the library of routines that make up the Macintosh interface, resources are the data your program uses to execute these library calls. `GetNewWindow()`, the Toolbox call that creates a new window, requires you to specify window parameters, such as size, location, and window type. To do this, you create a resource containing that information, passing the resource to `GetNewWindow()`. `GetNewWindow()` uses the resource information to build the requested window.

Resources come in various types, each relating to a particular element of the Macintosh interface. For example, a resource of type `WIND` contains all the information necessary to build a window. There may be a number of resources of type `WIND`, but there is only one `WIND` type, which is identical for all Mac applications.

Resources are integrated into the design of the Macintosh. Each Mac application file may possess dozens of resources. This simplifies many of the tasks of the applications programmer. For example, resources make it easy to localize a program for a different area. If you want to sell your program in France, say, it is relatively easy to replace resources containing English text with their French equivalents.

Resources are also essential in developing the complex code that drives the Macintosh interface and hardware. Because they can easily be copied from one program to another, menus and dialog boxes need not be created more than once. After you have built up a collection of programs, creating new ones may begin with a cut-and-paste session with your existing programs.

To edit resources, Apple developed a program called **ResEdit**, which allows you to edit any of the resources in *Macintosh Primer* programs. You can also use ResEdit to explore other Macintosh applications—even system files! Because these resources exist as part of the completed application, they can be edited without recompilation.

We make extensive use of version 2 of ResEdit throughout the *Mac Primer*, and describe the new resources required for successful System 7 programming. Even if you've never worked with ResEdit, you'll find the instructions in the *Macintosh Primer* complete and easy to follow.

The Macintosh interface, Toolbox, and resources are the three intertwined subjects we'll cover using THINK C and ResEdit to create standalone Macintosh applications. The next sections discuss our approach to learning about these issues.

---

## About the Book

---

Most Macintosh reference books, such as *Inside Macintosh* and *Macintosh Revealed*, are excellent texts for those who already understand Macintosh programming. They can be frustrating, however, if you're new to the Macintosh programming world. The *Mac Primer* bridges the gap for those of you who are just learning the basics of Mac programming.

Our aim is to help you write properly structured Mac applications. If you're used to programming on an MS-DOS computer or a UNIX system, the *Mac Primer* is the perfect place to start your Mac programming education. Our formative years were spent programming under UNIX on machines like the PDP-11 and the VAX-11/780; we've also spent a lot of time with IBM PCs and compatibles. We wrote the *Macintosh Programming Primer* with you in mind.

If you've programmed on the Macintosh before, but haven't checked out System 7, you'll find a lot of solid code in the *Primer* that should help you implement new System 7-friendly versions of your applications.

## What You Need to Know

There are only two prerequisites for reading this book. Before starting the *Macintosh Primer*, you should already have basic Mac experience: You should be able to run Macintosh applications and have a good feel for the Mac user interface. In addition, you should have some experience with a programming language, such as C, Pascal, or BASIC. If you have no programming experience, or if your computer language skills are rusty, get a book on the C language to supplement this book. *Learn C on the Macintosh* by Dave Mark is designed for this purpose and comes with a custom version of the THINK C language on disk.

The *Macintosh C Programming Primer* examples are all written in C, using the THINK C development environment. Our general

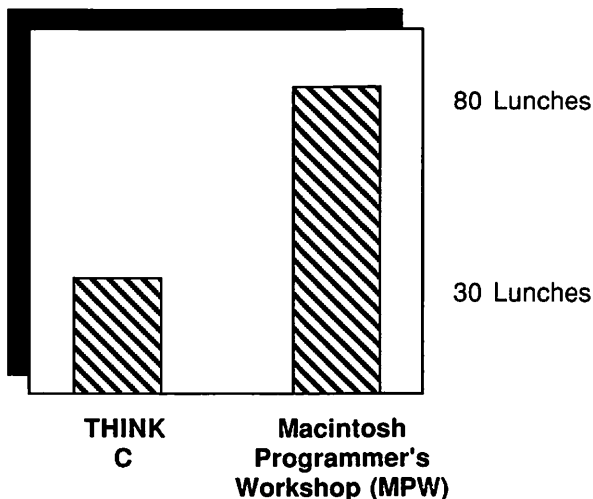
approach, however, emphasizes the techniques involved in programming with the Mac Toolbox. The skills you learn will serve you no matter what programming language or environment you intend to use in the future.

## Why We Chose THINK C

Many development environments are available to the Mac programmer. The **Macintosh Programmer's Workshop (MPW)** is a complex and powerful development system written and marketed by Apple. Most of Apple's internal development is done with MPW, and many of the large Macintosh software development houses have made MPW their first choice. MPW uses an "everything but the kitchen sink" approach to software development. The basic system consists of an editor shell that allows you to edit your source code as well as build and execute complex command scripts. You can do just about anything in MPW, but it is definitely not a system for beginners. In addition to learning the editor and shell, you have to install, configure, and (oh, yes) pay for your choice of compilers. You can buy C, Pascal, and even FORTRAN compilers for MPW. MPW is ideal for complex, multilanguage development efforts, but not for learning to program the Macintosh.

THINK C is a powerful and friendly development environment. It has concise, accurate documentation. For those inevitable bugs, it has the best debugging utilities on the market. It also has excellent support for programmers who write exclusively for System 7.

Finally, THINK C is still reasonably priced (Figure 1.3).



**Figure 1.3** Lunch economics.

## Using THINK C

THINK C is an integrated development environment. The source code editor follows all the standard Macintosh conventions and is very easy to use. The compiler is smart: It keeps track of the files you're currently working with, noting which have been changed since they were last compiled. THINK C recompiles only what it needs to.

THINK C has a well-thought-out Macintosh interface. For example, to build a standalone application, pull down the **Project** menu and select **Build Application**. Installation is simple: Just pull the floppies out of the box, copy the files onto your hard drive, and go!

THINK C also comes with integrated debugging utilities that allow you to test-drive your program while monitoring its progress in other windows. The debugging utilities also work with other Macintosh debugging tools, such as MacsBug and TMON.

## Inside THINK C

The **project** file is unique to Symantec's C and Pascal development environments. It contains the names of all of your source code files, as well as the name you'll eventually give to your application. The project file also contains compilation information about each source file, such as the size of the compiled code (Figure 1.4).

THINK C comes with class libraries similar to MacApp, Apple's ready-made library of user interface routines. THINK C debugging facilities are without peer. You can use THINK C to write programs that will take full advantage of the most advanced features of the Mac OS. All of these features are supported in the way Apple intended. THINK C also provides routines to support extensions to Apple's **HyperCard**, or Silicon Beach's **SuperCard**.

WorldClock.π		
Name	obj size	
MacTraps	8342	↑
WorldClock.c	1642	
		↓

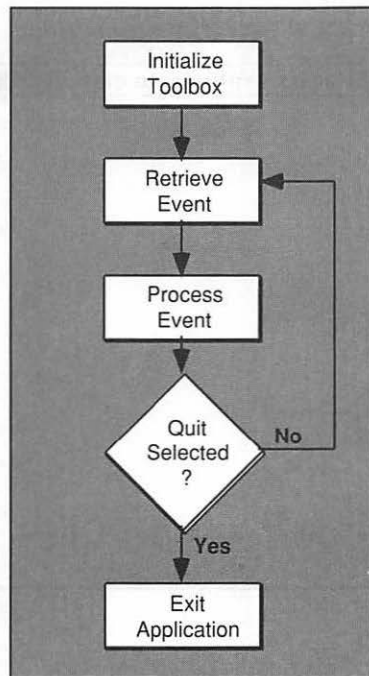
Figure 1.4 Think C Project Window.

THINK C also comes with a full complement of utilities, including ResEdit, the resource editor mentioned earlier, and useful code on various types of Mac projects, including text editors, control panel devices (cdevs), and desk accessories.

## Writing Macintosh Applications

Most Macintosh applications share a basic structure (Figure 1.5). They start off by initializing the Toolbox data structures and routines that support the Macintosh user interface. Then the application enters an event loop and patiently waits for the user to do something—hit a key, move the mouse, or some other action. Events outside the application are also checked: Desk accessories may be used, or disks may be inserted. No matter how complex the Macintosh program, this simple structure is maintained.

At the heart of the *Macintosh Programming Primer* is a set of seventeen sample applications. Each builds on the basic program structure to provide successively more sophisticated use of the



**Figure 1.5** How a Macintosh application works.

Macintosh Toolbox. Each new chapter constructs a more powerful implementation of the basic program structure. Chapter 3 programs show how to create windows and draw inside them. Chapter 4 illustrates how to handle events (including Apple events). Chapter 5 implements menus, and Chapter 6 makes use of dialogs. Chapter 7 presents a potpourri of Macintosh applications, each designed to showcase a different part of the Macintosh Toolbox.

Each *Mac Primer* example program is presented as completely as possible, and each program listing is discussed extensively. Nothing is left as an “exercise for the reader.” Each chapter contains complete instructions and figures for entering, compiling, and running the programs using THINK C.

## Chapter Synopses

The *Macintosh Primer* is made up of nine chapters and seven appendices. This introductory chapter provides an overview and starts you on your way. Chapter 2 starts by stepping through the installation of THINK C and ResEdit. Then THINK C basics are introduced. We present the standard C approach to the classic Hello, World program (Figure 1.6), and discuss drawbacks. We then go on to illustrate the programming conventions we’ll use in the *Primer*.



**Figure 1.6** Hello, World.

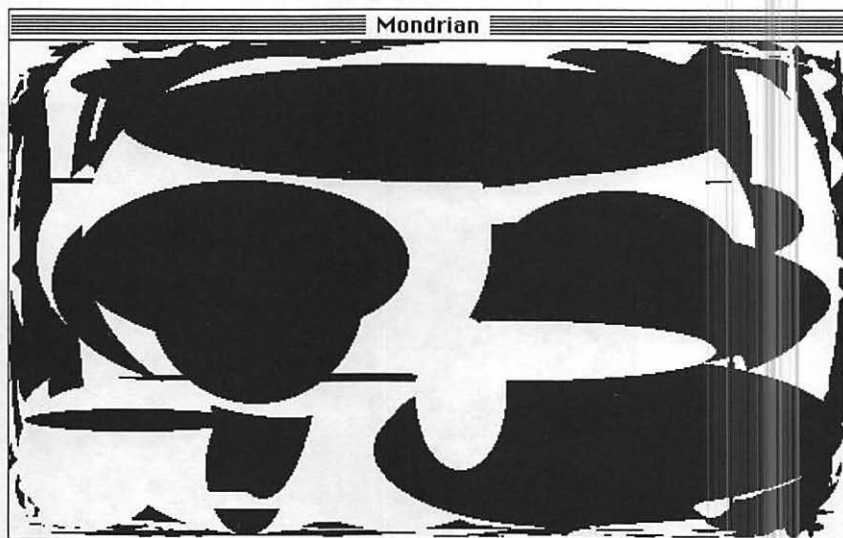


Chapter 3 starts with an introduction to the fundamentals of drawing on the Macintosh using QuickDraw. The Window Manager and windows are discussed. We then introduce resources and the Resource Manager.



QuickDraw, the Window Manager, and resources are closely related. Windows are drawn using QuickDraw commands from information stored in resource files.

Four programs are introduced in Chapter 3. The Hello2 program introduces some of the QuickDraw drawing routines related to text; the Mondrian program (Figure 1.7) demonstrates QuickDraw shape-drawing routines. ShowPICT (Figure 1.8) illustrates how easy it is to copy a picture from a program like MacDraw or MacPaint into a resource file, then draw the picture in a window of your own. Finally, as a bonus for completing the first three programs, you can try the FlyingLine (Figure 1.9), an intriguing program that can be used as a screen saver.



**Figure 1.7** Mondrian.

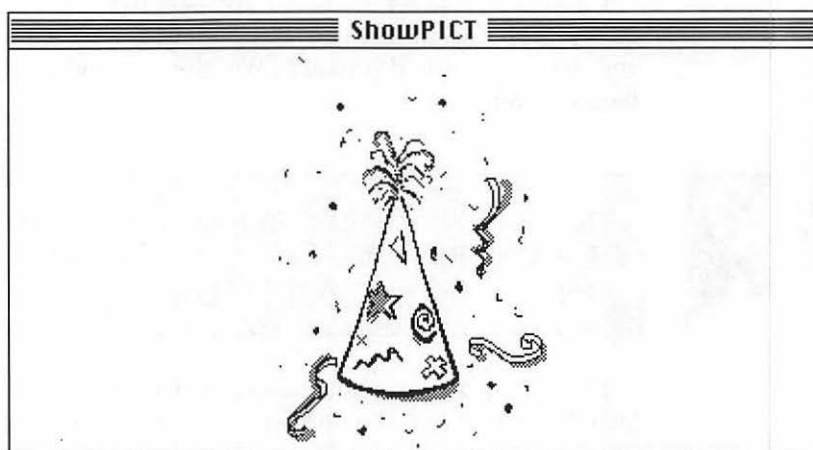


Figure 1.8 ShowPICT.

Chapter 4 introduces one of the most important concepts in Macintosh programming: **events**. Events are the Mac's mechanism for describing the user's actions to your application. When the mouse button is clicked, a key is pressed, or a disk is inserted into the floppy drive, the operating system lets your program know by queueing an event. The event architecture can be found in almost every Macintosh

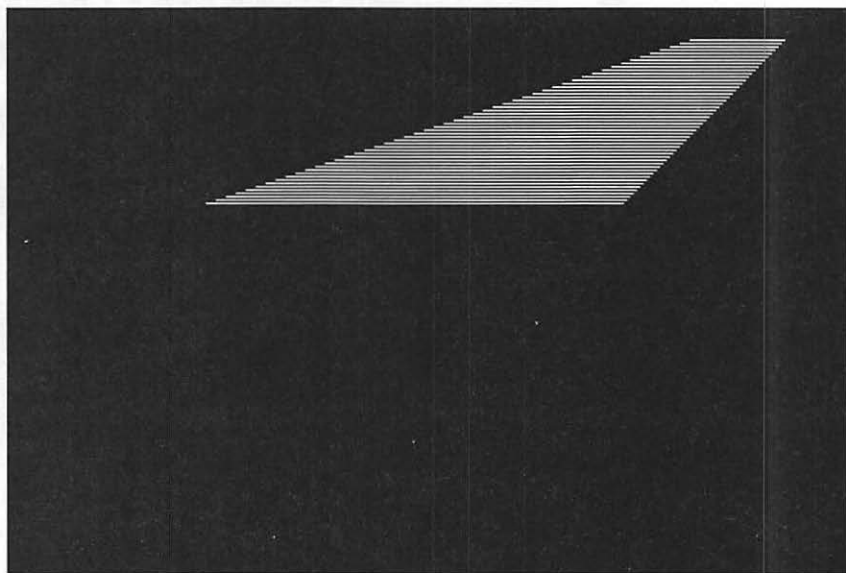


Figure 1.9 FlyingLine.

application written. This chapter presents the architecture of the main event loop and shows how events should be handled. EventTracker, Chapter 4's first program (Figure 1.10), provides a working model of the event architecture. It also demonstrates how to receive Apple events from other applications. Updater, the second program, demonstrates the proper way to handle update events in windows (Figure 1.11). The final Chapter 4 program, EventTrigger, demonstrates how to send Apple events to other applications (Figure 1.12).

Chapter 5 shows you how to add the classic pull-down, hierarchical, and pop-up menus to your own programs. Chapter 5's program, WorldClock (Figure 1.13), uses all three kinds of menus.

Chapter 6 introduces dialogs and alerts. Dialog boxes are another intrinsic part of the Macintosh user interface. They provide a vehicle for customizing your applications as you use them. Alerts are simplified dialogs, used to report errors and give warnings to the user.

The Reminder program in Chapter 6 (Figure 1.14) uses dialogs, alerts, and the Notification Manager to allow you to set an alarm. The application then starts a countdown and notifies you when it goes off—even if you're running another application.



**Figure 1.10** EventTracker.

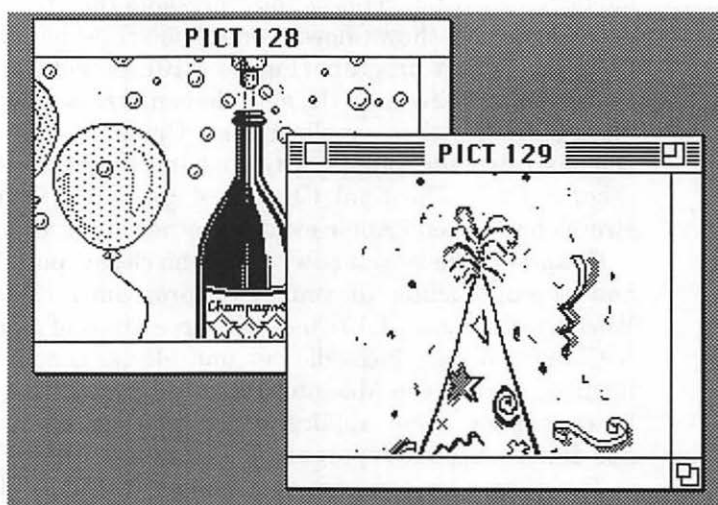


Figure 1.11 Updater.

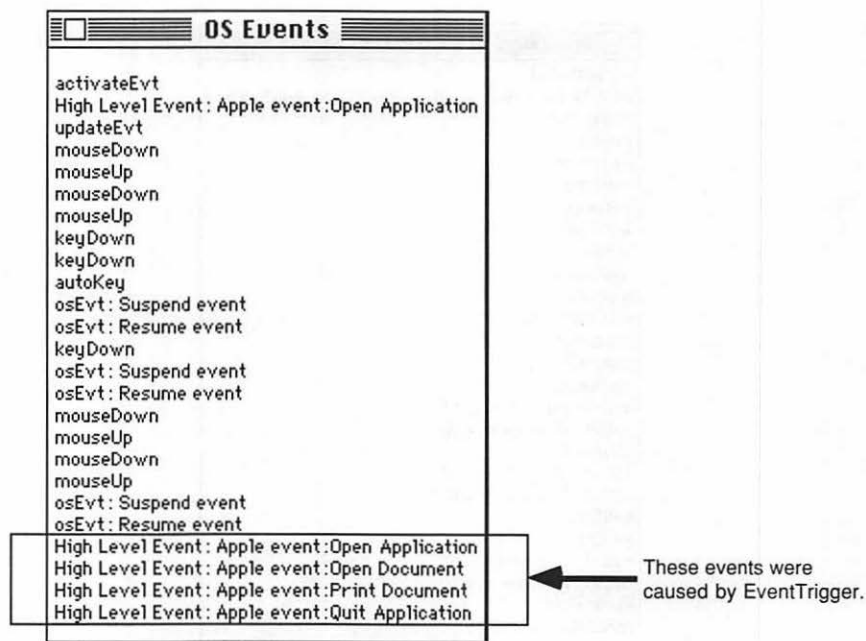


Figure 1.12 EventTrigger sends events to EventTracker.

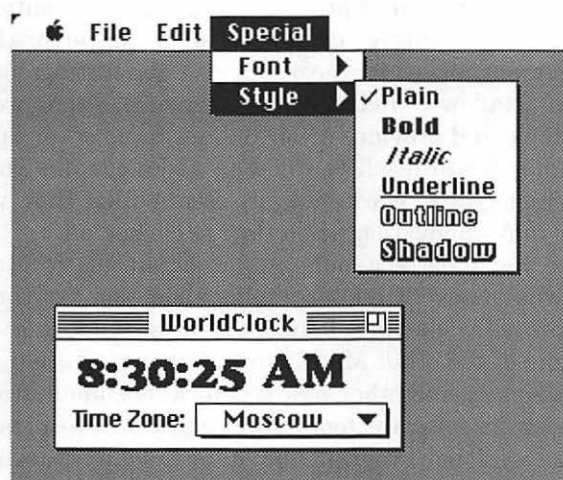


Figure 1.13 WorldClock.

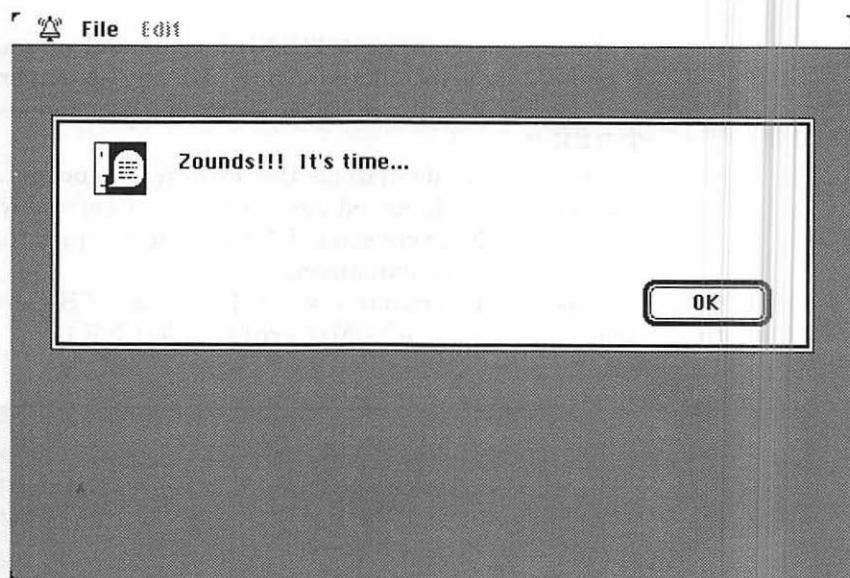


Figure 1.14 Reminder.

Chapter 7, the final programming chapter, contains a potpourri of programs illustrating concepts, such as error-checking, memory management, printing, recording and generating sound, adding scroll bars to windows, and file management. Each program explores a single topic and provides a working example of reusable code.

Chapter 8 will teach you how to add a custom icon to your application. Then, you'll learn how to create files that will automatically launch your application when they're clicked on.

After you have a handle on the essentials of Macintosh programming, what's next? Chapter 9 talks about some of the tools available to help you with your development efforts. It looks at *Inside Macintosh* and some of the other Mac technical documentation, such as the *Apple Event Registry* and other new technical documentation on System 7. It also looks at software tools, from compilers to debuggers, as well as Apple's Developer Program and other Macintosh technical resources.

Appendix A is a glossary of the technical terms used in the *Macintosh Primer*.

Appendix B contains a complete listing of each of the *Mac Primer* applications, presented in the same order in which they appear in the book.

Appendix C contains a THINK C command summary. Each THINK C menu item is introduced, along with any accompanying dialog boxes and alerts. We also discuss some of the changes in version 5 of THINK C.

Appendix D summarizes the THINK C debugger. The operation of the debugger is discussed, and each menu item and window is detailed.

Appendix E covers some debugging techniques that may be helpful in the THINK C environment.

Appendix F contains a short discussion of HyperCard 2.1 XCMDs, along with a sample XCMD written in THINK C.



For non-HyperCard aficionados, XCMDs are procedures written in C or Pascal that can be called from within HyperCard. XCMDs allow you to go beyond the limits of HyperCard, performing functions not normally available from within HyperCard.

Appendix G is a bibliography of Macintosh programming references.

Appendix H features *The New Inside Macintosh* series coming soon from Addison-Wesley Publishing Company.

## How to Use This Book

Each *Macintosh Primer* chapter is made up of the **main text** and **tech blocks**. The main text is the narrative portion of the book. Read this first. It contains the information you need to input and run the example programs. Because we've placed a premium on getting you going immediately, we have you run the program before discussing how the code works. Impatient programmers are invited to go directly to Appendix B, which contains listings of all the programs discussed in the book. If you have questions after typing in the programs, refer to the chapter in which the program is discussed. If you prefer a more sedate pace, read a chapter at a time, type in the programs, and test them as you go.

At some points, we expand on the narrative with a tech block, indicated by a distinctive gray background. It's OK to ignore them during your first read-through. An icon at the left of a tech block tells you what the tech block subject matter is. For example:



Sometimes System 7 provides a way of doing things that is incompatible with earlier systems. This kind of tech block shows you how to handle these incompatibilities.



This kind of tech block continues the current issue being discussed at a deeper, more bit-twiddly level. Such tech blocks can be passed by in your first reading.



This kind of tech block gives historical perspective to why a certain feature is the way it is, or discusses the way earlier versions of the Mac OS handled specific situations. Forward-looking, healthy-minded individuals unconcerned with the past can skip these tech blocks.



This final kind of tech block contains a warning about a technique or coding situation where novice Mac programmers may go astray. Read these carefully.



Several important terms and conventions are used throughout the *Macintosh Primer*. Whenever you see a notation like this:

(VI:256–272)

it refers to a volume of *Inside Macintosh* and a set of pages within that volume. The example here refers to Volume VI, pages 256 to 272. References to *Tech Notes*, documentation from Apple's Macintosh Developers Technical Support Group, are annotated like this: (TN:78) (referring to Tech Note 78). (See Chapter 9 to find out how to get *Tech Notes*.) These references to *Inside Macintosh* and *Tech Notes* are intended to help readers who are interested in a further discussion of a topic.

All of our source code is presented in a special font. For example:

```
{
    i = 0;
    MakeItSo();
}
```

Toolbox routines and C functions are also in the code font when they are described in the text. **Menu titles**, **menu items**, and **dialog items** appear in the book in Chicago font just as they do on the screen.

Finally, **boldface** is used to point out the first occurrence of important new terms.

---

## What You Need to Get Started

---

First, you'll need THINK C from Symantec. The examples from the book use version 5. You'll also need a Toolbox reference manual. Apple's *Inside Macintosh* series is the authoritative reference on Macintosh software development. We suggest you purchase Volumes I, V, and VI of *Inside Mac*. Volume I contains a description of a majority of the Toolbox routines used in this book. Volume V contains color QuickDraw information that also affects the Window and Menu Managers. Volume VI is your authoritative reference to System 7.

Buy Volumes I, V, and VI with your lunch money. Buy Volumes II through IV with somebody else's lunch money.

Those of you who have been getting by with your 1 megabyte floppy-based Mac Plus should grit your teeth and fork over the cash for some memory and a hard drive. This book assumes you'll be using System 7 with THINK C: you'll need (at least!) a Mac Plus or Classic with 2 megabytes of RAM and a few megabytes of space on a hard drive.



This book uses THINK C version 5. Get this version. Version 4 of THINK C works somewhat differently than version 5 and, more importantly, doesn't work as well with System 7. If you're not sure how to put THINK C on your Mac's hard drive, read Chapter 2 for the installation procedure.

Finally, use the latest system files with *Mac Primer* programs. We will be working with System 7 throughout the book, although we will occasionally discuss possible code workarounds for computers still using System 6. Since Apple no longer supports System 6 for its new Macs, you should get on the stick and upgrade if you plan to do development work. Certainly, don't use any system software older than version 6.07.



The compiled, standalone programs developed in this book won't work on Macs that don't support System 6, specifically the 512K and the 128K Macs. Since these models are museum pieces, you probably don't have to worry about supporting them when writing Mac programs. They represent less than a few percent of the Macintoshes out there, in any case.

---

## Ready, Set . . .

---

When you finish this book, you'll be able to create your own Macintosh applications.

Get all your equipment together, take the phone off the hook, and fire up your Mac.

Go!

---

# Setting Up

*This chapter introduces you to the software tools used in this book. It also examines some issues that are specific to the implementation of C on the Macintosh.*

---

THINK C IS THE programming environment we'll use throughout the *Macintosh Primer*. First, we'll show you how to install it. Then, we'll look at how to type in and run a sample program. We'll talk about the programming conventions used in this book and some of the rules you need to follow when using the Mac and THINK C together.

If you've already installed THINK C on your Mac, skip to the next section of this chapter.

---

## Installing THINK C 5

---

Before you copy THINK C onto your hard drive, there are a few preparations you'll need to make. For starters, make backup copies of the four floppy disks that came with your copy of THINK C. Tuck your original disks back into the box and use the backup floppies for the actual installation.

Next, delete any old versions of THINK C you might have on your hard drive. *Make sure you don't delete any of your source code or project files.* You may want to place these in an out-of-the-way folder. Once you've installed the new version of THINK C, you can move your source code files back into place.

Now, create a folder named `Development` at the top level of your hard drive. This folder will contain all of your source code, as well as all of the files that make up THINK C.

Finally, make sure you have at least 5 megabytes (5Mb) of free space on your hard drive. Once you've freed up enough space on your hard drive, you're ready to go.

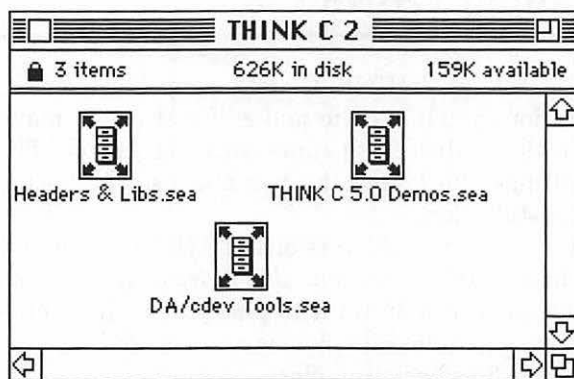


If you don't have 5Mb available on your hard drive, don't panic. The instructions in this chapter show you how to install the seven pieces that make up the full THINK C development environment. Page 22 of the *THINK C User Manual* tells you what's in each of the seven pieces, so you can leave out pieces you may not need right away. If there's any way you can come up with the 5Mb, though, install the whole shebang.

Insert the disk labeled THINK C Disk 2 into your floppy disk drive. A window similar to the one shown in Figure 2.1 should appear. Each of the three files shown in Figure 2.1 is known as a **self-extracting archive**.



An archive is a file containing other files, but in a compressed format. A self-extracting archive is an archive that knows how to convert the archived files from their compressed format back into their normal format. Self-extracting archives are what enable 5Mb of THINK C to fit onto four 800K floppy disks.



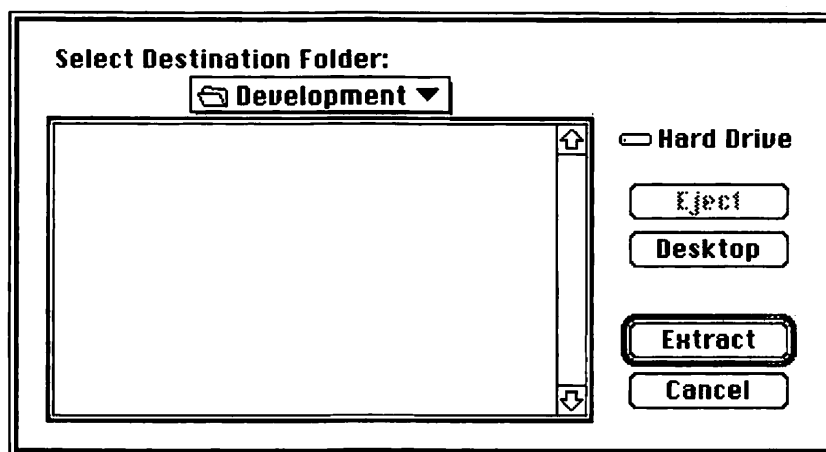
**Figure 2.1** THINK C Disk 2.

Double-click on the self-extracting archive labeled `Headers & Libs.sea`. A dialog box similar to the one in Figure 2.2 will appear. The dialog is asking you where you'd like to place the files extracted from the archive. Use the normal Macintosh file-navigation techniques to guide the dialog box inside your newly created `Development` folder. Make sure the name `Development` appears in the pop-up menu at the top of the dialog box, as it does in Figure 2.2.

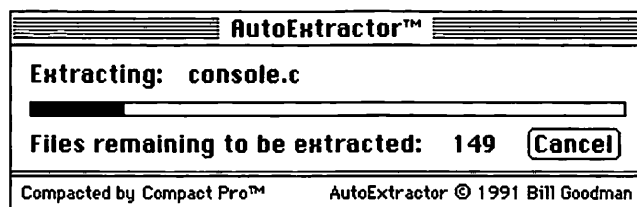
Once inside the `Development` folder, press the **Extract** button. An `AutoExtractor` window will appear, telling you how many files remain to be extracted (Figure 2.3). Once all of the files in the archive are extracted, move on to the next archive.

Double-click on the self-extracting archive labeled `THINK C 5.0 Demos.sea`. Again, a dialog box similar to the one in Figure 2.2 will appear. Just as you did before, guide the dialog into the `Development` folder, making sure the name `Development` appears in the pop-up menu toward the top of the dialog box. Now click the **Extract** button. An `AutoExtractor` window will appear. Once all the files from `THINK C 5.0 Demos.sea` are extracted, the window will disappear.

Repeat this process to extract all of the files from the third self-extracting archive on this disk, `DA/cdev Tools.sea`.



**Figure 2.2** Click the **Extract** button to save the compressed files in the Development folder.



**Figure 2.3** A self-extracting archive in action.

Eject the disk labeled THINK C Disk 2, and place the floppy labeled THINK C Disk 3 into the floppy drive. A window similar to the one shown in Figure 2.4 should appear. The window contains two self-extracting archives. First, double-click on the archive named THINK Class Library 1.1.sea. When the familiar dialog box appears, move into the Development folder and press the **Extract** button.

Once the files are extracted from THINK Class Library 1.1.sea, double-click on the archive named TCL 1.1 Demos.sea. When the dialog box appears, move into the Development folder and press the **Extract** button.

Once the files are extracted from TCL 1.1 Demos.sea, eject the disk labeled THINK C Disk 3, and place the floppy labeled THINK C Disk 4 into the floppy drive. A window similar to the one shown in Figure 2.5 should appear. The window contains two self-extracting

archives. Double-click on the archive named Resource Utilities.sea. When the dialog box appears, move into the Development folder and press the **Extract** button.

Once the files are extracted from Resource Utilities.sea, double-click on the archive named THINK C Utilities.sea. When the dialog box appears, move into the Development folder and press the **Extract** button. OK, one more disk to go!

Eject the disk labeled THINK C Disk 4, and place the floppy labeled THINK C Disk 1 into the floppy drive. A window similar to the one shown in Figure 2.6 should appear. Drag the files THINK C 5.0 and THINK C Debugger 5.0 into the THINK C 5.0 Folder located inside the Development folder.

That's it! Congratulations, you've just completed the grueling THINK C 5 installation process.

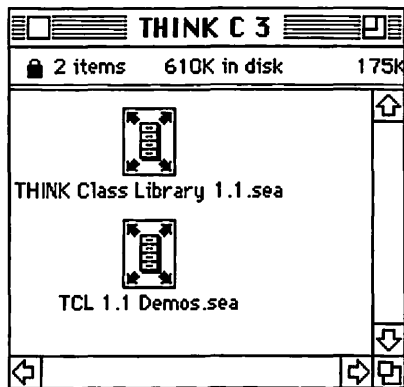


Figure 2.4 THINK C Disk 3.

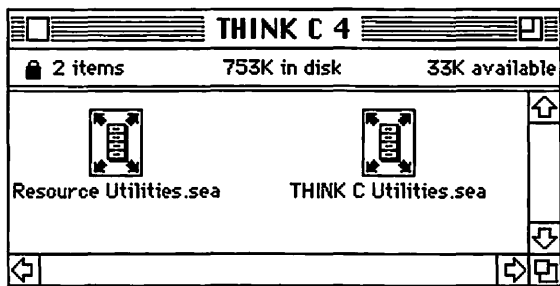


Figure 2.5 THINK C Disk 4.

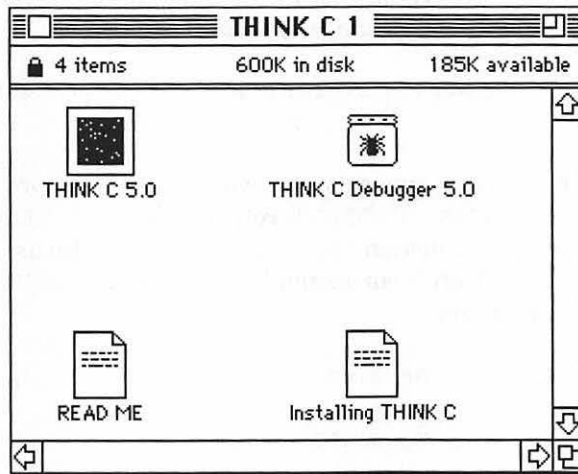


Figure 2.6 THINK C Disk 1.



These instructions were specifically designed to help you install THINK C 5. If you are installing anything other than THINK C 5, read the instructions inside your *THINK C User Manual*.

---

## Accessing the Toolbox with C

---

Built into every Macintosh, regardless of model, is a set of more than 1,400 routines, collectively known as the Mac Toolbox. These include routines for drawing windows on the screen, routines for handling menus, even routines for changing the date on the real-time clock built into the Mac. The existence of these routines helps explain the consistency of the Mac user interface. Everyone uses these routines. When you pull down a menu in Claris MacDraw, you're calling a Toolbox routine. When you pull down a menu in Deneba's Canvas, you're calling the same routine. That's why the menus look alike from application to application, which has a rather soothing effect on users. This same principle applies to scroll bars, windows, lists, dialog boxes, alerts, and so on.

If you look at Toolbox calls in the pages of *Inside Macintosh*, you'll notice that the calling sequences and example code presented in each

chapter are written in Pascal. For instance, the calling sequence for the function `GetNewWindow()` (I:283) is listed as:

```
FUNCTION GetNewWindow (windowID: INTEGER; wStorage: Ptr;
                      behind: WindowPtr) : WindowPtr;
```

Each calling sequence starts with either the word **FUNCTION** or the word **PROCEDURE**. **FUNCTIONs** return values; **PROCEDURES** don't. In the example, the function `GetNewWindow()` returns a value of type `WindowPtr`. Here's an example of a call to `GetNewWindow()` from within a program:

```
WindowPtr    myNewWindow, myOldWindow;
Ptr          myStorage = nil;
int          myWindowID = 400;

myNewWindow = GetNewWindow( myWindowID, myStorage,
                          myOldWindow );
```

In the Pascal calling sequence, the function `GetNewWindow()` returns a value of type `WindowPtr`. In our code, we receive the value returned by `GetNewWindow()` in the variable `myNewWindow`, which is declared as a `WindowPtr`. Most of the data types found in *Inside Macintosh* are automatically available to you in THINK C. (If you're feeling adventurous, check out the folder `Mac #includes` inside the THINK C 5.0 Folder, where all of these types are defined.) The exceptions are summarized in the following table:

<i>Pascal Data Type</i>	<i>C Equivalent</i>
INTEGER	short
LONGINT	long
CHAR	short
BOOLEAN	Boolean

For example, the Pascal **BOOLEAN** data type corresponds to the THINK C data type **Boolean**. The Pascal calling sequence for the `Button()` function can be found on (I:259):

```
FUNCTION Button : BOOLEAN;
```

Here's an example of a call to `Button()` in C:

```
Boolean isButton;

isButton = Button();
if ( isButton == TRUE)
    SysBeep( 20 );
```



Although Pascal is not case-sensitive, C is: `Boolean` and `BOOLEAN` are different. THINK C provides this Pascal type as a convenience to the programmer. Even though `Button()` has no parameters, you must use the parentheses. If you forget them, your program won't compile correctly.

You can also pass literals directly as parameters. For example, our call to `GetNewWindow()` can be rewritten as:

```
WindowPtr    newWindow, oldWindow;

myNewWindow = GetNewWindow( 400, nil, oldWindow );
```

This code will work fine. Passing literals as parameters, however, doesn't necessarily make for readable code. At the very least, we suggest limiting your literal parameters to `#defined` constants. This brings up the next topic.

## **#include, #define, and extern statements**

The `#include` statement tells the C compiler to substitute the source code in the specified file in place of the `#include` statement. Here's an example:

```
#include "BigFile.h"
```

The `#define` statement tells the C compiler to substitute the second argument for the first argument throughout the rest of the source code file. For example:

```
#define MAXFILES 20
```

Most C compilers use two passes to compile source code. During the first pass through a source code file, the compiler pulls in `#include` files and performs all `#define` substitutions. The actual compilation occurs during the second pass through the source code.

`extern` is a C key word used in variable and procedure declarations. Here's an example of an `extern` variable declaration:

```
extern Boolean  done;
```

This `extern` declaration doesn't cause any space to be allocated for the variable `done`. Instead, references to `done` inside the `extern` declaration's file are replaced with pointers to the "real" declaration of `done`:

```
Boolean  done;
```

The absence of the `extern` keyword tells the compiler to allocate space for the variable and tie all the `extern` references to the variable to this allocated space. In the code of this book, each program keeps its source code in a single file, so `extern` is not used. Once your programs reach a certain size, you'll want to break your source into multiple files (for example, the `WorldClock` program in Chapter 5 could easily be broken down into three or four files).

## C and Pascal Strings

C and Pascal use different techniques to implement their basic string data types. Pascal strings start with a single byte, called the **length byte**, which determines the length of the string. For example, in Pascal, the string "Hello, world!" would be stored as a single byte with the value 13, followed by the 13 bytes containing the string:

13	H	e	l	l	o	,		W	o	r	l	d	!
----	---	---	---	---	---	---	--	---	---	---	---	---	---

The C version of this string starts with the 13 bytes containing the string and is terminated with a single byte with the value 0:

H	e	l	l	o	,		W	o	r	l	d	!	0
---	---	---	---	---	---	--	---	---	---	---	---	---	---

The Macintosh Toolbox uses Pascal strings, embodied by the `Str255` data type. Since 1 byte can only hold values from 0 to 255, Pascal strings can be at most 255 bytes in length (not counting the length byte).

Using Pascal strings in THINK C is easy. The THINK C compiler will automatically convert C strings that start with the characters "\p" to Pascal format. Consider the calling sequence for the Toolbox routine `DrawString()` (I:172):

```
PROCEDURE DrawString( s: Str255);
```

You can call `DrawString()` in C like this:

```
DrawString( "\pHello, world!" );
```

You can also use the two routines `CToPstr()` and `PtoCstr()` to translate between C and Pascal formats. These routines are provided as part of THINK C. They are not part of the Macintosh Toolbox.

## Passing Parameters: When to Use the &

Here are the rules to guide your use of the & operator in Toolbox calls:

1. If a parameter is declared as a VAR parameter in the Pascal calling sequence, precede it by an &. Here's the Pascal calling sequence for GetFNum() (I:223):

```
PROCEDURE GetFNum( fontName: Str255; VAR theNum:
INTEGER );
```

Here's a C code fragment that calls GetFNum():

```
short myFontNumber;

GetFNum( "\pGeneva", &myFontNumber );
```

2. If the parameter is bigger than 4 bytes (as is the case with most Pascal and C data structures), precede it by an & whether or not it is declared as a VAR parameter. Here's the Pascal calling sequence for UnionRect() (I:175):

```
PROCEDURE UnionRect( src1, src2: Rect; VAR dstRect:
Rect );
```

Here's a C fragment that calls UnionRect():

```
Rect src1, src2, dstRect;

.
. /* assign values to src1 and src2 */
.

UnionRect( &src1, &src2, &dstRect );
```



If you're wondering where Rect came from, it's one of the data structures defined in *Inside Macintosh* (I:141). A Rect holds the upper left and lower right points of a rectangle. We'll see more of these "predefined" Mac data structures later.

3. If the parameter is 4 bytes or smaller and is not declared as a VAR parameter, pass it without the &. This rule applies even if the parameter is a struct. This is the Pascal declaration of the routine PtToAngle() (I:175):

```
PROCEDURE PtToAngle( r: Rect; pt: Point; VAR angle:
                    INTEGER );
```

Here's a C fragment that calls PtToAngle():

```
Rect      r;
Point     pt;
short     angle;

.
.  /* assign values to r and pt */
.

PtToAngle( &r, pt, &angle );
```

Notice that pt was passed without a leading &. This is because Points are only 4 bytes in size. Most of the predefined Mac types are larger than 4 bytes. Point is one of the few exceptions.

4. If the parameter is a Str255, do not use an &, even if the parameter is declared as a VAR. This is the Pascal declaration of the routine GetFontName() (I:223):

```
PROCEDURE GetFontName( fontNum: INTEGER; VAR theName:
                      Str255 );
```

Here is an example of a Pascal string used in a Toolbox call:

```
Short     fontNum;
Str255    fontName;

fontNum = 4;

GetFontName ( fontNum, fontName );
```

Note that fontName was passed without a leading &.

## Conventions

The purpose of any standard is to ensure consistency and quality. With that in mind, we present our standard for writing C code. We use this standard and feel comfortable with it. Feel free to use your own standard or adapt ours to your own personal style. Most important is to pick a standard and stick with it.

When discussing (as in, arguing over) C standards, people fight most over indentation style. Here's an example of our indentation standard:

```
main()
{
    int i;

    for ( i=0; i<10; i++ )
    {
        DoNastyStuff();
    }
    WrapItUp();
}

DoNastyStuff()
{
    DoOneNastyThing();
}
```

Notice that all of our curly brace pairs (`{` with its corresponding `}`) line up in the same column. Some people like to put the open curly brace at the end of the previous line, like this:

```
main() {
    int i;

    for ( i=0; i<10; i++ ) {
        DoNastyStuff();
    }

    DoNastyStuff() {
        DoOneNastyThing();
    }
}
```

**Hmmmm.**

Well, do what you like, but be consistent.

Another standard adopted by the *Mac Primer* concerns the naming of variables and routines. Generally, we name our variables and routines according to the standards in *Inside Macintosh*. This means that the names look like Pascal names. The advantage of this is that you can use the same variable names used by *Inside Macintosh*. This makes your code much easier to debug and compare with *Inside Macintosh*. Our general rules for variable and routine names are as follows:

- If you're naming a variable, start with a lower-case letter and capitalize the first letter of every subsequent word. This yields variables named `i`, `myWindow`, and `bigDataStructure`.
- If you're naming a global variable, start the name with a lower-case `g`. This yields variables named `gCurrentWindow` and `gDone`.
- If you're creating a `#define`, start the name with a lower-case `k`. This yields `#defines` like this one:

```
#define kNumRecords 20
```

- If you're naming a routine (function, procedure, subroutine, and so on), start with a capital letter and capitalize the first letter of every subsequent word. This yields routines named `MainLoop()`, `DeleteEverything()`, and `PutThatDown()`.



Adherence to a good set of standards will make your code more robust and easier to maintain. Why? Most of the Toolbox routines are built right into the Macintosh in read-only memory, or ROM. The original Macintosh came with 64K of ROM; the Mac Plus comes with 128K of ROM; the Mac SE, II, and IIx have 256K of ROM. The Quadra series has a massive 1 megabyte of ROM. Many of the routines built into the newer Macs are not found in the original Mac, Mac Plus, or SE. Likewise, many routines found in the Mac Plus were not found in the original Macintosh. The point is, things change. If you carefully follow Apple's programming guidelines, the program you write on today's machine will continue to work on tomorrow's.

## ResEdit

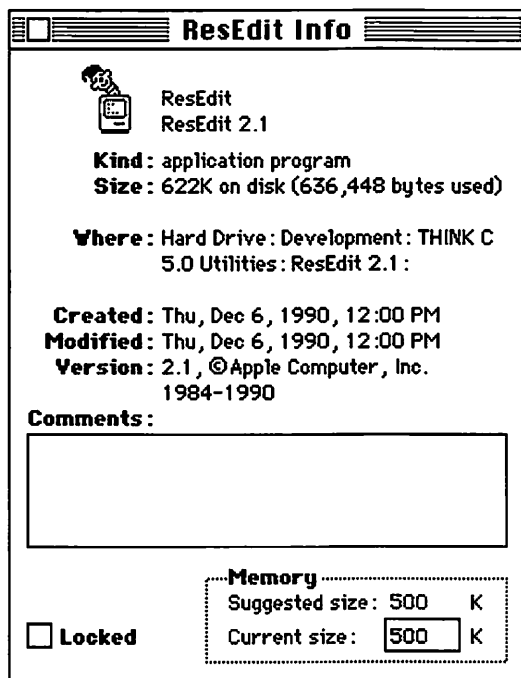
Inside your `Development` folder, in the folder labeled `THINK C 5.0 Utilities`, you'll find a folder containing `ResEdit`, the popular

resource editor from Apple. In the Finder, click on the ResEdit icon, then select **Get Info** from the **File** menu. A ResEdit Info window should appear, similar to the one in Figure 2.7. Make sure you are working with ResEdit version 2.1 or later.

It's a good idea to increase the amount of memory ResEdit uses to edit resources. In the lower-right corner of the ResEdit Info window, increase the **current size**: to at least 1500K to avoid memory problems.

ResEdit is free, and you can usually find the latest version on your favorite BBS. If you purchase ResEdit from the Apple Programmers and Developers Association (APDA), you'll also receive additional documentation. See Chapter 9 for more information about APDA. ResEdit versions consistently improve, so use the latest version you can find.

You'll make extensive use of ResEdit as you create and customize your program's resources. The next section explores some of the resources you'll be working with in this book.



**Figure 2.7** The ResEdit Info window.

## Resources

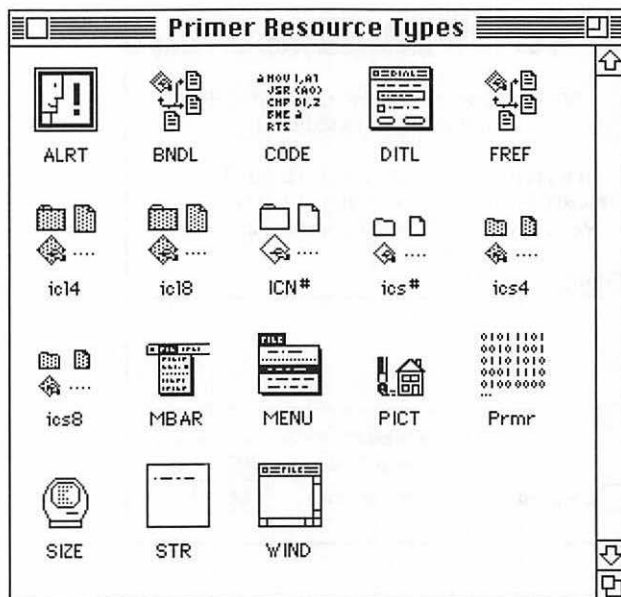
As we mentioned in Chapter 1, much of a program's descriptive information is stored in resources. Resources may be defined by their type and either their resource ID number or their name.

Each resource has a certain type, and each type has a specific function. For example, the resource type `WIND` contains the descriptive information needed to create a window; `MENU` resources describe the menus at the top of the screen. Figure 2.8 shows some of the resource types you'll see in this book.

Each resource type comprises four characters. Case is *not* ignored: `WIND` and `wind` are considered different resource types. Occasionally, resource types may include a space. For example, `'STR '`, where the fourth character is a space.

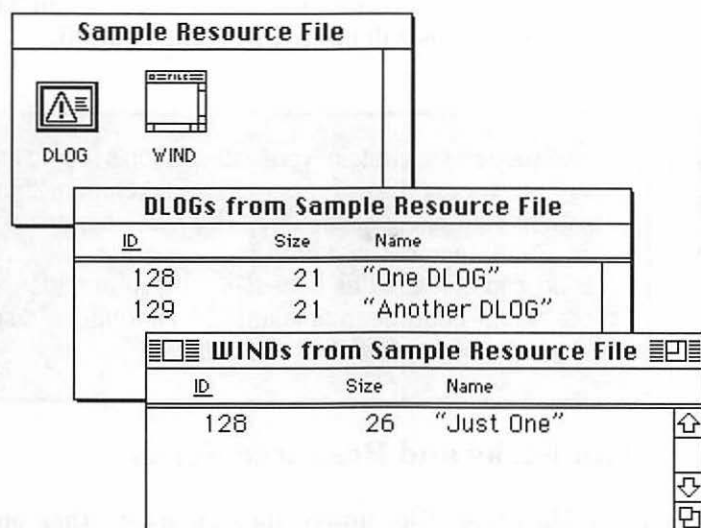


Actually, resource types are just `long ints` (4 bytes) represented in ASCII format. Usually, the types are selected so the ASCII version is readable (like `WIND`, `MENU`, and so on).



**Figure 2.8** Some resource types used in this book.





**Figure 2.9** Sample resource file with two DLOGs and one WIND resource.

Resources of the same type, residing in the same file, must have unique resource IDs. For example, an application may have several resources of type DLOG, as long as each DLOG has a unique resource ID. The resource file shown in Figure 2.9 contains two DLOGs, one with an ID of 128, and one with an ID of 129. The file also contains a single WIND resource with an ID of 128. Thus, each resource is uniquely identified by ID number and type.

If you prefer, you may also name your resources. Each of the three resources shown in Figure 2.9 has a type, an ID, and a name. All of the examples presented in the *Mac Primer* use the resource type and resource ID to uniquely specify a resource. When you create your resources, however, you might want to specify resource names, as well as resource IDs. This will make your resource files easier to read in ResEdit.



Resource ID numbers follow these conventions:

Range	Use
-32,768 to -16,385	Reserved by Apple
-16,384 to 127	Used for system resources
128 to 32,767	Free for use

In certain situations, there may be additional restrictions placed on resource IDs; check *Inside Macintosh* for more information.

In this book, CODE resources will be created in THINK C; most of the other resources will be created using ResEdit.



CODE resources contain your application's compiled object code. You may be used to an operating environment that allows you to segment your executable code. The Mac supports segmentation as well. Each segment is stored in a separate CODE resource and is loaded and unloaded as necessary. If you are interested in learning more about code segmentation, an informative discussion begins on page 97 of the *THINK C User Manual*.

## Data Forks and Resource Forks

Each Macintosh file, unlike files on most other operating systems, contains two parts: a data fork and a resource fork. The resource fork stores the resources, and the data fork contains everything else. Some word processors store a document's text in the document's data fork and use the resource fork for storing the document's formatting information. HyperCard stacks, interestingly enough, have most of their information on the data fork side. The THINK C projects in this book will use the resource fork exclusively.

Now that we've covered these weighty and important topics, let's get on with the fun stuff: our first THINK C program.

---

## The Hello, World Program

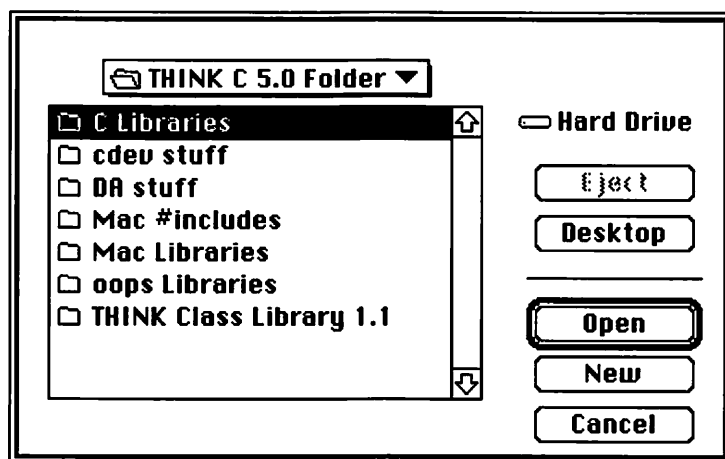
---

Our first program is a classic you may have encountered before: Hello draws the text "Hello, world!" in a window on the screen.

Just to keep things orderly, create a folder named Hello inside your Development folder. Keep each of the files associated with the Hello project inside this folder.

### Create a New Project

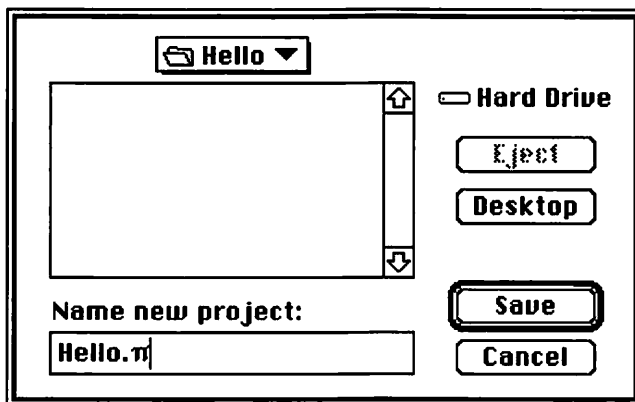
To create your first program, double-click on the THINK C application in the THINK C 5.0 Folder. The first thing you'll see is the Open Project dialog box (Figure 2.10).



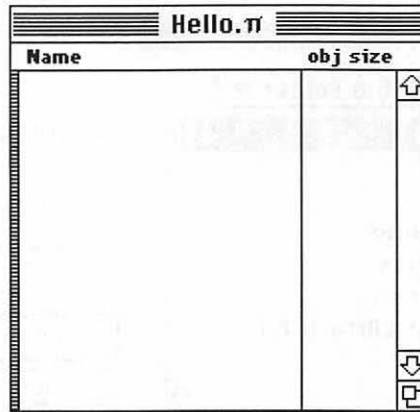
**Figure 2.10** The Open Project dialog box.

Click on the **New** button. The Name Project dialog box will appear (Figure 2.11). Use the standard Macintosh mechanisms to guide the dialog into the Hello folder you just created (move up once to the Development folder, and down once more into the Hello folder). Type `Hello.π` (key **Option-p** for  $\pi$ ) in the **Name new project:** field, then click the **Save** button.

When you click **Save**, the project window (titled `Hello.π`) will appear (Figure 2.12). As you add files to your project, their names will appear in the project window. At this point, the project window is empty because you have not yet added any files to the project.



**Figure 2.11** The Name Project dialog box.



**Figure 2.12** The Hello.π project window.



The project file acts as an information center for all files involved in building an application. In addition, the project file contains information about the THINK C environment, such as the preferred font and font size for displaying and printing source code. Projects are a THINK C concept, not a Macintosh concept.

As you compile your source code, the object code generated will be stored in the project file as a `CODE` resource. The **obj size** column in the project window reflects the current size of the object code associated with each file in the project. An uncompiled file will have an **obj size** of 0.



As you may have noticed, we've managed to sneak in another naming convention. This one came directly out of the THINK C *User Manual*. To stay true to THINK C, name your source code files `xxx.c`, your project files `xxx.π`, and your resource files `xxx.π.rsrc`. The `π` character is created by keying **Option-p**.

Now, you're ready to type in your first program.

## The Code

Pull down the **File** menu and select **New**. An untitled source code window will appear. Type in the following program:

```
/****** Hello.c *****/  
  
#include <stdio.h>  
  
main()  
{  
    printf( "Hello, world!" );  
}
```

The THINK C compiler doesn't care how you use white space, such as tabs, blanks, and spaces. Be generous with your white space—don't be afraid to throw in a blank line or two if it will improve the readability of your code.

Check the code for typing errors. If everything looks all right, select **Save As...** from the **File** menu. Save the file as `Hello.c`. Next, select **Add** (make sure you select **Add**, *not* **Add...**) from the **Source** menu to add the file `Hello.c` to the project. **Add** adds the frontmost window to the project, whereas **Add...** allows you to select one or more files to add to the project.



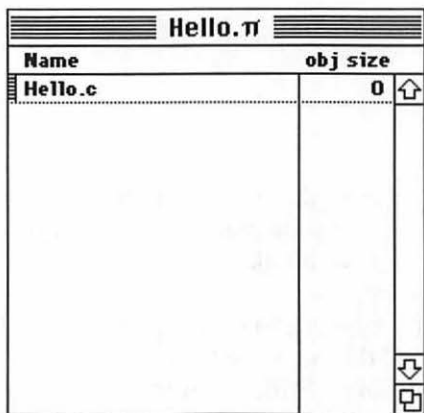
A common mistake at this point is to save the file as `Hello.c.` (note the period after the `c`), instead of as `Hello.c` (without the trailing period). To repeat: source code files should be named `xxx.c`, project files should be named `xxx.π`, and resource files (when we get to them in Chapter 3) should be named `xxx.π.rsrc` and that's it. Periods are *not* used at the end of any file names in this book.

## Running Hello, World

Note that as soon as you added `Hello.c` to the project, the name `Hello.c` appeared in the project window (Figure 2.13). Since `Hello.c` has not been compiled yet, its **obj size** is 0. Try running the program by choosing **Run** from the **Project** menu, or by keying **⌘R**. Respond to the **Bring the project up to date?** dialog box by clicking the **Yes** button.



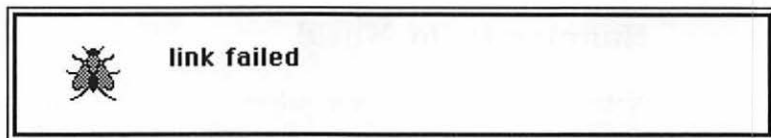
For readers who have a tendency to get depressed if you get an error on the computer: The first time you run this program, it's not going to work. It's OK, it's not your fault. We'll show you why in a moment.



**Figure 2.13** Hello.c appears in the project window.

THINK C will now compile Hello.c. If the compiler encounters an error, it will do its best to describe the problem to you. If you make any typing mistakes, correct them, then type **⌘R** again.

Once the program compiles correctly, THINK C will try to link your code. Basically, THINK C is trying to make sure that all of the functions you call are available somewhere in the project. When a link fails, THINK C displays a **link failed** error message (Figure 2.14). At the same time, THINK C creates a Link Errors window, telling you which functions could not be resolved (Figure 2.15).



**Figure 2.14** The link failed error dialog box.

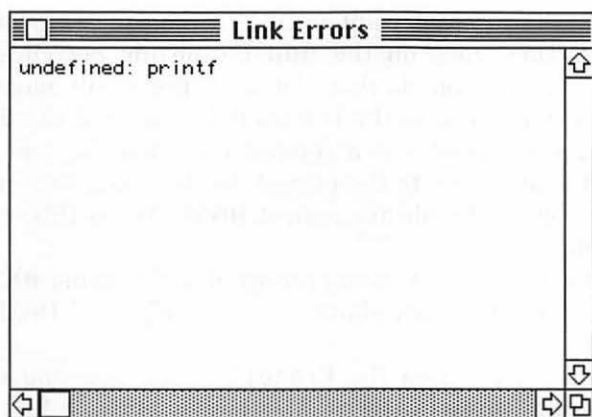


Figure 2.15 The Link Errors window.

In this case, THINK C couldn't find the function `printf()`. Don't worry—this is a simple problem to fix. You need to add the library containing `printf()` to your project.

Click anywhere on the screen to make the **link failed** dialog box disappear. Next, select **Add...** from the **Source** menu. The Add Files dialog box will appear. The library containing `printf()` is located in the C Libraries folder. To get there from the Hello folder, move up once to get to the Development folder, down into the THINK C 5.0 Folder, and down again into the C Libraries folder (Figure 2.16).

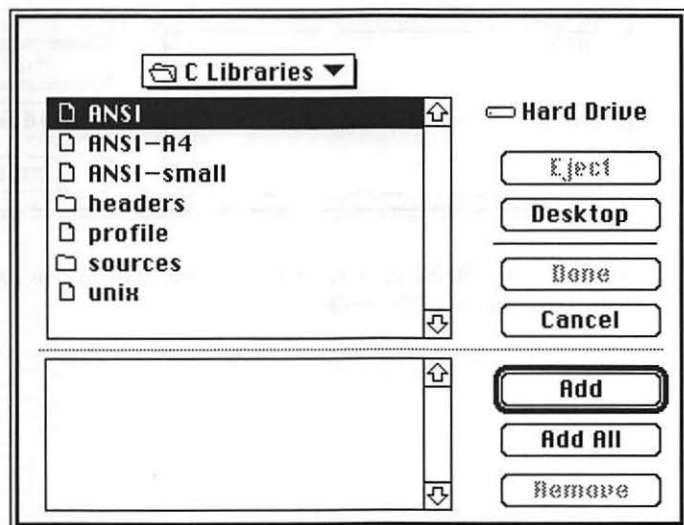
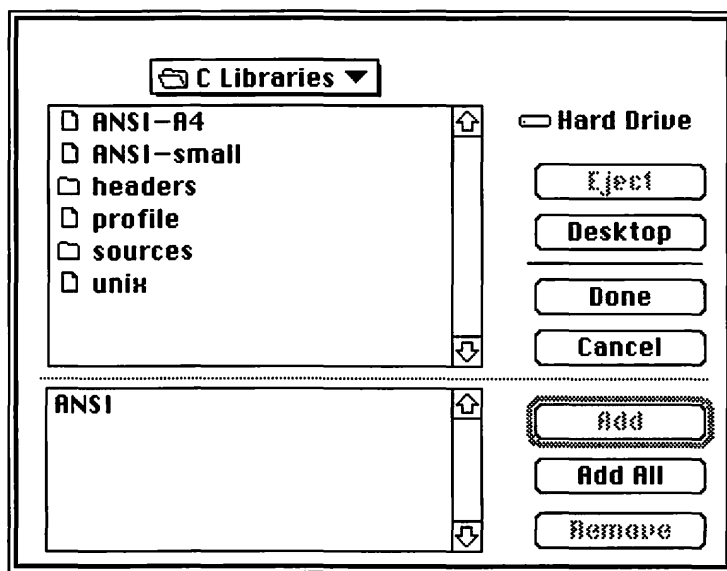


Figure 2.16 The Add Files dialog box.

The library you want to add to the project is called **ANSI**. Select **ANSI**, then click on the **Add** button (do *not* click on the **Add All** button). Once you do this, the name **ANSI** will move from the top half of the dialog box to the bottom half (Figure 2.17). The bottom half of the dialog box acts as a staging area, allowing you to add more than one file at a time to the project. In this case, we'll just add one file to the project, the library named **ANSI**. To do this, click on the **Done** button.

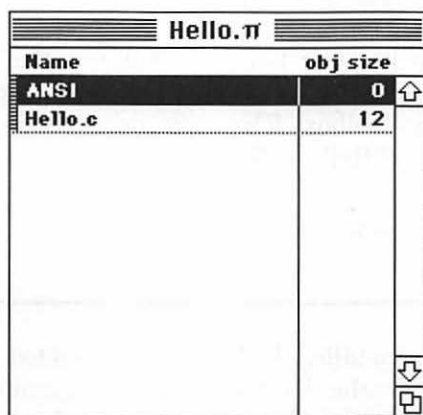
You'll know you were successful if the name **ANSI** appears in the project window, just above `Hello.c` (Figure 2.18). Now the project is complete.

Select **Run** from the **Project** menu. Respond to the **Bring the project up to date?** dialog box by clicking **Yes**. First, THINK C will load the **ANSI** library's object code. Then, assuming everything



**Figure 2.17** **ANSI** is now in the bottom half of the Add Files dialog box, ready to be added to the project.



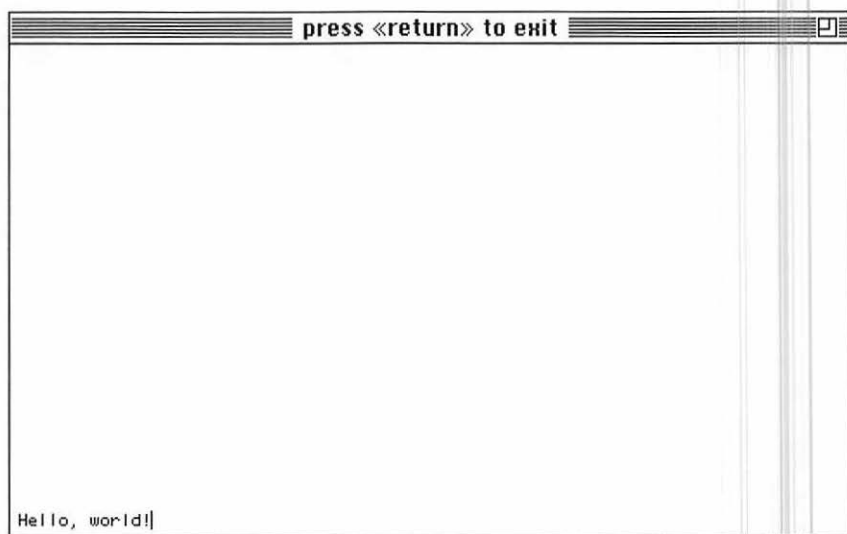


Name	obj size
ANSI	0
Hello.c	12

**Figure 2.18** ANSI has been successfully added to the project.

else went smoothly, THINK C will run the program. A window should appear on the screen, containing the text `Hello, world!` (Figure 2.19).

To exit the program, either type **return** or select **Quit** from the **File** menu.



**Figure 2.19** Hello.π in action.

## **The Problem with Hello, World**

We don't want to get you too excited about this version of Hello. Although it does illustrate how to use THINK C, it does not make use of the Macintosh Toolbox. The first program in Chapter 3 is a Macintized version of Hello, called Hello2.



## **In Review**

---

In Chapter 2, you installed THINK C and created your first project. Chapter 3 looks at the basics of Mac programming: QuickDraw, windows, and resources. It also presents four applications that demonstrate the versatility of the Macintosh.

It's almost too late to turn back. To all of those who have come from other environments: beware! QuickDraw is addictive!

# Drawing on the Macintosh

*Now that you have installed THINK C,  
you can start programming. A good  
starting point is the unique routines  
that define the Macintosh graphic  
interface. On the Macintosh, the  
Toolbox routines that are responsible  
for all drawing are known collectively  
as QuickDraw.*

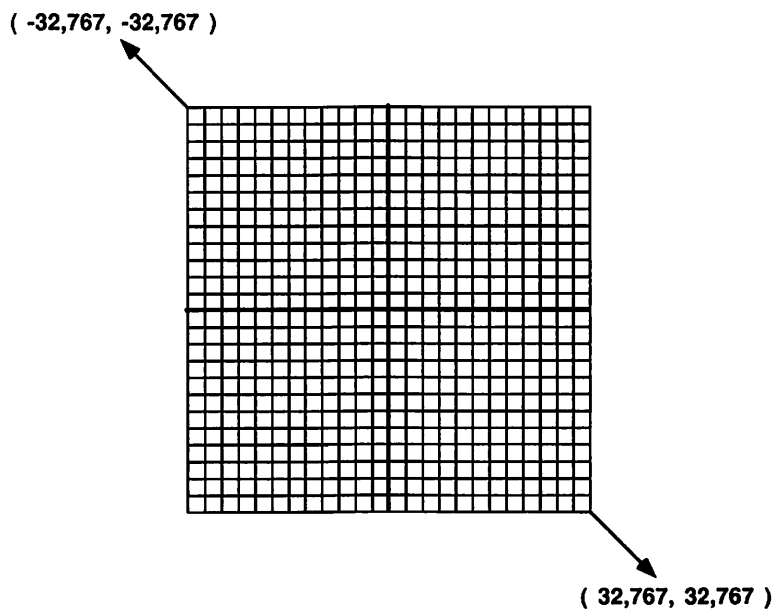
## Introduction

**QUICKDRAW** IS THE **MACINTOSH** drawing environment. With it, you can draw rectangles and other shapes and fill them with different patterns. You can draw text in different fonts and sizes. The windows, menus, and dialogs displayed on the Macintosh screen are all created using QuickDraw routines.

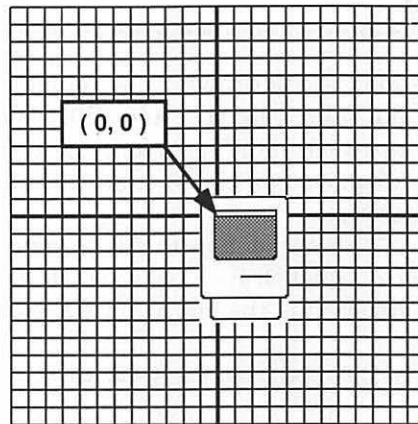
In this chapter, we'll show you how to create your own windows and draw in them with QuickDraw. Let's start by examining the QuickDraw coordinate system, the mathematical basis for QuickDraw.

### The QuickDraw Coordinate System

QuickDraw drawing operations are all based on a two-dimensional grid coordinate system. The grid is finite, running from  $(-32,767, -32,767)$  to  $(32,767, 32,767)$ , as shown in Figure 3.1.



**Figure 3.1** The grid.



**Figure 3.2** The Macintosh screen on the grid.

Every Macintosh screen is actually an array of pixels aligned to the grid. The lines of the grid surround the pixels. The grid point labeled (0,0) is just above, and to the left of, the upper left-hand corner of the Mac screen (Figure 3.2).



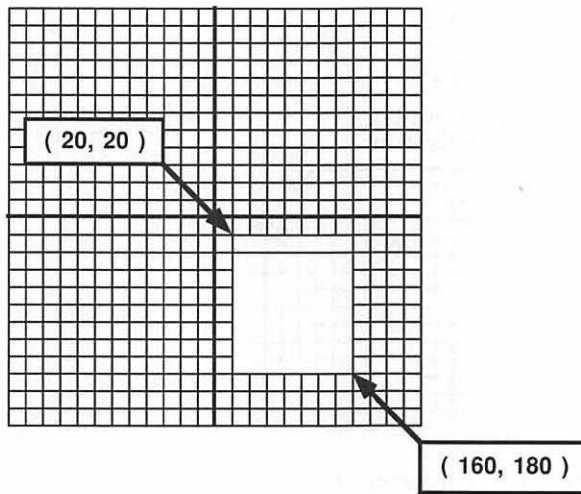
A screen measuring 32,768 pixels x 32,768 pixels with a screen resolution of 1 pixel = 1/72 inch would be 38 feet wide and 38 feet tall. The Mac Classic, Plus, and SE monitors are 512 x 342 pixels. Apple's Mac 13" color monitor is 640 x 480 pixels.

The grid is also referred to as the **global coordinate system**. Each window defines a rectangle in global coordinates. Every rectangle has a top, left, bottom, and right. For example, the window in Figure 3.3 defines a rectangle whose left is 20, top is 20, right is 160, and bottom is 180.

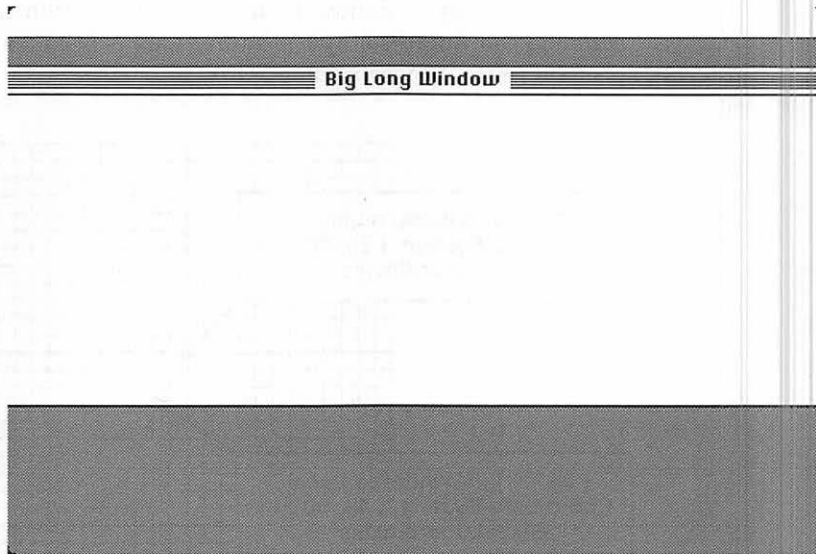


Interestingly, the window does not have to be set up within the boundaries of the screen. You can set up a window whose left is -50, top is 100, bottom is 200, and right is 800. On a Classic, this window would extend past the left and right sides of the screen (Figure 3.4). This is known as the Big Long Window Technique.

Use of the Big Long Window Technique is discouraged.

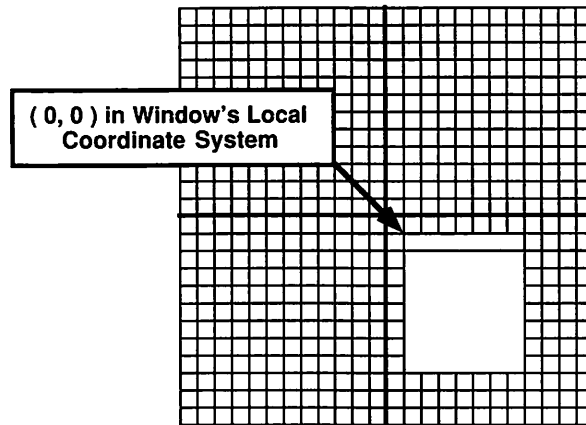


**Figure 3.3** A window on the grid.



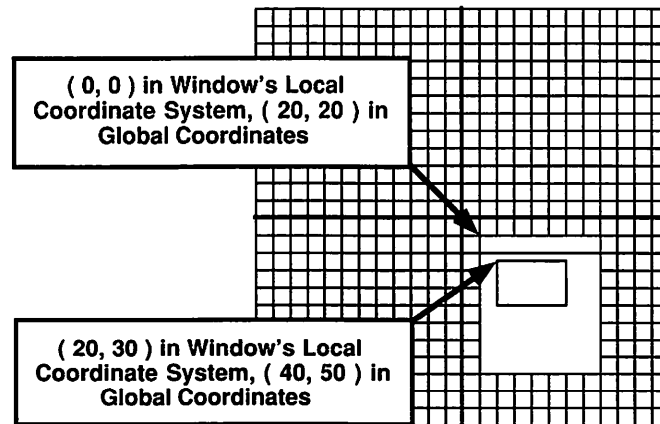
**Figure 3.4** A big, long window.

When drawing inside a window, you'll always draw with respect to the window's **local coordinate system**. The upper left-hand corner of a window lies at coordinate (0,0) in that window's local coordinate system (Figure 3.5).



**Figure 3.5** Local coordinates.

To draw a rectangle inside your window, specify the top, left, bottom, and right in your window's local coordinates (Figure 3.6). Even if you move your window to a different position on the screen, the rectangle coordinates stay the same. This is because the rectangle was specified in local coordinates.

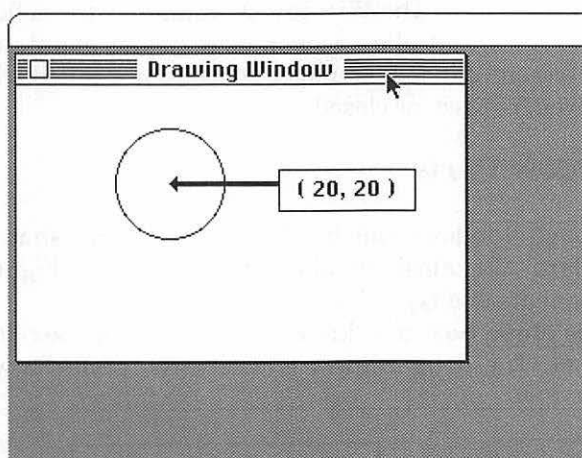


**Figure 3.6** Rectangle drawn in window's local coordinates.

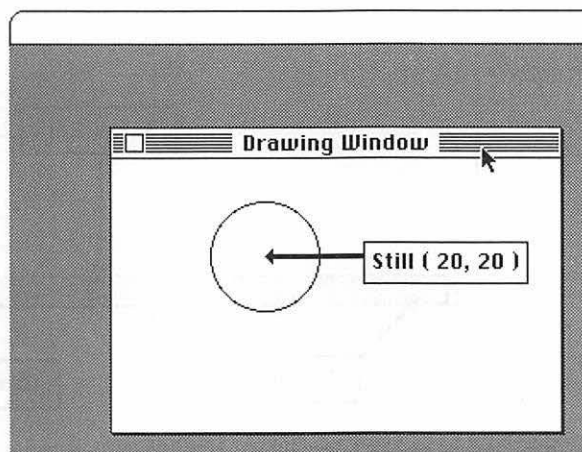


Local coordinates are handy! Suppose you write an application that puts up a window and draws a circle in the window (Figure 3.7). Then, the user of your application drags the window to a new position (Figure 3.8).

You still know exactly where that circle is, even though its window has been moved. That's because you specified your circle in the window's local coordinates.



**Figure 3.7** Circle drawn in window's local coordinates.



**Figure 3.8** When window moves, local coordinates stay the same.



On the Macintosh, text and graphics created by your programs will be displayed in **windows**. Windows are the devices that Macintosh programs use to present information to a user.

Because we need windows to draw in, let's look more closely at windows and the Window Manager.

---

## Window Management

---

When you draw graphics and text on the Macintosh, you draw them inside a window. The **Window Manager** is the collective name for all the routines that allow you to display and maintain the windows on your screen. Window Manager routines are called whenever a window is moved, resized, or closed.

### Window Parts

Although windows can be defined to be any shape you choose, the standard Macintosh window is rectangular. Figure 3.9 shows the components of a typical window.

The **close box** (also known as the go-away box) is used to close the window. The **drag region** is where you grab the window to move it

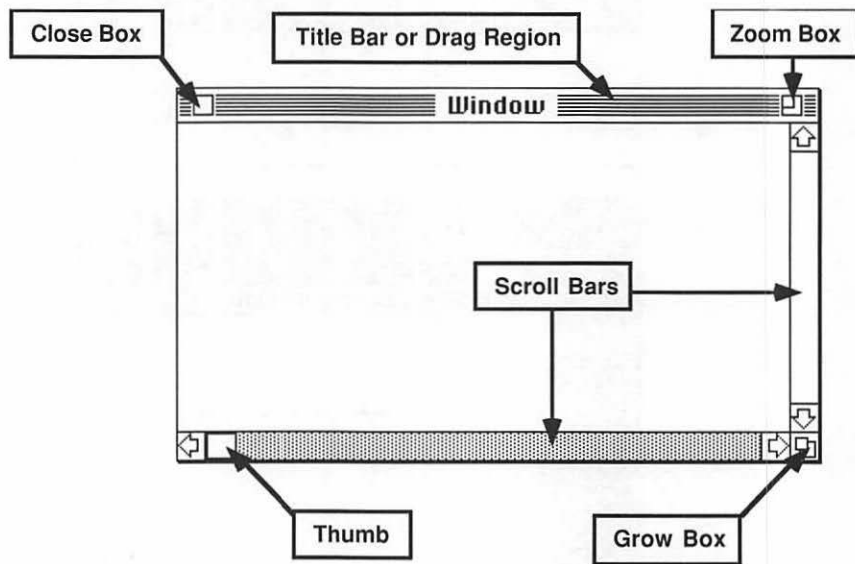


Figure 3.9 Window components.

around the screen; this region also contains the window's title. **Scroll bars** are used to examine contents of the window not currently in view. The **thumb** is dragged within the scroll bar to display the corresponding section of the window. The **grow box** (also known as the **size box**) lets you resize the window. The **zoom box** toggles the window between its standard size and a predefined size, normally about the size of the full screen.

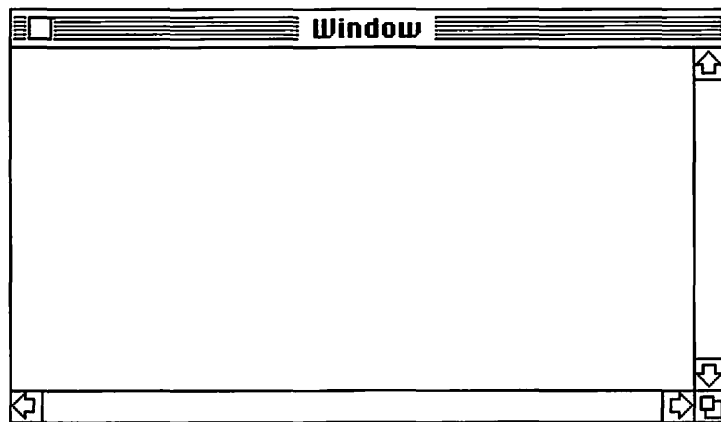
There are several types of windows. The window in Figure 3.9 is known as a **document window**. When you use desk accessories or print documents, you will notice other kinds of windows. These windows may not have all of the same components as the standard window, but they operate the same way.

## Window Types

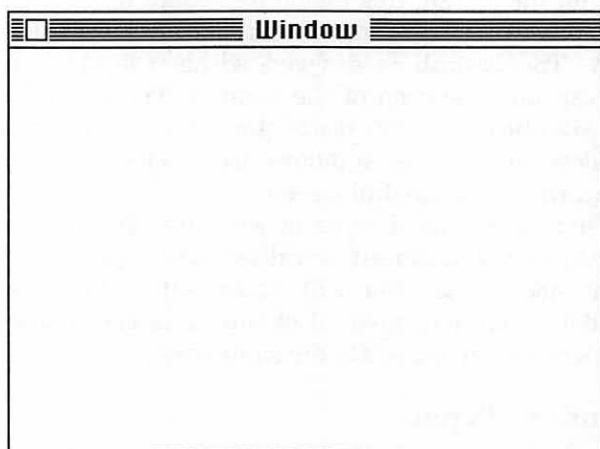
Six standard types of windows are defined by the Window Manager. Each type has a specific use. In this section, each type is described and its use is discussed.

The `documentProc` window (Figure 3.10) is the standard window used in applications. This one has a size box, so it is resizable; it also has a close box in the upper left-hand corner that closes the window.

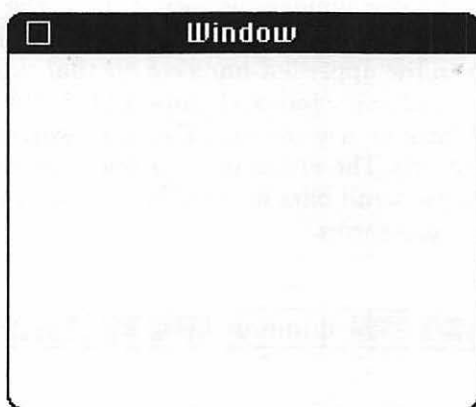
The `noGrowDocProc` window (Figure 3.11) is the standard window without scroll bars or a grow box. Use this window for information that has a fixed size. The `rDocProc` window (Figure 3.12) has a black title bar; it has no scroll bars or grow box. This window is most often used with desk accessories.



**Figure 3.10** The `documentProc` window.

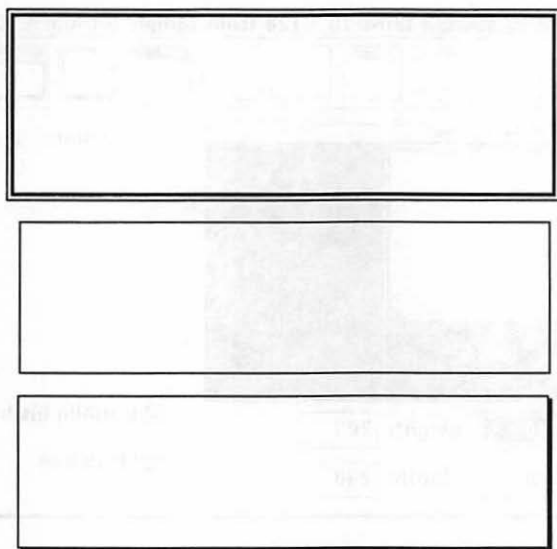


**Figure 3.11** The noGrowDocProc window.



**Figure 3.12** The rDocProc window.

The remaining three types of windows are dialog box windows: `dBoxProc`, `plainDBox`, and `altDBoxProc` (Figure 3.13). Dialog boxes will be discussed in Chapter 6.



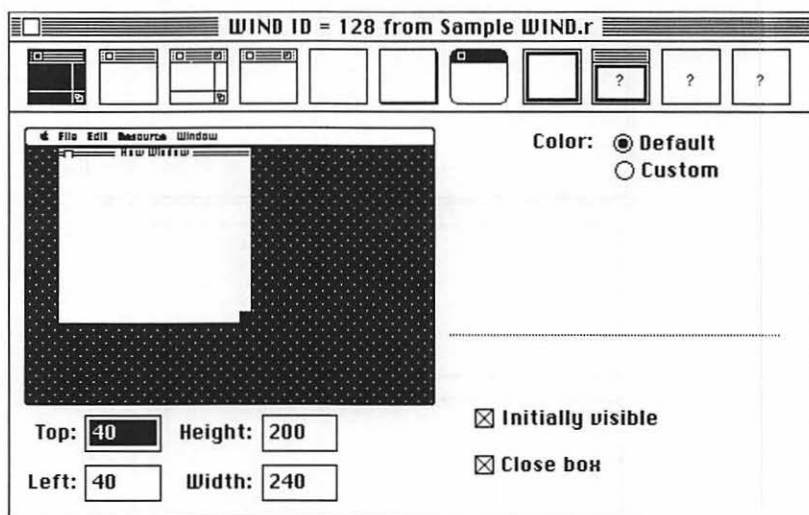
**Figure 3.13** The dBoxProc, plainDBox, and altDBoxProc windows.



The windows described here are the standard models. You can customize them by adding a few options. For example, the documentProc, noGrowDocProc, and rDocProc window types can come either with or without the close box. A zoom box can be added to documentProc and noGrowDocProc windows (see Chapter 4). We'll show you everything you need to know to create exactly the type of window you want for your application.

## Setting Up a Window for Your Application

If you plan to use one of the standard window designs for your applications, creating a window is easy. First, build a WIND resource using ResEdit (we'll show you how a little later in the chapter). Figure 3.14 shows ResEdit's WIND editor. Like all resources, each WIND has its own unique resource ID. As you'll see, this resource ID is used to fetch the WIND resource from the resource file. Later in the chapter, we'll walk through the WIND creation process in detail.



**Figure 3.14** ResEdit's WIND editor.

Once your WIND resource is built, you're ready to start coding. One of the first things your program will do is initialize the Toolbox. The Window Manager is initialized at this point.

Next, load your WIND resource from the resource file, using the `GetNewWindow()` Toolbox routine:

```
short      windowID;
Ptr        wStorage;
WindowPtr  window, behind;

window = GetNewWindow( windowID, wStorage, behind);
```

`GetNewWindow()` loads the WIND resource that has a resource ID of `windowID`. The WIND information is stored in memory at the space pointed to by `wStorage`. The Window Manager will automatically allocate its own memory if you pass `nil` as your `wStorage` parameter. For now, this technique is fine. As your applications get larger, you'll want to consider developing your own memory management scheme.

The parameter `behind` determines whether your window is placed in front of or behind any other windows. If the value is `nil`, the new window is placed behind the rest of your application's windows. If

(WindowPtr)-1L is passed as the third parameter, the new window appears in front of all other windows. For example:

```
window = GetNewWindow( 400, nil, (WindowPtr)-1L );
```

loads a WIND with a resource ID of 400, asks the Window Manager to allocate storage for the window record, and puts the window in front of all other windows. A pointer to the window data is returned in the variable window.



The expression (WindowPtr)-1L is a typecast, asking the compiler to convert the constant -1L (a long with a value of -1) to the type WindowPtr before passing it as a parameter. Depending on the options you have set for your projects, THINK C may or may not require you to typecast your parameters to match the type of the receiving parameter. The programs in this book were designed to work with THINK C's factory settings. For more information on the THINK C options dialog, see page 179 in the *THINK C User Manual*.

When you create the WIND resource with ResEdit, you are given a choice of making the window visible or not. Visible windows appear as soon as they are loaded from the resource file with GetNewWindow(). If the visible flag is not set, you can use ShowWindow() to make the window visible:

```
ShowWindow( window );
```

where window is the pointer you got from GetNewWindow(). Most applications start with invisible windows and use ShowWindow() when they want the window to appear. The Window Manager routine HideWindow() makes the window invisible again. In general, you'll use ShowWindow() and HideWindow() to control the visibility of your windows.

At this point, you've learned the basics of the Window Manager. You can create a WIND resource using ResEdit, load the resource using GetNewWindow(), and make the window appear and disappear using ShowWindow() and HideWindow(). This technique will be illustrated shortly. After you have put up the kind of window you want, you can start drawing in it. The next section shows you how to use QuickDraw routines to draw in your window.

## Drawing in Your Window: The QuickDraw Toolbox Routines

There are many QuickDraw drawing routines. They can be conveniently divided into four groups: routines that draw lines, shapes, text, or pictures. These routines do all of their drawing using a graphics “pen.” The pen’s characteristics affect all drawing, whether the drawing involves lines, shapes, or text.

Before starting to draw, you have to put the pen somewhere (`MoveTo()`), define the size of the line it will draw (`PenSize()`), choose the pattern used to fill thick lines (`PenPat()`), and decide how the line you are drawing changes what’s already on the screen (`PenMode()`). Figure 3.15 shows how changing the graphics pen changes the drawing effect.

Every window you create has its own pen. The location of a window’s pen is defined in the window’s local coordinate system. Once a window’s pen characteristics have been defined, they will stay defined until you change them.

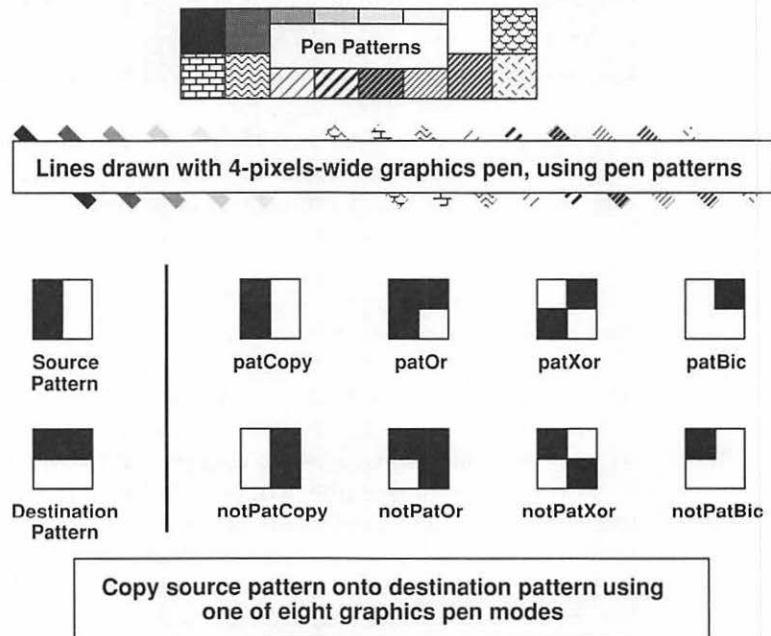


Figure 3.15 Graphics pen characteristics.

## Setting the Current Window

Because your application can have more than one window open at a time, you must first tell QuickDraw which window to draw in. This is done with a call to `SetPort()`:

```
window = GetNewWindow( 400, nil, (WindowPtr)-1L );  
SetPort( window );
```

In this example, `SetPort()` made `window` the current window. Until the next call to `SetPort()`, all QuickDraw drawing operations will occur in `window`, using `window`'s pen. Once you've called `SetPort()` and set the window's pen attributes, you're ready to start drawing.



The basic data structure behind all QuickDraw operations is the `GrafPort`. When you call `SetPort()`, you are actually setting the current `GrafPort` (I:271). Since every window has a `GrafPort` data structure associated with it, in effect you are setting the current window. The `GrafPort` data structure contains fields such as `pnSize` and `pnLoc`, which define the `GrafPort` pen's current size and location. QuickDraw routines such as `PenSize()` modify the appropriate field in the current `GrafPort` data structure.

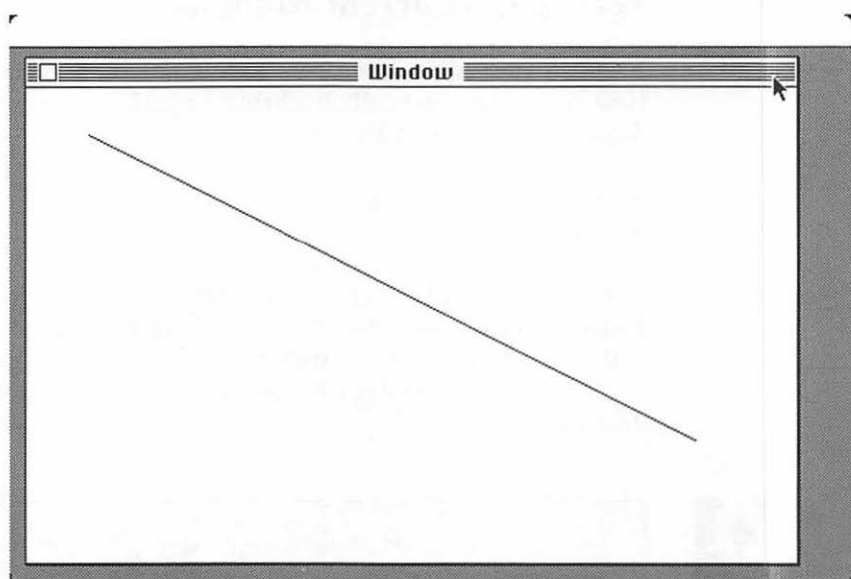
## Drawing Lines

The `LineTo()` routine allows you to draw lines from the current pen position (which you have set with `MoveTo()`) to any point in the current window. For example, a call to:

```
window = GetNewWindow( 400, nil, (WindowPtr)-1L );  
  
SetPort( window );  
  
MoveTo( 39, 47 );  
LineTo( 407, 231 );
```

would draw a line from (39, 47) to (407, 231) in `window`'s local coordinate system (Figure 3.16).





**Figure 3.16** Drawing a line with QuickDraw.



It is perfectly legal to draw a line outside the current boundary of a window. QuickDraw will clip it automatically so that only the portion of the line within the window is drawn. QuickDraw will keep you from scribbling outside of the window boundaries. This is true for all QuickDraw drawing routines.

The last program in this chapter is the *FlyingLine*, an extensive example of what you can do using the QuickDraw line-drawing routines.

## Drawing Shapes

QuickDraw has a set of drawing routines for each of the following shapes: rectangles, ovals, rounded-corner rectangles, and arcs. Each shape can be filled, inverted, or drawn as an outline (Figure 3.17).

The current pen's characteristics are used to draw each shape where appropriate. For example, the current fill pattern will have no effect on a framed rectangle. The current `PenMode()` setting, however, will affect all drawing. The second program in this chapter, *Mondrian*, shows you how to create different shapes with QuickDraw (Figure 3.18). It also demonstrates the different pen modes.

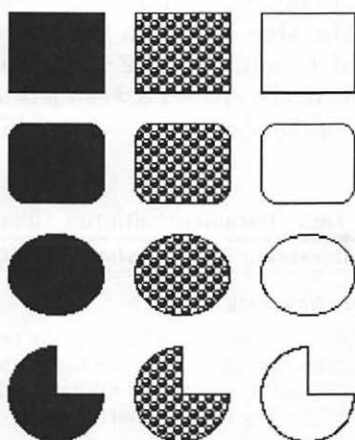


Figure 3.17 Some QuickDraw shapes.

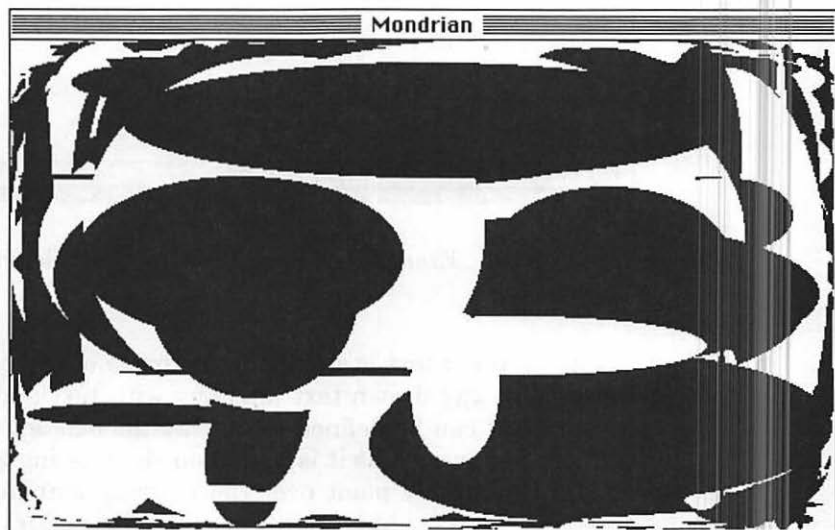


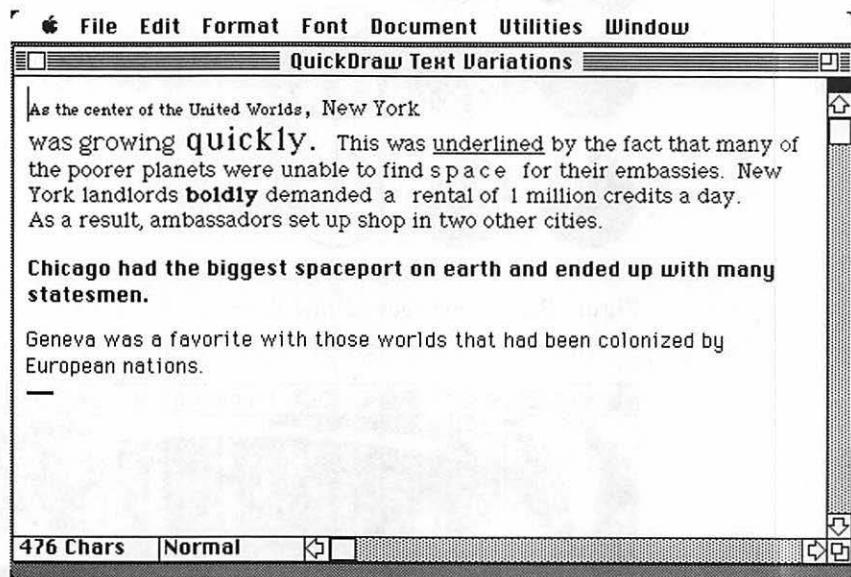
Figure 3.18 Mondrian.

## Drawing Text

QuickDraw allows you to draw different text formats easily on the screen. QuickDraw can vary text by font, style, size, spacing, and mode. Let's examine each of the text characteristics.

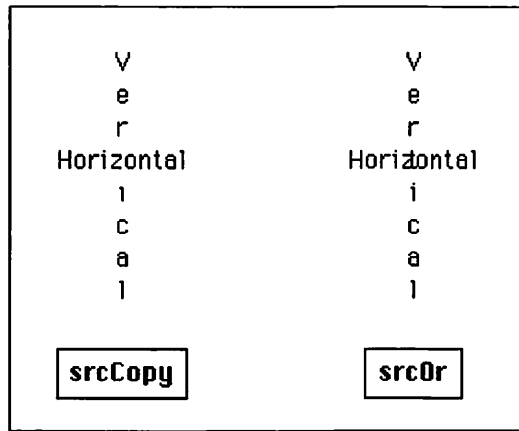
**Font** refers to the typeface of the text you are using. *Courier*, *Helvetica*, and *Geneva* are some of the typefaces available on the

Macintosh. **Style** refers to the appearance of the typeface (**bold**, *italic*, underline, and so on). The **size** of text on the Macintosh is measured in points, where a point is equal to 1/72 inch. **Spacing** defines the average number of pixels in the space between letters on a line of text. Figure 3.19 shows some of the characteristics of QuickDraw text.



**Figure 3.19** Examples of QuickDraw text and derivative science fiction writing.

The **mode** of text is similar to the mode of the pen. The text mode defines the way drawn text interacts with text and graphics already drawn. Text can be defined to overlay the existing graphics (`srcOr`); text can be inverted as it is placed on the existing graphics (`srcXor`); or text can simply paint over the existing graphics (`srcCopy`). The other modes (`srcBic`, `notSrcCopy`, `notSrcOr`, and so on) are described in *Inside Macintosh* (I:157). Figure 3.20 demonstrates the two most popular text modes.



**Figure 3.20** The two most popular QuickDraw modes.

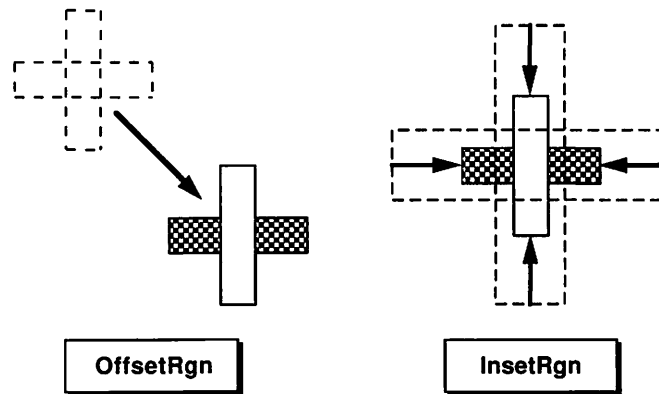
## Drawing Pictures

QuickDraw can save text and graphics created with the drawing routines as picture resources called `PICTs`. You can create a picture (using a program such as Canvas or MacDraw), copy the picture to the clipboard, and paste it into a `PICT` resource using ResEdit. Later in the chapter, you'll see how to make use of `PICT` resources in the ShowPICT program.

## About Regions

QuickDraw allows you to define a collection of lines and shapes as a **region**. You can then perform operations on the entire region (Figure 3.21).

By now most of you are probably itching to start coding. First, let's look at the basic Mac programming structure used in this chapter's programs. Then we'll hit the keyboards!



**Figure 3.21** Two QuickDraw region operations.

## Basic Mac Program Structure

We've looked at a general outline of the QuickDraw and Window routines needed to make a Macintosh application go. The basic algorithm used in each of the Chapter 3 programs goes something like this:

```
main()
{
    ToolboxInit();
    OtherInits();
    DoPrimeDirective();

    while ( ! Button() ) ;
}
```

Like most C programs, our program starts with the routine `main()`, which first initializes the Toolbox. It then takes care of any program-specific initialization, such as loading windows or pictures from the resource file. Next, the program performs its prime directive. In the case of the Hello, World! program, the prime directive is drawing a text string in a window. Finally, the program waits for the mouse button to be clicked. This format is very basic: Except for clicking the button, there is no interaction between the user and the program. This will be added in the next chapter.



Danger! Will Robinson! Normal Macintosh applications do not exit with a click of the mouse button. Mac programs are interactive. They use menus, dialogs, and events. We'll add these features later. For the purpose of demonstrating QuickDraw, however, we'll bend the rules a bit.

---

## The QuickDraw Programs

---

Each of the following programs demonstrates different parts of the Toolbox. The Hello2 program demonstrates some of the QuickDraw routines related to text; Mondrian displays QuickDraw shapes and modes; ShowPICT loads a PICT resource and draws the picture in a window. Finally, you'll code the FlyingLine, an intriguing program that can be used as a screen saver.

Let's look at another version of the Hello, World! program presented in Chapter 2.

---

## Hello2

---

The Hello2 program will do the following:

- Initialize the Toolbox;
- Load a resource window, show it, and make it the current port;
- Draw the text string "Hello, world!" in the window;
- Quit when the mouse button is clicked.

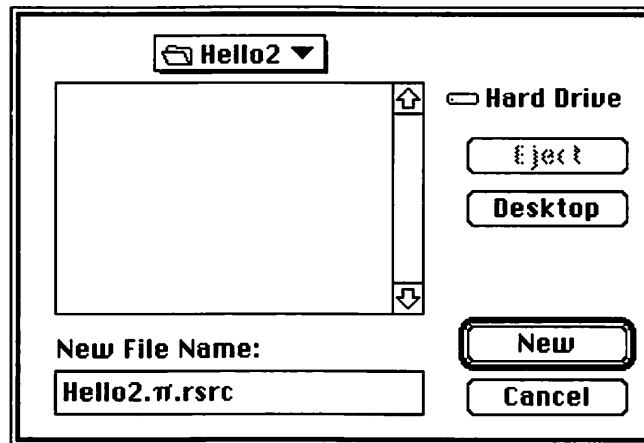
To get started, create a folder inside the Development folder and name it Hello2. This is where you'll build your first Macintosh application. The next few sections will show you how to create the three files you'll need for this project. First, you'll create a resource file to hold Hello2's resources. Next, you'll create a project file, just as you did with Hello2's predecessor in Chapter 2. Finally, you'll create a source code file for Hello2's source code and add the source code file to the project.

## Hello2 Resources

As we discussed in Chapter 2, at the heart of every THINK C program is a project file. A typical project file has a name like `xxx.π`, where `xxx` is the name of the program. When you open a project named `xxx.π`, THINK C automatically looks for a file named `xxx.π.rsrc` and makes any resources in this file available to your program.

In a bit, you'll create a project file named `Hello2.π`. Before you do that, you'll use ResEdit to create a file named `Hello2.π.rsrc` and, inside `Hello2.π.rsrc`, you'll create a single WIND resource.

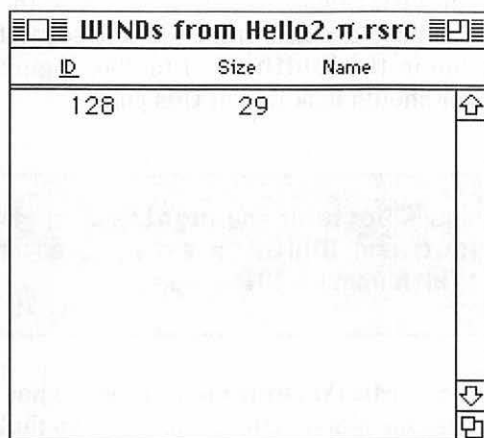
Find your copy of ResEdit (make sure you use version 2.1 or later) and double-click on its icon. Then click the mouse and ResEdit will prompt you for a resource file to open. Click on the **New** button. When the **Save File** dialog box appears, use the standard Mac navigation techniques to move into the `Hello2` folder you just created. Type `Hello2.π.rsrc` in the **New File Name:** field and click the **New** button (Figure 3.22).



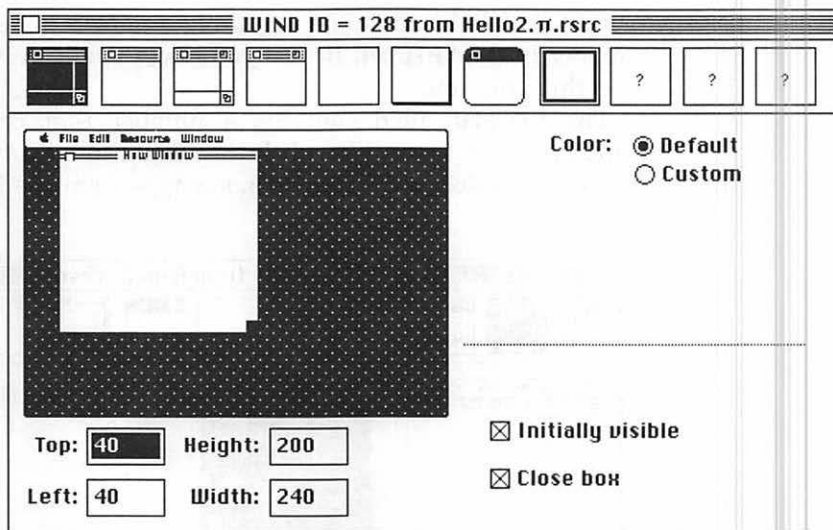
**Figure 3.22** Create a new resource file in the `Hello2` folder.

ResEdit will create an empty resource file named `Hello2.π.rsrc` and open a window listing of all that file's resources. Since the file is empty, no resources are listed. You're about to change that.

Select **Create New Resource** from the **Resource** menu. When prompted to select a resource type, select WIND from the scrolling list and click **OK** (you could also have typed in WIND and clicked **OK**). Two new windows should appear, a window listing all of the WIND resources (Figure 3.23) and, on top of that, a window showing the newly created WIND (Figure 3.24).



**Figure 3.23** ResEdit's list of WIND resources for Hello2.π.rsrc.



**Figure 3.24** ResEdit's WIND editor.

The WIND resource you've just created will act as a template for the Hello2 program, telling Hello2 the type and size of window in which to display the Hello, World! text.

The icons toward the top of the WIND editor allow you to select the window's type. Click on the second icon from the left, choosing the window with no grow box and no zoom box.



Next, edit the **Left:** field, changing its value to 5. This will cause the window to appear 5 pixels from the left edge of the screen. Also, change the value in the **Width:** field to 300. Figure 3.25 shows what your **WIND** editor should look like at this point.

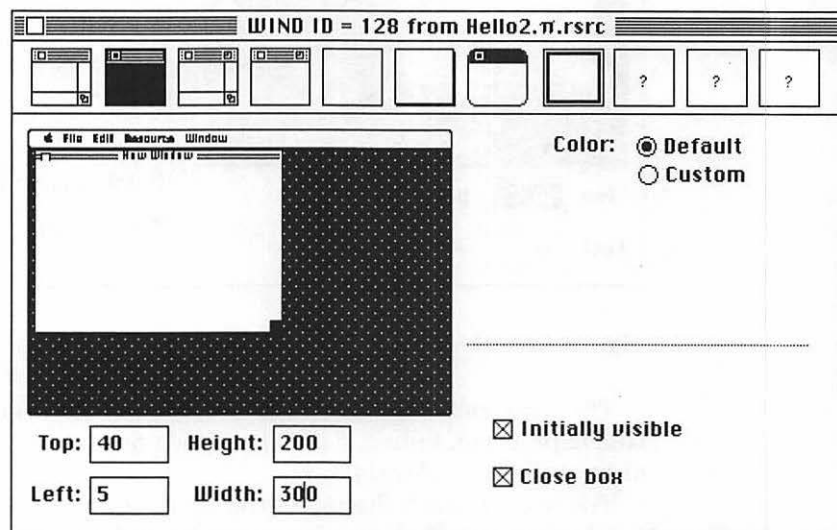


If the field names **Bottom:** and **Right:** appear instead of the field names **Height:** and **Width:** in the **WIND** editor, select **Show Height & Width** from the **WIND** menu.

You can use the **MiniScreen** menu to see what your window will look like on various Macintosh screens. By default, ResEdit shows your **WIND** on a Mac Classic screen (512 pixels wide, 342 pixels tall).

Next, select **Set 'WIND' Characteristics...** from the **WIND** menu (Figure 3.26). Change the **Window title:** to *My First Window*. The **refCon:** field is a 4-byte integer reserved for use by your application. You can use this field for anything you want. Feel free to type a number in the **refCon:** field if you like, though we won't make use of it in this program.

The **ProcID:** field contains a number that corresponds to the window's type. If you like, click the **OK** button, returning to the **WIND** editor, and select a different window type from the icons at the top of



**Figure 3.25** The **WIND** editor, after a few changes have been made.

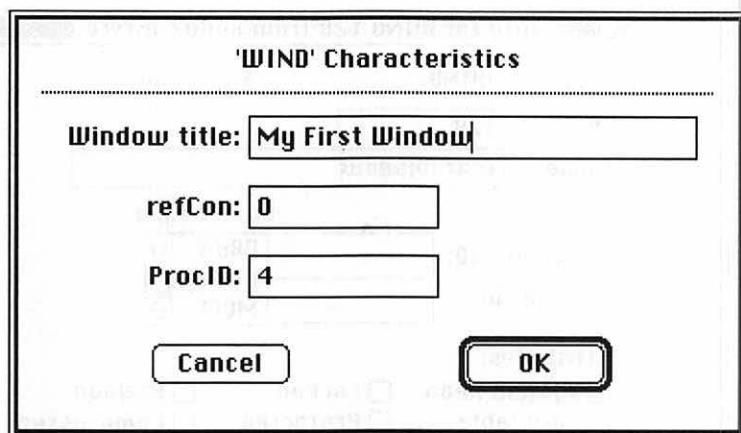


Figure 3.26 The 'WIND' Characteristics dialog box.

the window. Go back to the 'WIND' Characteristics window and check out the **ProcID:** field. Notice the change? Make sure you change the **ProcID:** back to 4, then click the **OK** button to return to the WIND editor.



The number found in the **ProcID:** field can be derived from the pages of *Inside Macintosh* (I:273), which contains a list of constants corresponding to the different WIND types available from the Window Manager. The WIND we just created used the constant `noGrowDocProc`, which has a value of 4. To add a zoom box to one of these window types, add 8 to the value of its constant. For example, to add a zoom box to a `noGrowDocProc`, change the constant from 4 to 12.

For the moment, leave **ProcID:** with a value of 4. You'll learn how to add a zoom box to your windows in Chapter 4.

Next, select **Preview at Full Size** from the **WIND** menu. A model of your window will appear in the proper position on the screen. Click the mouse to make the test window disappear.

Next, select **Get Resource Info** from the Resource menu. When the resource information window appears (Figure 3.27), make sure the WIND's **resource ID** is set to 128. The resource ID is the number you'll pass to `GetNewWindow()` to retrieve this WIND from the resource file.

Info for WIND 128 from Hello2.π.rsrc

Type: WIND Size: 36

ID: 128

Name: Text Window

Owner type

Owner ID: [ ] DRVR [↑]  
WDEF [ ]  
Sub ID: [ ] MDEF [↓]

Attributes:

☐ System Heap ☐ Locked ☐ Preload  
☒ Purgeable ☐ Protected ☐ Compressed

**Figure 3.27** Resource Info window for WIND 128.

Next, make sure the **Purgeable** checkbox is checked. This allows the Macintosh Memory Manager to purge the WIND resource from memory once it's not needed anymore. This approach maximizes the amount of memory available for your application.

If you like, name your WIND by typing some text in the **Name:** field. While this won't affect your program, assigning a name to a resource can make it easier to tell one WIND resource from another in a list of resources. In general, when you have more than one resource of a given type, assign each resource a name, so you can tell them apart at a glance. Figure 3.28 shows how the name appears in the list of WINDs first shown in Figure 3.23.

ID	Size	Name
128	36	"Text Window"

**Figure 3.28** WIND 128 appears in the list of WIND resources. Notice the resources' name appearing on the right side of the window.

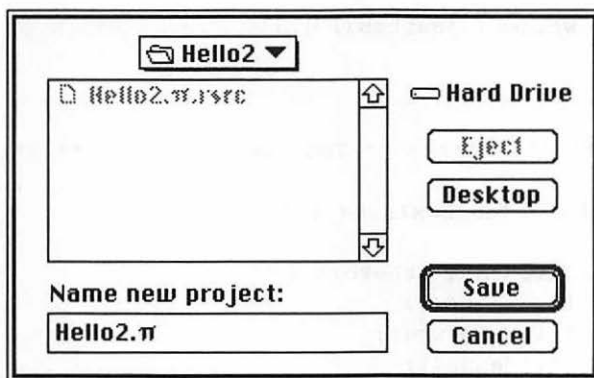
Next, choose **Save** from the **File** menu, saving the changes you've made to the resource file. Finally, choose **Quit** from the **File** menu. Now you're ready to start up THINK C.



Some of you may note that the **Size:** field in Figure 3.27 has a number different from that in the **WIND** you created. There are several explanations for this. You may be using a different version of ResEdit than we are. You may have used a different name for your **WIND** than we did. This does make a difference as far as resource size goes. As long as the basic values in your ResEdit screens match the values in the screen shots, you should be OK.

## The Hello2 Project File

Start up THINK C by opening the **THINK C 5.0 Folder** and double-clicking on the **THINK C 5.0** icon. When the dialog box appears, click on the **New** button. When the next dialog box appears, navigate into the **Hello2** folder and save the new project file as **Hello2.π** (Figure 3.29).



**Figure 3.29** Save the new project file as **Hello2.π**.

Next, select **New** from the **File** menu and type the following source code into the window that appears:

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

#define kHorizontalPixel 30
#define kVerticalPixel   50

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
void    WindowInit( void );

/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();

    while ( !Button() ) ;
}

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;
```

```
window = GetNewWindow( kBaseResID, nil, kMoveToFront );

if ( window == nil )
{
    SysBeep( 10 );    /* Couldn't load the WIND
                      resource!!! */

    ExitToShell();
}

ShowWindow( window );
SetPort( window );

MoveTo( kHorizontalPixel, kVerticalPixel );
DrawString("\pHello, world!");
}
```

When you've typed that in, select **Save As...** from the **File** menu and save your source code (in the **Hello2** folder) as **Hello2.c**. Select **Add** from the **Source** menu to add **Hello2.c** to the project. The name **Hello2.c** should appear in the project window.

Next, you'll need to add a file called **MacTraps** to your project. **MacTraps** is a precompiled file that contains everything your project will need to access the Macintosh Toolbox routines. Select **Add...** from the **Source** menu. Find **MacTraps** inside the **THINK C 5.0 Folder**, inside the **Mac Libraries** folder. In the **Add...** dialog, double-click on **MacTraps** to move it from the top half of the dialog to the bottom half. Once **MacTraps** (and nothing but **MacTraps**) appears in the bottom half of the dialog, click on **Done**.

When you're done, the Project window should look like Figure 3.30.

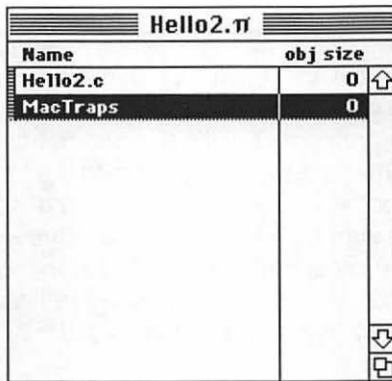


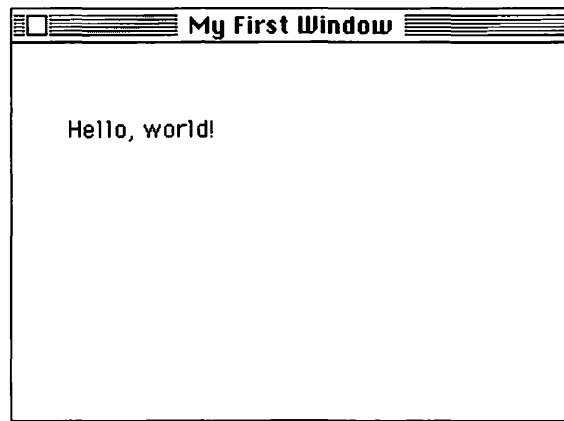
Figure 3.30 The project file after **Hello2.c** has been added.

## Running Hello2

Now you're ready to run Hello2. Select **Run** from the **Project** menu. When asked to **Bring the project up to date?**, click **Yes**. You may get a complaint about a syntax error or two. If so, just retype the line the compiler highlights.

If you make any changes to `Hello2.c`, you'll be asked whether you'd like to **Save changes before running?** Click **Yes**.

Once you've gotten Hello2 to compile without a hitch, it will automatically start running, as shown in Figure 3.31. *Voilà*. Hello2 should display a window with the text `Hello, world!` in it. Quit the program by clicking the mouse button.



**Figure 3.31** Hello2 in action.



If Hello2 compiles, but the Hello2 window fails to appear, it may indicate a problem with the resource file. If you heard a beep when THINK C ran your program, THINK C could not find your resource file. Make sure your project file is named `Hello2.π` and your resource file is named `Hello2.π.rsrc` (no spaces in either name). Also, make sure both files are in the same Hello2 folder.

Another common reason why Hello2 doesn't work is that code font left the `Hello2.π.rsrc` window open in ResEdit. Close and save your resource file before running projects!

## Walking Through the Hello2 Code

We'll be walking through the source code of each of the programs presented in the *Mac Primer*. We'll start with each program's #defines and global variables, then dig into every one of the program's functions.

The first few lines of `Hello2.c` are #defines. THINK C #defines are the same as those found in other C programming environments. During compilation, THINK C takes the first argument of the #define, finds each occurrence in the source code, and substitutes the second argument. For example, in the first #define, the number 128 will be substituted for each occurrence of `kBaseResID`.



#defines don't actually modify your copy of the source code. THINK C creates its own copy of the source code and makes the substitution on its copy.

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

#define kHorizontalPixel 30
#define kVerticalPixel   50
```

The constants `kHorizontalPixel` and `kVerticalPixel` describe, in the window's local coordinates, where the text string will be drawn.

Next come the **function prototypes**. Each program in this book uses function prototypes at the top of each source code file. While not strictly necessary, function prototypes will make your source code easier to read, and will aid in ensuring that each function is called with parameters of the correct type.

```
void    ToolboxInit( void );
void    WindowInit( void );
```

`main()` calls `ToolboxInit()` to initialize the Macintosh Toolbox, then `WindowInit()` to load a window from the resource file then draw some text in the window.



```

/***** main *****/

```

```

void    main( void )
{
    ToolBoxInit();
    WindowInit();

    while ( !Button() ) ;
}

```

`ToolBoxInit()` will remain unchanged throughout the book. Although you won't always use all the data structures and variables initialized by `ToolBoxInit()`, you are perfectly safe in doing so. It is much easier and safer to initialize each of the Macintosh Toolbox managers than to try to figure out which ones you'll need and which you won't.

```

/***** ToolBoxInit *****/

```

```

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

Each call initializes a different part of the Macintosh interface. The call to `InitGraf()` initializes QuickDraw.

`InitFonts()` initializes the Font Manager and loads the system font into memory. Since the Window Manager uses the Font Manager (to draw the window's title, for example), you must initialize fonts first. `InitWindows()` initializes the Window Manager and draws the desktop and the empty menu bar. `InitMenus()` initializes the Menu Manager so you can use menus. (Chapter 5 shows how to use the Menu Manager). `InitMenus()` also draws the empty menu bar.

`TEInit()` initializes TextEdit, the Text-Editing Manager that MiniEdit uses (discussed in the *THINK C User Manual* and in *Inside Macintosh*). `InitDialogs()` initializes the Dialog Manager (demonstrated in Chapter 6). `InitCursor()` sets the cursor to the arrow cursor and makes the cursor visible.



The following global variables are initialized by `InitGraf()` and can be used in your routines:

- `thePort` always points to the current `GrafPort`. Because it is the first QuickDraw global, passing its address to `InitGraf()` tells QuickDraw where in memory all of the other QuickDraw globals are located.
- `white` is a pattern variable set to a white fill; `black`, `gray`, `ltGray`, and `dkGray` are initialized as different shades between black and white.
- `arrow` is set as the standard cursor shape, an arrow. You can pass `arrow` as an argument to QuickDraw's cursor-handling routines.
- `screenBits` is a data structure that describes the main Mac screen. The field `screenBits.bounds` is declared as a `Rect` and contains a rectangle that encloses the main Mac screen.
- `randSeed` is used as a seed by the Macintosh random number generator (we'll show you how to use the random number generator in this chapter).



`InitWindows()` and `InitMenus()` both draw the empty menu bar. This is done intentionally by the ROM programmers for a reason that is such a dark secret they didn't even document it in *Inside Macintosh*.

As we said, it's not necessary to call each of these routines in every program you'll ever write. Why, then, should you call `InitMenus()`, for example, if you don't use menus? Well, suppose you decide to add menus later. Calling `InitMenus()` now means you won't spend time later wondering why your program is crashing when all you did was add a new menu-handling routine.

You may take advantage of an external procedure that does use menus. As we discussed earlier, some managers require the use of information in other managers. If one manager is not initialized, your program may not work. Use the `ToolboxInit()` routine in all your programs.

```

/***** WindowInit *****/

void      WindowInit( void )
{
    WindowPtr      window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

```

`WindowInit()` calls `GetNewWindow()` to load the WIND resource with a resource ID of `kBaseResID` from your resource file. The first parameter specifies the resource ID. The second parameter tells the Toolbox how memory for the new window data structure should be allocated. Because you passed `nil` as the second parameter, the Toolbox will allocate the memory for you. Finally, the third parameter to `GetNewWindow()` tells the Window Manager to create this window in front of any of the application's open windows.

```

if ( window == nil )
{
    SysBeep( 10 );      /* Couldn't load the WIND
                        resource!!! */

    ExitToShell();
}

```

`GetNewWindow()` returns a pointer to the new window data structure in the variable `window`. `GetNewWindow()` will return a value of `nil` if it can't create the window for some reason. `GetNewWindow()` can fail for several reasons. On one hand, since we passed `nil` as the second parameter, we've asked `GetNewWindow()` to allocate memory for the window's data structure. If `GetNewWindow()` can't allocate enough memory to create the window, it will fail and return `nil`.

Even more likely, if `GetNewWindow()` returns `nil`, it's because it couldn't load the WIND resource from the resource file. If this is the case, check to make sure the resource has the proper resource ID, and that the resource file is named correctly.

If `window` is `nil`, both `SysBeep()` and `ExitToShell()` are called. `SysBeep()` will emit a single beep (or whatever sound passes for a beep on your Mac). The value passed to `SysBeep()` is ignored. Make sure you pass a value, though, as `SysBeep()` expects one. `ExitToShell()` will immediately exit the program, returning to whatever program spawned it (in this case, THINK C).

```

ShowWindow( window );
SetPort( window );

```

```
MoveTo( kHorizontalPixel, kVerticalPixel );  
DrawString("\pHello, world!");  
}
```

Next, `WindowInit()` calls `ShowWindow()` to make the window visible. If the WIND resource's **visible** check box was not checked, it is at this point that the window actually appears on the screen. The call to `SetPort()` makes window the current window. All subsequent QuickDraw drawing operations will take place in window. Next, window's pen is moved to the local coordinates 50 down and 30 across from the upper left-hand corner of window, and `DrawString()` is used to draw the string `Hello, world!` starting at the current pen coordinates.



As mentioned earlier, the characters `\p` found at the beginning of a quoted string tell the compiler to generate the string in Pascal form (using the leading length byte), as opposed to C form (with a trailing `nil` byte). Use this technique whenever you pass a quoted string to a Toolbox function that requires an `Str255` parameter.

Hello2 can easily be turned into a standalone application. Pull down the **Project** menu and select **Build Application....** When the Build Application dialog box appears, type `Hello2` in the **Save application as:** field, then click **Save**. THINK C will turn the projects compiled code into a standalone application, copying all of the resources from the project resource file (`Hello2.π.rsrc`) into the application's resource fork. Take your new application out for a test drive by double-clicking its icon in the Finder. You'll find out how to add a custom icon to your applications in Chapter 8.

---

## Variants

This section presents some variants to the Hello2 program. We'll start by changing the font used to draw `Hello, world!`. Next, we'll modify the style of the text, using **boldface**, *italics*, and so on. We'll also show you how to change the size of your text. Finally, we'll experiment with different window types.

## Changing the Font

Every window has an associated font. You can change the current window's font by calling `SetFont()`, passing an integer that represents the font you'd like to use:

```
short    myFontNumber;

SetFont( myFontNumber );
```

Macintosh font numbers start at 0 and count up from there. THINK C has predefined a number of font names with which you can experiment. For example, `monaco` is defined as 4, `times` as 20. If you want to check out the whole list, open the file `Fonts.h` in the Apple `#includes` folder, which is inside the Mac `#includes` folder.

The best way to make use of a specific font is to pass its name as a parameter to the Toolbox routine `GetFNum()`. `GetFNum()` will return the font number associated with that name. You can then pass the font number to `SetFont()`.



Did someone in the back ask, "How can you tell which fonts have been installed in the system?" An excellent question! Not every Mac has the same set of fonts installed. Some folks have the LaserWriter font set; others a set of fonts for their StyleWriter. Some people might even have a complete set of foreign language fonts. For the most part, your applications shouldn't care which fonts are installed. There are, however, two exceptions to this rule. All dialog boxes and menus are drawn in the **system font**, which defaults to font number 0. The default font for applications is called the **application font**, usually font number 1. In the United States, the system font is **Chicago**, and the application font is **Geneva**.

For now, put the `GetFNum()` and `SetFont()` calls before your call to `MoveTo()` and after your call to `SetPort()`, and try different font names (use the Key Caps desk accessory for a list of font names on your Mac). `GetFNum()` will set `fontNum` to 0 if it can't find the requested font. Don't forget to declare `fontNum` at the top of `WindowInit()`.

```

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;
    short        fontNum;

    window = GetNewWindow( kBaseResID , nil,
                          kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND
                           resource!!! */

        ExitToShell();
    }

    ShowWindow( window );
    SetPort( window );

    GetFNum( "\pMonaco", &fontNum );

    if ( fontNum != 0 )
        TextFont( fontNum );

    MoveTo( kHorizontalPixel, kVerticalPixel );
    DrawString("\pHello, world!");
}

```

## Changing Text Style

The Macintosh supports seven font styles: **bold**, *italic*, underlined, outline, shadow, ~~condensed~~, and `extended`, or any combination of these. Chapter 5 shows you how to set text styles using menus. For now, try inserting the call `TextFace( style )` before the call to `DrawString()`. Here's one example:

```

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

```

```

window = GetNewWindow( kBaseResID , nil,
                      kMoveToFront );

if ( window == nil )
{
    SysBeep( 10 ); /* Couldn't load the WIND
                  resource!!! */

    ExitToShell();
}

ShowWindow( window );
SetPort( window );

TextFace( bold ); /* Try the other styles */

MoveTo( kHorizontalPixel, kVerticalPixel );
DrawString("\pHello, world!");
}

```



Some predefined styles taken from the #include file QuickDraw.h:

bold	shadow
italic	condense
underline	extend
outline	

You can also combine styles; try `TextFace( bold + italic )` or some other combination.

## Changing Text Size

It's also easy to change the size of the fonts, using the `TextSize()` Toolbox routine:

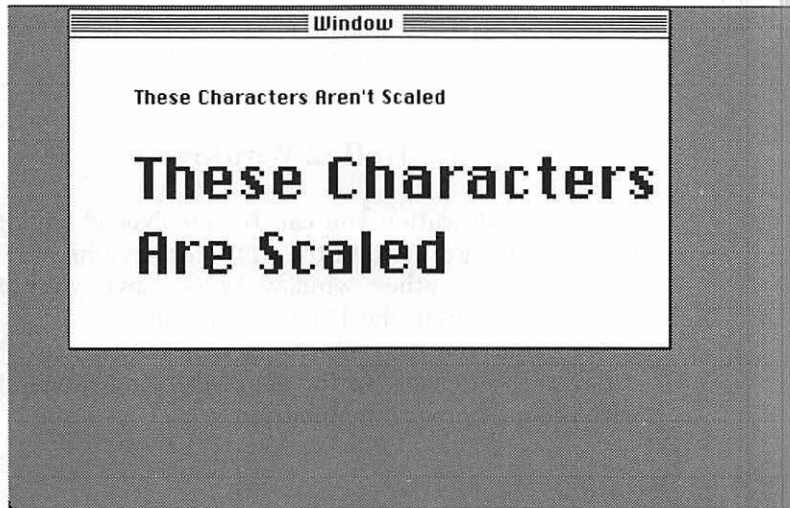
```

short    myFontSize;

TextSize( myFontSize );

```

The number you supply as an argument to `TextSize()` is the font size that will be used the next time text is drawn in the current window. The Font Manager will draw the smoothest text it can in the



**Figure 3.32** Font scaling with a non-TrueType font.

font size you specify. **TrueType** fonts (Apple's new font technology available under System 7) will yield the best results. If the current font is not a TrueType font, and the requested size is not available, the Font Manager will scale the font to the requested size; this may result in jagged characters (Figure 3.32).

Try this variation in your code:

```

/***** WindowInit *****/

void WindowInit( void )
{
    WindowPtr window;

    window = GetNewWindow( kBaseResID , nil,
                          kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 ); /* Couldn't load the WIND
                       resource!!! */
        ExitToShell();
    }

    ShowWindow( window );
    SetPort( window );
}

```



```
    TextSize( 24 );    /* Try some other sizes... */

    MoveTo( kHorizontalPixel, kVerticalPixel );
    DrawString("\pHello, world!");
}
```

## Changing the Hello2 Window

Another modification you can try involves changing Hello2's window type. Use ResEdit to edit the WIND resource in Hello2.π.rsrc. Click on one of the other window types, save your changes, and run Hello2.π again to check out your results.

Now that you have mastered QuickDraw's text-handling routines, you're ready to exercise the shape-drawing capabilities of QuickDraw with the next program: Mondrian.

---

## Mondrian

---

The Mondrian program opens a window and draws randomly generated ovals, alternately filled with white or black. Like Hello2, Mondrian waits for a mouse click to exit. The program, with its variants, demonstrates most of QuickDraw's shape-drawing functionality.

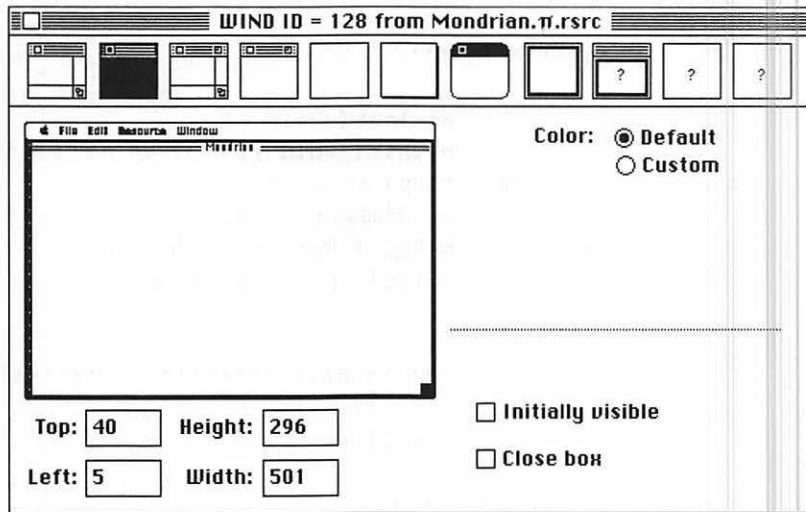
Mondrian is made up of three steps:

- As always, start by initializing the Toolbox;
- Next, initialize the drawing window;
- Finally, draw random QuickDraw ovals in a loop until the mouse button is clicked.

Create a new folder called Mondrian in the Development folder. Just as you did with Hello2, you'll use this folder to collect all the files associated with the Mondrian program. Start by creating the Mondrian resource file.

## Resources

The Mondrian program needs a WIND resource, just as Hello2 did. Use ResEdit to create a new resource file called Mondrian.π.rsrc inside the Mondrian folder. Next, use ResEdit to create a new WIND resource, matching the specifications in Figure 3.33. Select **Set 'WIND' Characteristics...** from the **WIND** menu and change the



**Figure 3.33** The WIND resource from Mondrian.π.rsrc.

**Window Title:** field to Mondrian. Next, select **Get Resource Info** from the **Resource** menu, set the **ID:** field to 128, and check the **Purgeable** checkbox. Quit ResEdit, saving your changes.

Next, go into THINK C and create a new project called Mondrian.π inside the Mondrian folder. Use **Add...** from the **Source** menu to add MacTraps to the project. You'll find MacTraps inside the Development folder, inside the THINK C 5.0 Folder, inside the Mac Libraries folder.

Once MacTraps is added, open a new source code window, as you did with Hello2, and enter the program:

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L
#define kRandomUpperLimit 32768

/*****
/*  Globals
*****/

long      gFillColor = blackColor;
```

```

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
void    WindowInit( void );
void    MainLoop( void );
void    DrawRandomRect( void );
void    RandomRect( Rect *rectPtr );
short   Randomize( short range );


/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();
    MainLoop();
}


/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}


/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

```

```
        window = GetNewWindow( kBaseResID , nil,
                                kMoveToFront );

        if ( window == nil )
        {
            SysBeep( 10 );    /* Couldn't load the WIND
                               resource!!! */

            ExitToShell();
        }

        ShowWindow( window );
        SetPort( window );
    }

/***** MainLoop *****/

void    MainLoop( void )
{
    GetDateTime( (unsigned long *) ( &randSeed ) );

    while ( ! Button() )
    {
        DrawRandomRect();

        if ( gFillColor == blackColor )
            gFillColor = whiteColor;
        else
            gFillColor = blackColor;
    }
}

/***** DrawRandomRect *****/

void    DrawRandomRect( void )
{
    Rect    randomRect;

    RandomRect( &randomRect );
    ForeColor( gFillColor );
    PaintOval( &randomRect );
}
```

```

/***** RandomRect *****/

void      RandomRect( Rect *rectPtr )
{
    WindowPtr    window;

    window = FrontWindow();

    rectPtr->left = Randomize( window->portRect.right
        - window->portRect.left );
    rectPtr->right = Randomize( window->portRect.right
        - window->portRect.left );
    rectPtr->top = Randomize( window->portRect.bottom
        - window->portRect.top );
    rectPtr->bottom = Randomize( window->portRect.bottom
        - window->portRect.top );
}

/***** Randomize *****/

short      Randomize( short range )
{
    long randomNumber;

    randomNumber = Random();

    if ( randomNumber < 0 )
        randomNumber *= -1;

    return( (randomNumber * range) / kRandomUpperLimit );
}

```

## Running Mondrian

Once you've finished typing in the code, save it as `Mondrian.c` and add it to the project using **Add (not Add...)** from the **Source** menu. Next, select **Run** from the **Project** menu, clicking **Yes** to the question **Bring the project up to date?** If the source code compiles correctly, you should see something like Figure 3.34. The Mondrian window will appear, filled with black and white randomly generated ovals. Click the mouse button to exit Mondrian. If you get a different result, check out your resource; make sure the WIND resource has the correct resource ID; make sure your resource file is named correctly. If your resource file appears to be all right, go through the code for typing errors.

Now let's look at the Mondrian code.

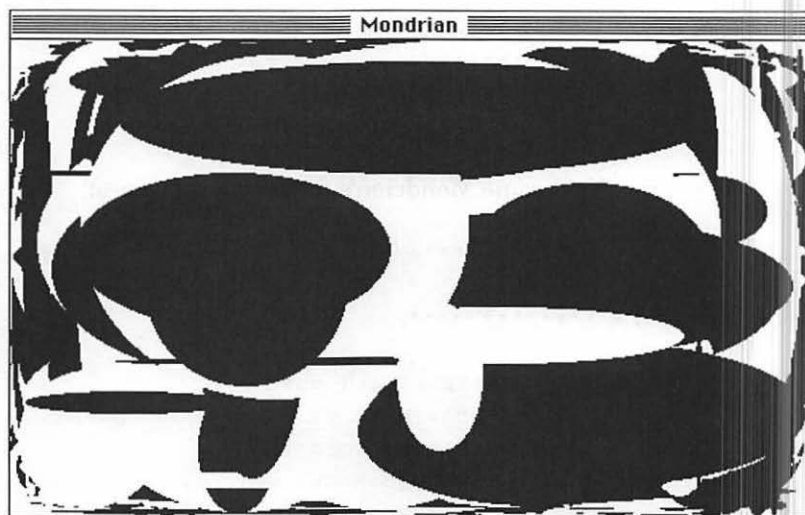


Figure 3.34 Running Mondrian.

---

## Walking Through the Mondrian Code

---

The *Mac Primer* uses the convention of starting resource ID numbers at 128, adding one each time a new resource ID is needed. Use any number you want (as long as it's between 128 and 32,767).



Remember, if you change the number of the starting resource ID, you'll need to change the resource ID of all the resources in your .rsrc files, too.

The `#define kBaseResID` and `kMoveToFront` are identical to those used in *Hello2*. The global variable `gFillColor` determines the color used to draw each oval. As each oval is drawn, `gFillColor` is alternated between `blackColor` and `whiteColor`.

```
#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L
#define kRandomUpperLimit   32768
```

```

/*****
/*  Globals  */
*****/

long      gFillColor = blackColor;

```

Next come Mondrian's function prototypes:

```

/*****
/*  Functions  */
*****/

void      ToolBoxInit( void );
void      WindowInit( void );
void      MainLoop( void );
void      DrawRandomRect( void );
void      RandomRect( Rect *rectPtr );
short     Randomize( short range );

```

The main routine is exactly the same as it was in Hello2:

```

/***** main *****/

void      main( void )
{
    ToolBoxInit();
    WindowInit();
    MainLoop();
}

```

The Toolbox initialization routine is also the same as in Hello2.

```

/***** ToolBoxInit *****/

void      ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

WindowInit() loads WIND number 128 from the resource file, storing a pointer to it in myWindow. If the window couldn't be created for some reason, GetNewWindow() will return a value of nil. In that case, Mondrian issues a beep, then exits to the Finder.

```
/****** WindowInit *****/  
  
void WindowInit( void )  
{  
    WindowPtr window;  
  
    window = GetNewWindow( kBaseResID , nil,  
                           kMoveToFront );  
  
    if ( window == nil )  
    {  
        SysBeep( 10 ); /* Couldn't load the WIND  
                        resource!!! */  
        ExitToShell();  
    }  
}
```

If GetNewWindow() was able to create the window, window is made visible and is made the current port.

```
ShowWindow( window );  
SetPort( window );  
}
```

MainLoop() starts by using the current time (in seconds since January 1, 1904) to seed the Mac random number generator. The QuickDraw global randSeed is used as a seed by the random number generator. If you didn't modify randSeed, you'd generate the same patterns every time you ran Mondrian.



The GetDateTime ( ) Toolbox routine requires a pointer to an unsigned long integer. That's why (Unsigned long \*) is put in front of randSeed in MainLoop ( ). Although providing this information is not required for Toolbox calls by THINK C, it will save you plenty of debugging time later on.



```

/***** MainLoop *****/

void    MainLoop( void )
{
    GetDateTime( (unsigned long *) ( &randSeed ) );

```

`MainLoop()` then sets up a loop that falls through when the mouse button is pressed. In the loop, `DrawRandomRect()` is called, first generating a random rectangle inside the window, then drawing an oval in the rectangle. Next, `gFillColor` is flipped from black to white or from white to black.

```

    while ( ! Button() )
    {
        DrawRandomRect();

        if ( gFillColor == blackColor )
            gFillColor = whiteColor;
        else
            gFillColor = blackColor;
    }
}

```

`DrawRandomRect()` controls the actual drawing of the ovals in the window. `RandomRect()` generates a random rectangle bounded by the Mondrian window, `ForeColor()` sets the current drawing color to `gFillColor`, and `PaintOval()` paints the oval inside the generated rectangle.

```

/***** DrawRandomRect *****/

void    DrawRandomRect( void )
{
    Rect    randomRect;

    RandomRect( &randomRect );
    ForeColor( gFillColor );
    PaintOval( &randomRect );
}

```

`RandomRect()` uses the `FrontWindow()` Toolbox routine to retrieve a pointer to the frontmost window. Since Mondrian only uses one window, `FrontWindow()` is guaranteed to return a pointer to the window we want.

```

/***** RandomRect *****/

void RandomRect( Rect *rectPtr )
{
    WindowPtr window;

    window = FrontWindow();

```

Next, `RandomRect()` sets up the rectangle to be used in drawing the oval. Each of the four sides of the rectangle is generated as a random number between the right and left (or top and bottom, as appropriate) sides of the window pointed to by `window`.



The notation `structPtr->aField` refers to the field `aField` in the struct pointed to by `structPtr`. For example:

```
rectPtr->left
```

refers to the field named `left` in the struct pointed to by `rectPtr`. Struct pointer notation is specific to C, not peculiar to the Macintosh.

Every window data structure has a field named `portRect` (of type `Rect`) that defines the boundary of the content region of the window. Because `window` is a pointer to a window data structure, you use `window->portRect` to access this rectangle.

```

rectPtr->left = Randomize( window->portRect.right
    - window->portRect.left );
rectPtr->right = Randomize( window->portRect.right
    - window->portRect.left );
rectPtr->top = Randomize( window->portRect.bottom
    - window->portRect.top );
rectPtr->bottom = Randomize( window->portRect.bottom
    - window->portRect.top );
}

```

`Randomize()` takes an integer argument and returns a positive integer greater than or equal to 0, and less than the argument. You may find `Randomize()` helpful in your own applications.

```

/***** Randomize *****/

```

```

short  Randomize( short range )
{
    long    randomNumber;

```

Randomize() starts by calling Random(), a Toolbox utility that returns a random number between -32,767 and 32,767.

```

    randomNumber = Random();

```

If the value returned is negative, multiply it by -1. This creates a number between 0 and 32,767.

```

    if ( randomNumber < 0 )
        randomNumber *= -1;

```

Finally, multiply that number by the input parameter, then divide by kRandomUpperLimit (which was defined earlier to be 32,768). This creates a number greater than or equal to 0 and less than the input parameter.

```

    return( (randomNumber * range) / kRandomUpperLimit );
}

```

## **Variants**

Here are some variants of Mondrian. The first few change the shape of the repeated figure in the window from ovals to some other shapes.

Your first new shape will be a rectangle. This one's easy: Just change the PaintOval() call to PaintRect(). When you run this, you should see rectangles instead of ovals.

Your next new shape is the rounded rectangle. You'll need two new parameters for PaintRoundRect(): ovalWidth and ovalHeight. These two parameters affect the curvature of the corners of the rectangle (I:179). Try the following values for ovalWidth and ovalHeight:

```

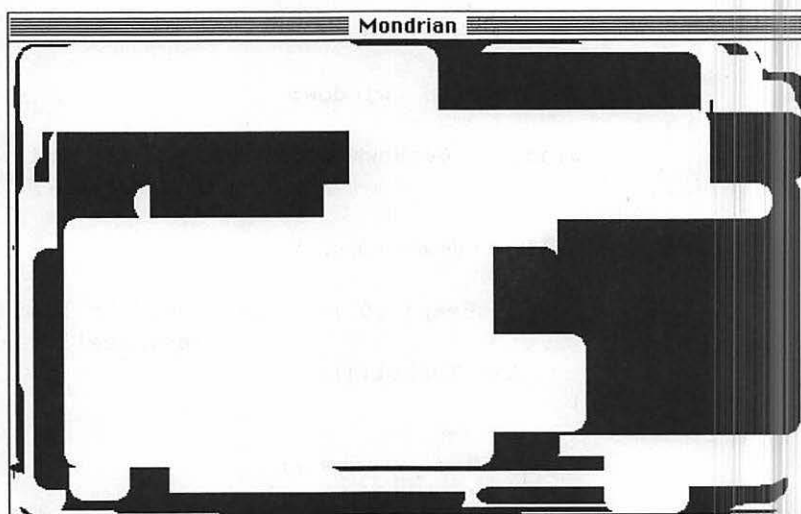
#define kOvalWidth    20
#define kOvalHeight   20

```

Now, change `DrawRandomRect()` as follows:

```
/* ***** DrawRandomRect ***** */  
  
void DrawRandomRect( void )  
{  
    Rect randomRect;  
    RandomRect( &randomRect );  
    ForeColor( gFillColor );  
    PaintRoundRect( &randomRect, kOvalWidth, kOvalHeight );  
}
```

When you run this variation, you should see something like Figure 3.35.



**Figure 3.35** Mondrian with rounded rectangles.

Instead of filling the rectangles, try using `FrameRoundRect()` to draw just the outline of your rectangles:

```
/* ***** DrawRandomRect ***** */  
  
void DrawRandomRect( void )  
{  
    Rect randomRect;
```

```

    RandomRect( &randomRect );
    ForeColor( gFillColor );
    FrameRoundRect( &randomRect, kOvalWidth, kOvalHeight );
}

```

The framing function is more interesting if you change the state of your pen: The default setting for your pen is a size of 1 pixel wide by 1 pixel tall, and the pattern used to fill drawn lines is black. Start by adding #defines of `PEN_WIDTH` and `PEN_HEIGHT`:

```

#define kPenWidth      10
#define kPenHeight     2

```

Change the pen state by modifying `WindowInit()` as follows:

```

/***** WindowInit *****/

void      WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID , nil,
                          kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND
                           resource!!! */

        ExitToShell();
    }

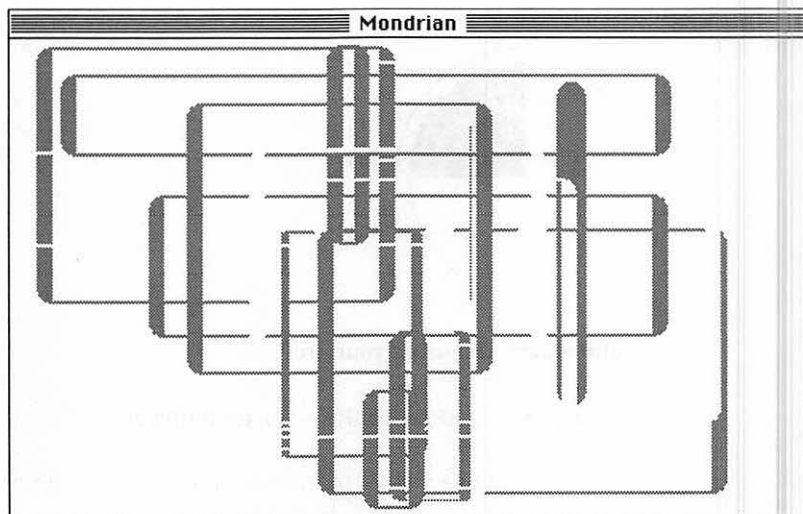
    ShowWindow( window );
    SetPort( window );

    PenSize( kPenWidth, kPenHeight );
    PenPat( gray );
}

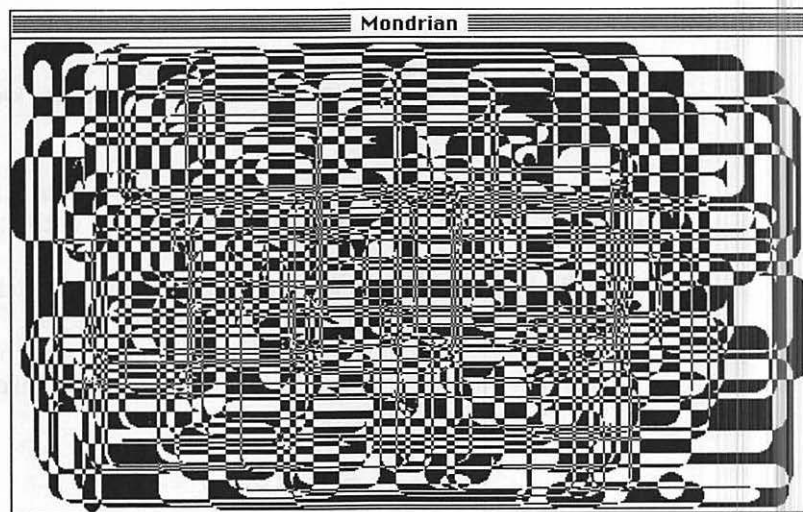
```

Here, you changed the pen pattern to gray, the pen width to `kPenWidth`, and the pen height to `kPenHeight`. Your result should look something like Figure 3.36.

While you're at it, try using `InvertRoundRect()` instead of `FrameRoundRect()`. `InvertRoundRect()` will invert the pixels in its rectangle. The arguments are handled in the same way (Figure 3.37).

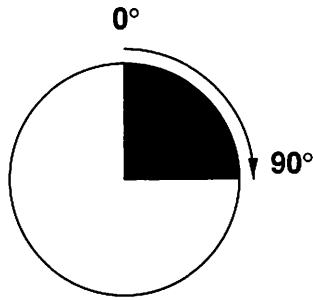


**Figure 3.36** Mondrian with framed, rounded rectangles.



**Figure 3.37** A '60s Mondrian with inverted, rounded rectangles.

Next, try using `FrameArc()` in place of `InvertRoundRect()`. `FrameArc()` requires two new parameters. The first defines the arc's starting angle, and the second defines the size of the arc. Both are expressed in degrees (Figure 3.38).



**Figure 3.38** Figuring your arc.

Change `DrawRandomRect()` as follows:

```

/***** DrawRandomRect *****/

void    DrawRandomRect( void )
{
    Rect    randomRect;

    RandomRect( &randomRect );
    ForeColor( gFillColor );
    FrameArc( &randomRect, kStartDegrees, kArcDegrees );
}

```

Don't forget to `#define` `kStartDegrees` and `kArcDegrees`. Try using values of 0 and 270. Experiment with `PaintArc()` and `InvertArc()`.

We'll do one final variation with `QuickDraw`, which is useful only on color monitors. If you change the `ForeColor()` arguments in `MainLoop()`, you can see colored filled ovals (or whatever your program is currently producing). Change the declaration of the global `gFillColor` as follows:

```

long    gFillColor = redColor;

```

Modify `MainLoop()` as follows:

```

/***** MainLoop *****/

void    MainLoop( void )
{
    GetDateTime( &randSeed );

```

```

while ( ! Button() )
{
    DrawRandomRect();

    if ( gFillColor == redColor )
        gFillColor = yellowColor;
    else
        gFillColor = redColor;
}
}

```

Finally, make sure `DrawRandomRect()` calls a `Paint()` function to do its drawing (as opposed to a `Frame()` or `Invert()` function):

```

/***** DrawRandomRect *****/

void    DrawRandomRect( void )
{
    Rect    randomRect;

    RandomRect( &randomRect );
    ForeColor( gFillColor );
    PaintArc( &randomRect, kStartDegrees, kArcDegrees );
}

```

The following colors have already been defined for you: `blackColor`, `whiteColor`, `redColor`, `yellowColor`, `greenColor`, `blueColor`, `cyanColor`, and `magentaColor`. These colors are part of Classic QuickDraw—the original, eight-color QuickDraw model that was part of the original Macintosh. Newer Macs support a new version of QuickDraw called Color QuickDraw, which supports millions of different colors. (Color QuickDraw is discussed in Volume II of the *Mac Primer*.) The programs you write using the eight colors of Classic QuickDraw will run on any Macintosh (even the Mac II series).

The next program demonstrates how to load QuickDraw picture resources and draw them in a window.

---

## ShowPICT

---

ShowPICT will take your favorite artwork (in the form of a PICT resource) and display it in a window. You can create a PICT resource by copying any graphic to the Mac clipboard, then pasting it into a ResEdit PICT window. We'll show you how a little later. We copied our




artwork from the scrapbook that comes with the Macintosh System disks.

ShowPICT is made up of five distinct steps:

- Initialize the Toolbox;
- Load a resource window, show it, and make it the current port;
- Load a picture from the resource file;
- Center the picture, then draw it in the window;
- Wait for the mouse button to be clicked.

## Resources

Start by creating a new folder, called ShowPICT, in the Development folder. Next, using ResEdit, create a new resource file called ShowPICT. $\pi$ .rsrc in the ShowPICT folder. Create a WIND resource using the specifications shown in Figure 3.39. Select **Set 'WIND' Characteristics...** from the **WIND** menu and set the **Window title:** field to ShowPICT, then click **OK**. Next, select **Get Resource Info** from the **Resource** menu, set the resource ID of the WIND to 128, and check the **Purgeable** checkbox. Close the Resource Info window, close the WIND editing window, and close the WIND list window, leaving the window titled ShowPICT. $\pi$ .rsrc as the only open window.

Now you'll create a PICT resource. Pull down the  menu and select the **Scrapbook**. Find a picture that is of type PICT—you can tell by checking the label on the bottom right of the Scrapbook

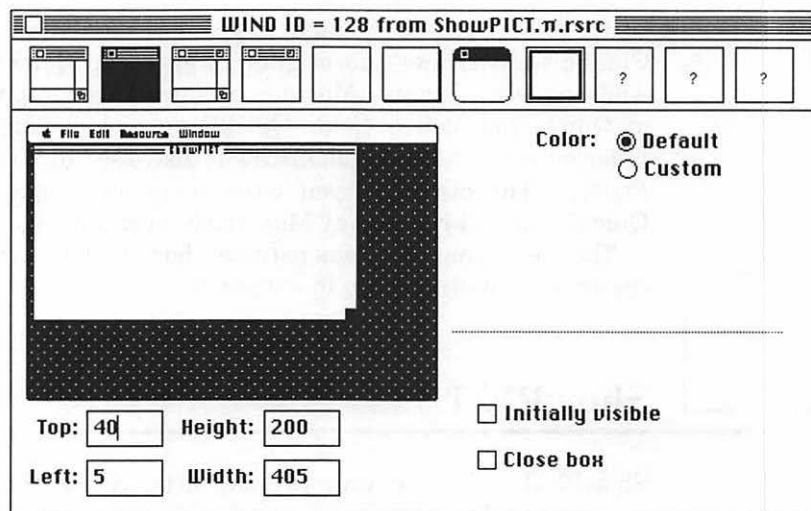
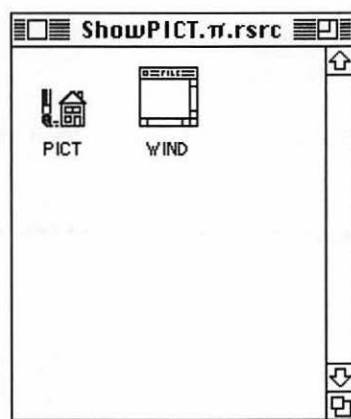


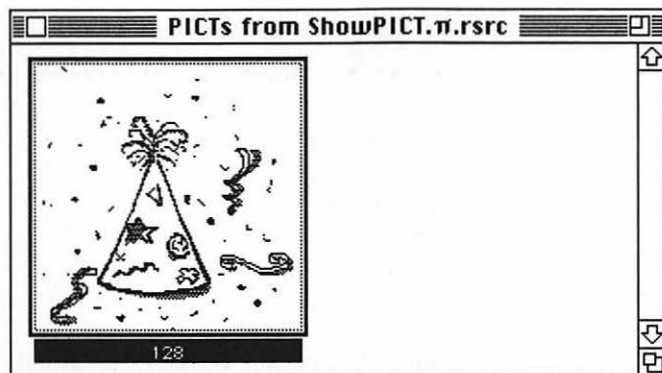
Figure 3.39 ShowPICT's WIND resource.

window—pull down the **Edit** menu, and select **Copy**. Close the **Scrapbook** and return to ResEdit. Select **Paste** from the **Edit** menu; ResEdit will use the copied picture to create a PICT resource (Figure 3.40).

In the ShowPICT.π.rsrc window, double-click on the PICT icon. A window displaying all of the PICTs in ShowPICT.π.rsrc will appear (Figure 3.41). Click on your picture (it should be the only one) and select **Get Resource Info** from the **Resource** menu. Set the resource ID of the PICT to 128 and check the **Purgeable** checkbox. Finally, quit ResEdit, saving your changes to ShowPICT.π.rsrc.



**Figure 3.40** ResEdit window showing ShowPICT's two resource types, PICT and WIND.



**Figure 3.41** The PICT resource created for ShowPICT.

Next, go into THINK C and create a new project called ShowPICT. $\pi$  inside the ShowPICT folder. Select **New** from the **File** menu and enter the following code:

```
#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
void    WindowInit( void );
void    DrawMyPicture( void );
void    CenterPict( PicHandle picture, Rect *destRectPtr );

/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();

    DrawMyPicture();

    while ( !Button() ) ;
}

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}
```

```

/***** WindowInit *****/

void      WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND
                           resource!!! */
        ExitToShell();
    }

    ShowWindow( window );
    SetPort( window );
}

/***** DrawMyPicture *****/

void      DrawMyPicture( void )
{
    Rect        pictureRect;
    WindowPtr    window;
    PicHandle    picture;

    window = FrontWindow();

    pictureRect = window->portRect;

    picture = GetPicture( kBaseResID );

    if ( picture == nil )
    {
        SysBeep( 10 );    /* Couldn't load the PICT
                           resource!!! */
        ExitToShell();
    }

    CenterPict( picture, &pictureRect );
    DrawPicture( picture, &pictureRect );
}

```

```
/****** CenterPict *****/  
  
void      CenterPict( PicHandle picture, Rect *destRectPtr )  
{  
    Rect      windRect, pictRect;  
  
    windRect = *destRectPtr;  
    pictRect = (*( picture )).picFrame;  
    OffsetRect( &pictRect, windRect.left - pictRect.left,  
                windRect.top - pictRect.top);  
    OffsetRect( &pictRect, (windRect.right -  
                pictRect.right)/2, (windRect.bottom -  
                pictRect.bottom)/2);  
    *destRectPtr = pictRect;  
}
```

## Running ShowPICT

After you've finished typing in the code, save the file as `ShowPICT.c` and add it to your project. Next, select **Add...** from the **Source** menu and add `MacTraps` to your project. Finally, select **Run** from the **Project** menu. If everything went well, a window similar to the one in Figure 3.42 should appear. Did your PICT appear in your `ShowPICT` window? If not, check your PICT's resource ID. Check the name of your project and resource file. Check your `WIND` resource. If everything else seems to be OK, check your code for typing errors.

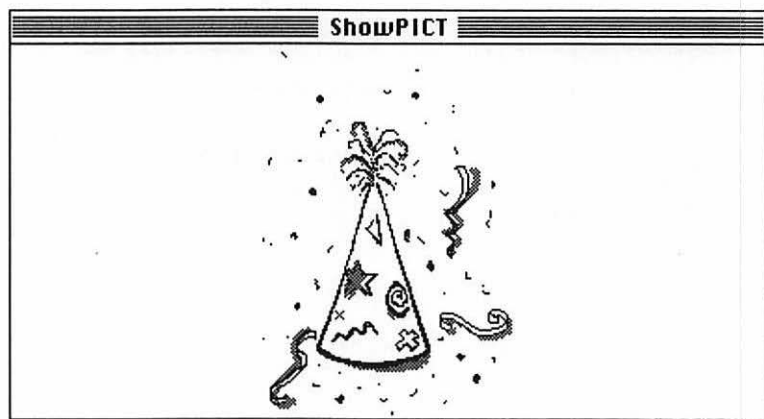


Figure 3.42 Running ShowPICT.

## Walking Through the ShowPICT Code

The #defines kBaseResID and kMoveToFront perform the same function as they did in earlier programs.

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

/*****
/*  Functions  */
*****/

void    ToolBoxInit( void );
void    WindowInit( void );
void    DrawMyPicture( void );
void    CenterPict( PicHandle picture, Rect *destRectPtr );
```

main() starts by initializing the Toolbox. Next, WindowInit() is called to set up ShowPICT's window.

```
/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();
```

DrawMyPicture() loads the PICT from the resource file, then draws the PICT in the ShowPICT window. Finally, main() waits for a click of the mouse button.

```
    DrawMyPicture();

    while ( !Button() );
}
```

The Toolbox initialization routine remains the same:

```
/***** ToolboxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
```

```

    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

WindowInit() remains the same as in Mondrian.

```

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND
                           resource!!! */
        ExitToShell();
    }

    ShowWindow( window );
    SetPort( window );
}

```

DrawMyPicture() calls FrontWindow() to retrieve a pointer to the frontmost window. Since ShowPICT uses only one window, FrontWindow() is guaranteed to return a pointer to the correct window.

```

/***** DrawMyPicture *****/

void    DrawMyPicture( void )
{
    Rect        pictureRect;
    WindowPtr    window;
    PicHandle    picture;

    window = FrontWindow();
}

```

Next, `pictureRect` is set to the size of the `ShowPICT` window's content region:

```
pictureRect = window->portRect;
```

Next, `GetPicture()` is called to retrieve the `PICT` resource from the resource file. If the `PICT` can't be loaded, the program beeps, then exits.

```
picture = GetPicture( kBaseResID );
if ( picture == nil )
{
    SysBeep( 10 );    /* Couldn't load the PICT
                       resource!!! */
    ExitToShell();
}
```



It's important to remember that `GetPicture()` may return `nil` for several different reasons. The most likely: `GetPicture()` either couldn't find or couldn't load the `PICT` resource. `GetPicture()` will also return `nil` if it can't allocate enough memory to create the picture's data structure.

Next, the picture and the `Rect` are passed to `CenterPict()`. `CenterPict()` constructs a new `Rect` the size of picture, centering it in the original `Rect`. Finally, `DrawMyPicture()` uses `DrawPicture()` to draw picture in the newly centered `Rect`.

```
CenterPict( picture, &pictureRect );
DrawPicture( picture, &pictureRect );
}
```

In this program, `CenterPict()` is used to center a picture in a window. The original `Rect` is copied into the local variable `windRect`. The picture's framing `Rect` is then copied to the local variable `pictRect`. Finally, each field in the original `Rect` is modified, based on the corresponding fields in `windRect` and `pictRect`. For example, `pictRect.top` is adjusted to become the new top of the picture.

`CenterPict()` is a useful utility routine. You'll be seeing it again in other chapters.



```

/***** CenterPict *****/

void      CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top);
    OffsetRect( &pictRect, (windRect.right -
                pictRect.right)/2, (windRect.bottom -
                pictRect.bottom)/2);
    *destRectPtr = pictRect;
}

```



A **Handle** is a specialized pointer to a pointer. Handles are a necessary part of the Mac's memory management scheme. They allow the Macintosh Memory Manager to relocate blocks of memory as it needs to, without disturbing your program.

If your program makes use of a pointer to a variable or a block of memory, you depend on that variable or block of memory to stay in one place. If the block moves, your pointer will no longer point to it!

Since pointer blocks are not allowed to move, they tend to fragment the memory available to your program. That's where **Handles** come in. A **Handle** is a pointer to a pointer. If you have a **Handle** to a block of memory, the Macintosh Memory Manager can move the block around in memory without affecting your program's **Handle** to the block. This allows the Memory Manager to maximize the efficiency of the memory available to your program.

We'll show you some of the basics of using handles, but we won't spend a lot of time on them (there's an entire chapter dedicated to handles and related topics in Volume II of the *Mac Primer*). You should read up on handles and the Mac memory management scheme. Eventually you'll want to write code that takes advantage of handles.



In ShowPICT, we declare a handle to a picture (pointer to a pointer to a picture). We then set the handle to the value returned by GetPicture():

```
PicHandle      picture;  
picture = GetPicture( kBaseResID );
```

Like most of the Toolbox functions that return handles, GetPicture() actually allocates the memory for the picture itself, as well as the memory for the pointer to the picture. The great thing about handles is that you hardly know they're there.

---

## Variants

---

Try using different pictures, either from the Scrapbook or from MacPaint or some other Macintosh graphics program. With a little experimentation, you should be able to copy and paste these files into your resource file. In Chapter 4, you'll see an enhanced ShowPICT program.

---

## Screen Saver: The FlyingLine Program

---

The FlyingLine is the last program in the QuickDraw chapter. Although it does demonstrate the use of line drawing in QuickDraw, we included it mostly because it's fun. The FlyingLine draws a set of lines that move across the screen with varying speeds, directions, and orientations. The program can be used as a screen saver (we even show you how to hide the menu bar).

The FlyingLine program consists of four steps:

- Initialize the Toolbox;
- Set up the FlyingLine window;
- Initialize the FlyingLine data structure, drawing it once;
- Redraw the FlyingLine inside a loop until a mouse click occurs.

Create a folder called FlyingLine inside your Development folder. FlyingLine needs no resources, so you won't need to create a resource file for this project. Go into THINK C and create a new project called FlyingLine.π inside the FlyingLine folder. Select

**New** from the **File** menu to open a new window for the FlyingLine source code:

```
#define kNumLines          50  /* Try 100 or 150 */
#define kMoveToFront      (WindowPtr)-1L
#define kRandomUpperLimit 32768
#define kEmptyString      "\p"
#define kEmptyTitle       kEmptyString
#define kVisible           true
#define kNoGoAway          false
#define kNilRefCon         (long)nil

/*****/
/*  Globals  */
/*****/

Rect      gLines[ kNumLines ];
short     gDeltaTop=3, gDeltaBottom=3; /* These four
                                         are the key to FlyingLine!*/
short     gDeltaLeft=2, gDeltaRight=6;
short     gOldMBarHeight;

/*****/
/*  Functions  */
/*****/

void      ToolBoxInit( void );
void      WindowInit( void );
void      LinesInit( void );
void      MainLoop( void );
void      RandomRect( Rect *rectPtr );
short     Randomize( short range );
void      RecalcLine( short i );
void      DrawLine( short i );

/***** main *****/

void      main( void )
{
    ToolBoxInit();
    WindowInit();
    LinesInit();
    MainLoop();
}
```

```

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** WindowInit *****/

void    WindowInit( void )
{
    Rect        totalRect, mBarRect;
    RgnHandle    mBarRgn;
    WindowPtr    window;

    gOldMBarHeight = MBarHeight;
    MBarHeight = 0;

    window = NewWindow( nil, &(screenBits.bounds),
        kEmptyTitle, kVisible, plainDBox, kMoveToFront,
        kNoGoAway, kNilRefCon );

    SetRect( &mBarRect, screenBits.bounds.left,
        screenBits.bounds.top,
        screenBits.bounds.right,
        screenBits.bounds.top+gOldMBarHeight );

    mBarRgn = NewRgn();
    RectRgn( mBarRgn, &mBarRect );
    UnionRgn( window->visRgn, mBarRgn, window->visRgn );
    DisposeRgn( mBarRgn );
    SetPort( window );
    FillRect( &(window->portRect), black );
                                /* Change black to ltGray, */
    PenMode( patXor );          /*<-- and comment out this line*/
}

```

```

/***** LinesInit *****/

void    LinesInit( void )
{
    short i;

    HideCursor();
    GetDateTime( (unsigned long *) ( &randSeed ) );
    RandomRect( &(gLines[ 0 ]) );
    DrawLine( 0 );

    for ( i=1; i<kNumLines; i++ )
    {
        gLines[ i ] = gLines[ i-1 ];
        RecalcLine( i );
        DrawLine( i );
    }
}

/***** MainLoop *****/

void    MainLoop( void )
{
    short i;

    while ( ! Button() )
    {
        DrawLine( kNumLines - 1 );
        for ( i=kNumLines-1; i>0; i- )
            gLines[ i ] = gLines[ i-1 ];
        RecalcLine( 0 );
        DrawLine( 0 );
    }
    MBarHeight = gOldMBarHeight;
}

/***** RandomRect *****/

void    RandomRect( Rect *rectPtr )
{
    WindowPtr    window;

    window = FrontWindow();

```

```

rectPtr->left = Randomize( window->portRect.right
    - window->portRect.left );
rectPtr->right = Randomize( window->portRect.right
    - window->portRect.left );
rectPtr->top = Randomize( window->portRect.bottom
    - window->portRect.top );
rectPtr->bottom = Randomize( window->portRect.bottom
    - window->portRect.top );
}

/***** Randomize *****/

short    Randomize( short range )
{
    long    randomNumber;

    randomNumber = Random();

    if ( randomNumber < 0 )
        randomNumber *= -1;

    return( (randomNumber * range) / kRandomUpperLimit );
}

/***** RecalcLine *****/

void    RecalcLine( short i )
{
    WindowPtr    window;

    window = FrontWindow();

    gLines[ i ].top += gDeltaTop;
    if ( ( gLines[ i ].top < window->portRect.top ) ||
        ( gLines[ i ].top > window->portRect.bottom ) )
    {
        gDeltaTop *= -1;
        gLines[ i ].top += 2*gDeltaTop;
    }

    gLines[ i ].bottom += gDeltaBottom;
    if ( ( gLines[ i ].bottom < window->portRect.top ) ||
        ( gLines[ i ].bottom > window->portRect.bottom ) )

```

```

    {
        gDeltaBottom *= -1;
        gLines[ i ].bottom += 2*gDeltaBottom;
    }
    gLines[ i ].left += gDeltaLeft;
    if ( ( gLines[ i ].left < window->portRect.left ) ||
        ( gLines[ i ].left > window->portRect.right ) )
    {
        gDeltaLeft *= -1;
        gLines[ i ].left += 2*gDeltaLeft;
    }

    gLines[ i ].right += gDeltaRight;
    if ( ( gLines[ i ].right < window->portRect.left ) ||
        ( gLines[ i ].right > window->portRect.right ) )
    {
        gDeltaRight *= -1;
        gLines[ i ].right += 2*gDeltaRight;
    }
}

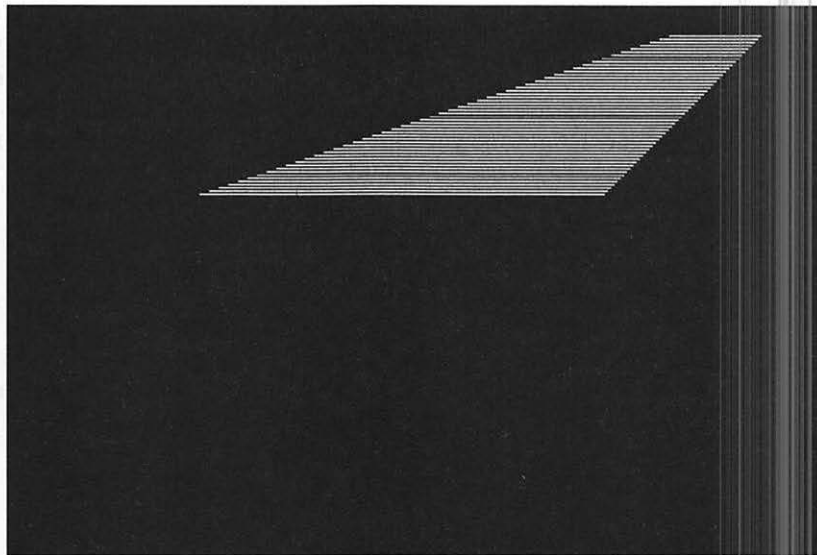
/***** DrawLine *****/

void DrawLine( short i )
{
    MoveTo( gLines[ i ].left, gLines[ i ].top );
    LineTo( gLines[ i ].right, gLines[ i ].bottom );
}

```

## Running FlyingLine

Save your source code in the FlyingLine folder as FlyingLine.c. Select **Add** from the **Source** menu to add the file to the project. Next, use **Add...** to add MacTraps to the project. Finally, select **Run** from the **Project** menu. If everything went well, you should see something like Figure 3.43. The window will be completely black except for the flying line; the menu bar should be hidden. As usual, click the mouse to exit. Now, let's take a look at the code.



**Figure 3.43** Running FlyingLine.

## Walking Through the FlyingLine Code

Most of FlyingLine should be familiar to you. The biggest change is in `WindowInit()`, where you create a window from scratch and hide the menu bar. We won't go into exhaustive detail on the FlyingLine algorithm, because it has little to do with the Toolbox. This one's just for fun!

`kNumLines` defines the number of lines in the FlyingLine. `kMoveToFront` plays the same role as in previous programs.

```
#define kNumLines          50  /* Try 100 or 150 */
#define kMoveToFront      (WindowPtr)-1L
#define kRandomUpperLimit 32768
```

The rest of the `#defines` will be used as parameters in the call to `NewWindow()` later in the program.

```
#define kEmptyString      "\p"
#define kEmptyTitle       kEmptyString
#define kVisible          true
#define kNoGoAway         false
#define kNilRefCon        (long)nil
```



The array `gLines` holds all of the individual lines in the `FlyingLine`. `gDeltaTop`, `gDeltaBottom`, `gDeltaLeft`, and `gDeltaRight` help determine the movement of the `FlyingLine` as it zooms around the screen. Finally, `gOldMBarHeight` saves the menu bar height, so you can restore it when the program exits.

```

/*****/
/*  Globals  */
/*****/

Rect      gLines[ kNumLines ];
short     gDeltaTop=3, gDeltaBottom=3; /*These four
                                     are the key to flying line!*/
short     gDeltaLeft=2, gDeltaRight=6;
short     gOldMBarHeight;

/*****/
/*  Functions  */
/*****/

void      ToolBoxInit( void );
void      WindowInit( void );
void      LinesInit( void );
void      MainLoop( void );
void      RandomRect( Rect *rectPtr );
short     Randomize( short range );
void      RecalcLine( short i );
void      DrawLine( short i );

```

The only change made to `main()` is the addition of the call to `LinesInit()` before the call to `MainLoop()`.

```

/***** main *****/

void      main( void )
{
    ToolBoxInit();
    WindowInit();
    LinesInit();
    MainLoop();
}

```

The Toolbox initialization for `FlyingLine` is the same as for the previous programs:

```

/***** ToolboxInit *****/
void    ToolboxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

The window initialization code for FlyingLine is unusual because the window itself is unusual. Normally, Mac programs display a menu bar. FlyingLine, however, will not. FlyingLine hides the menu bar (by making it 0 pixels tall) and creates a window that covers the entire screen.

```

/***** WindowInit *****/
void    WindowInit( void )
{
    Rect        totalRect, mBarRect;
    RgnHandle    mBarRgn;
    WindowPtr    window;

```

The Macintosh System's internal global variable `MBarHeight` contains the height of the menu bar in pixels. `WindowInit()` saves the current value of `MBarHeight` in the global `gOldMBarHeight`, then sets the menu bar height to 0.

```

gOldMBarHeight = MBarHeight;
MBarHeight = 0;

```



Although it's important to understand the technique involved here, it is even more important to remember that it's generally bad practice to mess with system globals. They are likely to change when new system versions come out. We use system globals in FlyingLine because Apple doesn't make it easy to hide the menu bar (mainly because they don't want programmers to do it). Because a screen saver has to hide the menu bar, FlyingLine uses a system global. Make sure you have good reasons to use system globals.

The call to `NewWindow()` is an alternative to `GetNewWindow()`. `GetNewWindow()` creates a window using the information specified in a WIND resource. `NewWindow()` also creates a window, but gets the window specifications from its parameter list:

```
FUNCTION NewWindow( wStorage : Ptr; boundsRect : Rect;
    title : Str255; visible : BOOLEAN; procID : INTEGER;
    behind : WindowPtr; goAwayFlag : BOOLEAN;
    refCon : LONGINT ) : WindowPtr;
```

The program specifies the size of the window as a `Rect`, using the `QuickDraw` global `screenBits.bounds` to create a window the size of the main screen.

```
window = NewWindow( nil, &(screenBits.bounds),
    kEmptyTitle, kVisible, plainDBox, kMoveToFront,
    kNoGoAway, kNilRefCon );
```

The next bit of code is tricky. It calls `SetRect()` to create a rectangle surrounding the normal menu bar. Next, it uses this `Rect` to create a new region. Then, it adds this region to the visible region of your window. As a result of this hocus-pocus, your window can overlap the menu bar, taking up the entire screen. If this makes you uncomfortable, don't panic. The call to `NewWindow()` is normally all you'll need in your applications. This extra code is just here to allow your window to obscure the menu bar.

```
SetRect( &mBarRect, screenBits.bounds.left,
    screenBits.bounds.top,
    screenBits.bounds.right,
    screenBits.bounds.top+gOldMBarHeight );

mBarRgn = NewRgn();
RectRgn( mBarRgn, &mBarRect );
UnionRgn( window->visRgn, mBarRgn, window->visRgn );
DisposeRgn( mBarRgn );
```

Next, the program calls `SetPort()` so that all its drawing will occur in window. Then, it uses `FillRect()` to fill the window with the black pattern. It uses `PenMode()` to set the pen's drawing mode to `patXor`. Experiment with some of the other pen modes. In addition, try changing the second `FillRect()` parameter to `ltGray`, and commenting out the call to `PenMode()`.

```

SetPort( window );
FillRect( &(amp;window->portRect), black );
/* Change black to ltGray, */
PenMode( patXor ); /*<-- and comment out this
line */
}

```



Don't be fooled by imitations. The second parameter to `FillRect()` is a pattern, not a color. These are the fill patterns you normally associate with the paint bucket in MacPaint, not the eight colors of Classic QuickDraw. You can experiment with colors by using a call to `PaintRect()`.

`LinesInit()` starts off by hiding the cursor. Next, it seeds the random number generator with the current date (*à la* Mondrian). Finally, it generates the first line of the flying line, draws it, then generates the rest of the lines and draws them.

```

/***** LinesInit *****/

void LinesInit( void )
{
    short i;

    HideCursor();
    GetDateTime( (unsigned long *) ( &randSeed ) );
    RandomRect( &(gLines[ 0 ]) );
    DrawLine( 0 );

    for ( i=1; i<kNumLines; i++ )
    {
        gLines[ i ] = gLines[ i-1 ];
        RecalcLine( i );
        DrawLine( i );
    }
}

```

`MainLoop()` sets up a loop that falls through when the mouse button is clicked. At the end of the loop, the menu bar height is restored. If you don't do this, you won't be able to pick from the menu bar when you exit the program. Oops! (If by accident, you don't reset the menu bar height, it won't come back when you return to the Finder. Restart your Mac to reset the menu bar height.)

Inside the loop, the program erases and redraws each line in the `FlyingLine`. It erases lines by redrawing them in exactly the same position. Because the pen mode is set to `patXor`, this has the effect of erasing the line. Thus, the first call to `DrawLine()` in `MainLoop()` erases the last line in the `gLines` array. This simulates the line moving across the screen.

```

/***** MainLoop *****/

void    MainLoop( void )
{
    short i;

    while ( ! Button() )
    {
        DrawLine( kNumLines - 1 );
        for ( i=kNumLines-1; i>0; i- )
            gLines[ i ] = gLines[ i-1 ];
        RecalcLine( 0 );
        DrawLine( 0 );
    }
    MBarHeight = gOldMBarHeight;
}

```

**You've seen this routine in Mondrian:**

```

/***** RandomRect *****/

void    RandomRect( Rect *rectPtr )
{
    WindowPtr    window;

    window = FrontWindow();

    rectPtr->left = Randomize( window->portRect.right
        - window->portRect.left );
    rectPtr->right = Randomize( window->portRect.right
        - window->portRect.left );
    rectPtr->top = Randomize( window->portRect.bottom
        - window->portRect.top );
    rectPtr->bottom = Randomize( window->portRect.bottom
        - window->portRect.top );
}

```

Another routine you've seen before:

```

/***** Randomize *****/

short Randomize( short range )
{
    long    randomNumber;

    randomNumber = Random();

    if ( randomNumber < 0 )
        randomNumber *= -1;

    return( (randomNumber * range) / kRandomUpperLimit );
}

```

RecalcLine() determines where to draw the next line:

```

/***** RecalcLine *****/

void RecalcLine( short i )
{
    WindowPtr    window;

    window = FrontWindow();

    gLines[ i ].top += gDeltaTop;
    if ( ( gLines[ i ].top < window->portRect.top ) ||
        ( gLines[ i ].top > window->portRect.bottom ) )
    {
        gDeltaTop *= -1;
        gLines[ i ].top += 2*gDeltaTop;
    }

    gLines[ i ].bottom += gDeltaBottom;
    if ( ( gLines[ i ].bottom < window->portRect.top ) ||
        ( gLines[ i ].bottom > window->portRect.bottom ) )
    {
        gDeltaBottom *= -1;
        gLines[ i ].bottom += 2*gDeltaBottom;
    }

    gLines[ i ].left += gDeltaLeft;
    if ( ( gLines[ i ].left < window->portRect.left ) ||

```

```

        ( gLines[ i ].left > window->portRect.right ) )
    {
        gDeltaLeft *= -1;
        gLines[ i ].left += 2*gDeltaLeft;
    }

    gLines[ i ].right += gDeltaRight;
    if ( ( gLines[ i ].right < window->portRect.left ) ||
        ( gLines[ i ].right > window->portRect.right ) )
    {
        gDeltaRight *= -1;
        gLines[ i ].right += 2*gDeltaRight;
    }
}

```

DrawLine() draws line number *i*, using the coordinates stored in gLines[ *i* ]. Because the pen mode is set to patXor, this may actually have the effect of erasing the line.

```

/***** DrawLine *****/

void    DrawLine( short i )
{
    MoveTo( gLines[ i ].left, gLines[ i ].top );
    LineTo( gLines[ i ].right, gLines[ i ].bottom );
}

```

---

## In Review

---

Whew! We've covered a lot in this chapter. We examined the basic Macintosh drawing model, QuickDraw, and showed you how to use many of the QuickDraw Toolbox routines. Now, you can read the QuickDraw chapter in *Inside Macintosh*, Volume I. Experiment with the programs presented here and try using some of the other QuickDraw routines. They're just as easy to use as the ones already covered.

Now that you understand how the Mac draws to the screen, you're ready to learn how the Mac interacts with users. Chapter 4 looks at the Event Manager—the manager that stage-directs operations.

---

# Events

*In this chapter, you'll learn about Events, the Macintosh mechanism for describing user actions to applications. When the mouse button is clicked or a key is pressed, the operating system lets your program know by queueing an event.*

---



ON THE MACINTOSH, the interaction enjoyed by users is more flexible than with any other computer. You can stop in the midst of writing a paper to do a quick spreadsheet, dial a phone, access a database, or play a game. The Mac lets you do what you want to do, when you want to do it. This is one of the guiding principles behind the Macintosh design.

The flip side of this wonderful, easy-to-use, empowering environment is that you, as a developer, have to support it. In the previous chapter, the programs you wrote responded to a mouse click. In this chapter, you'll write programs that respond to a variety of user actions. These actions, such as clicking on the mouse button or typing on the keyboard, are known as **events**.

---

## Understanding Events

---

Keeping track of events can be a pretty complicated operation. Put yourself in the place of a Mac application: When the mouse button is pressed, what do you do? If users click on the menu bar, they expect a pull-down menu to appear. If the click was in a window, they may want to select the window, bringing it to the front. They may want to resize the window or drag it to a new location on the screen. They may be trying to highlight some text, or select a portion of a picture.

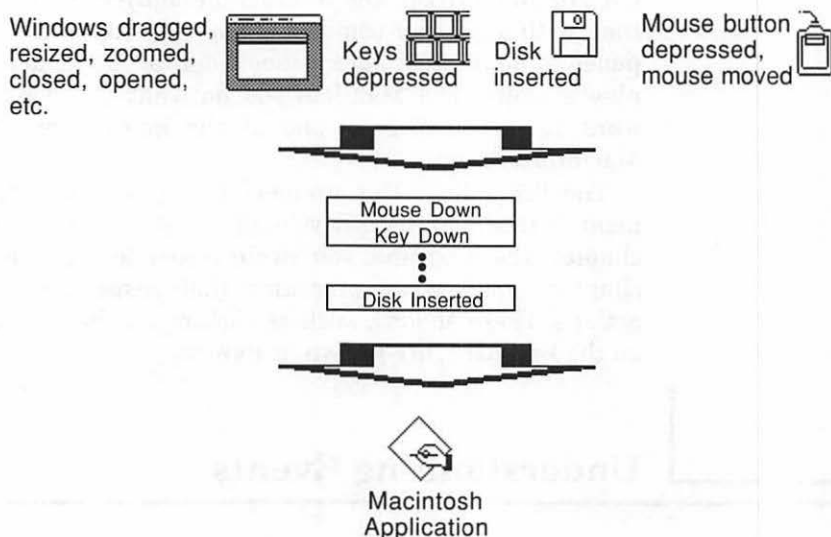
A piece of the Mac Toolbox, known as the **Event Manager**, makes it easy for you to handle user actions.

---

## The Event Manager

---

The Event Manager continually captures information about each keystroke and mouse click the user makes. It puts information about each action into an `EventRecord`. As more actions occur, additional `EventRecords` are created and joined to the first, forming an **event queue** as shown in Figure 4.1.



**Figure 4.1** The Event Queue.



The event queue is a FIFO (First In, First Out) queue: The event at the front of the queue is the oldest event in the queue. As you can see in Figure 4.1, different types of events live together in the same event queue. All events, no matter what their type, pass under the watchful eye of the Event Manager.

Let's take a look at the different kinds of events reported by the Event Manager.

## Event Types

The Event Manager handles 15 distinct events (V:249):

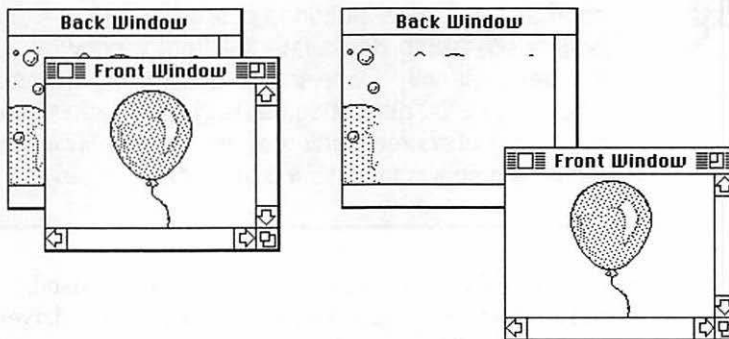
- **nullEvent:** This event is queued when the Event Manager has no other events to report.
- **mouseDown:** mouseDown events are queued whenever the mouse button is pressed. Note that the button doesn't have to be released for the event to qualify as a mouseDown.
- **mouseUp:** mouseUps are queued whenever the mouse button is released.
- **keyDown:** keyDown events are queued every time a key is pressed. Like mouseDowns, keyDowns are queued even if the key has not yet been released.

- **keyUp:** keyUps are queued whenever a key is released.
- **autoKey:** autoKey events are queued when a key is held down for a certain length of time (beyond the autoKey threshold). Usually, an autoKey event is treated just like a keyDown.



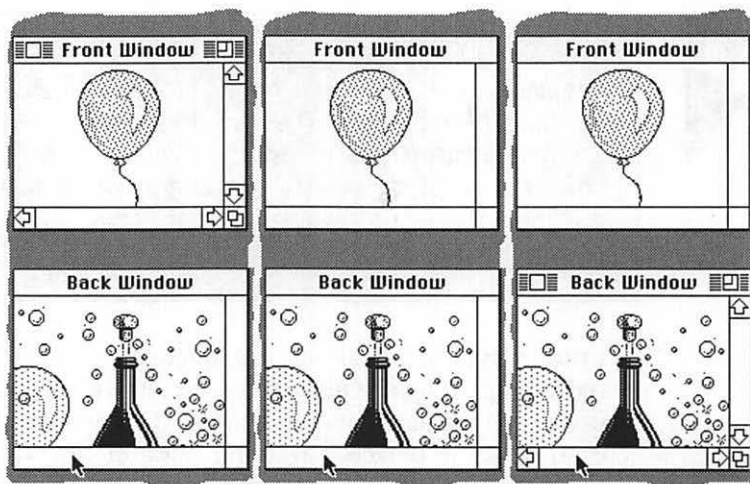
The autoKey threshold represents the time from the first keyDown until the autoKey event is generated. The default value is 16 ticks (sixtieths of a second). The autoKey rate is the interval between autoKeys. The default autoKey rate is 4 ticks. The user can change both of these in the keyboard control panel. Their values are stored in the system global variables KeyThresh and KeyRepThresh.

- **updateEvt:** updateEvs are queued whenever a window needs redrawing. They are always associated with a specific window. This usually happens when a window is partially obscured and the obstruction is moved, revealing more of the window, as shown in Figure 4.2.
- **diskEvt:** diskEvs are queued whenever a disk is inserted into a disk drive, or when an action is taken that requires a volume to be mounted.
- **activateEvt:** activateEvs are associated with windows. An activateEvt is queued whenever a window is activated (made to come to the front) or deactivated (replaced as the front-most window by another window).



**Figure 4.2** The window titled Front Window is moved to the right, leaving Back Window in need of updating.

Figure 4.3 shows the three-part sequence that occurs when one window is activated and another deactivated. Notice that while there may be more than one inactive window, there are never two active windows.

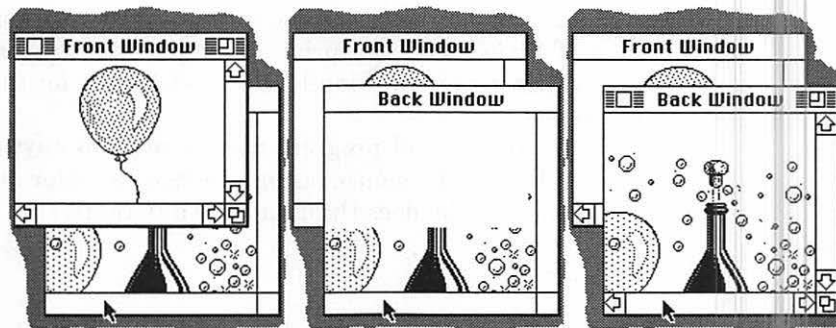


**Figure 4.3** Back Window is selected, an `activateEvt` is generated to deactivate Front Window, and an `activateEvt` is generated to activate Back Window. The front-most window is the one with the lines in the title bar.



Figure 4.4 shows what happens when a partially obscured window is brought to the front, covering the window that once covered it. As soon as the mouse button is clicked, a pair of `activateEvs` are generated, first to deactivate the front window and then to activate the back window. Once the back window moves to the front, an `updateEvt` is generated, asking your application to fill in the previously obscured portion of the window. The right-most panel in Figure 4.4 shows the finished product.

- **networkEvt:** `networkEvs` are no longer used.
- **driverEvt:** `driverEvs` are used by device drivers to signal special conditions. They (and device drivers in general) are beyond the scope of this book.
- **app1Evt, app2Evt, app3Evt:** These events can be defined by your application. The use of application-defined events is discouraged by Apple in System 7.



**Figure 4.4** Back Window is selected, an `activateEvt` is generated to deactivate Front Window, an `activateEvt` is generated to activate Back Window, and an `updateEvt` is generated to redraw Back Window.

- **osEvt** (formerly known as **app4Evt**): The system will post an `osEvt` just before it moves your application into the background (suspends it) and just after it brings your application back to the foreground (resumes it). You can also set your application up to receive `mouseMoved` events. `mouseMoved` events are posted when the user moves the cursor outside a predefined region (such as a text-editing window) or back in again. When your application receives a `mouseMoved` event, it can change the cursor to one appropriate to that region. We'll discuss `osEvts` in more detail later in this chapter.

## A New Structure for Macintosh Programming (Part 1)

In Chapter 3, we presented a primitive Macintosh application model that looked like this:

```
main()
{
    ToolboxInit();
    OtherInits();
    DoPrimeDirective();
    while ( !Button() );
}
```

First, the application model took care of any program-specific initialization, such as loading windows or pictures from the resource file.

Next, the model performs its “prime directive.” For example, in the case of ShowPICT, the prime directive was drawing a PICT in the application’s window. Finally, the model waits for the mouse button to be pressed.

Since this kind of program doesn’t react to anything that users do except clicking the mouse button, we need a better model.

The new model does things a little differently:

```
main( )
{
    ToolBoxInit();
    OtherInits();

    EventLoop();
}

EventLoop( )
{
    EventRecord      event;

    gDone = false;
    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event,
                           sleepValue, mouseRgn ) )
        {
            DoEvent( &event );
        }
    }
}

DoEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown();
            break;

        case keyDown:
            HandleKeyDown();
            break;

        .
        .
        .
    }
}
```

This model starts the same way as the basic model, with calls to the initialization routines.

The difference in the new model lies in the call of `EventLoop()`. `EventLoop()` contains the **main event loop**. The main event loop is part of the basic structure of any Mac program. Each time through the loop, your program retrieves an event from the event queue and processes the event.

Eventually, some event will cause `DoEvent()` to set `gDone` to `true`, and the program will end. This typically is the result of a `mouseDown` in the menu bar (selecting **Quit** from the **File** menu) or a `keyDown` (typing the key sequence `⌘Q`).

The most critical Event Manager routine is called `WaitNextEvent()`. To find out what events have been sent to your application, call `WaitNextEvent()` from within a loop.

```
Boolean WaitNextEvent( short eventMask, EventRecord
                      *event, long sleep, RgnHandle
                      mouseRgn );
```

Every time it is called, `WaitNextEvent()` retrieves an event from the event queue, passing the results back to your application.

## Calling WaitNextEvent()

The first parameter to `WaitNextEvent()` is an **event mask**, used to limit the types of events your program will handle. Figure 4.5 contains a list of predefined event mask constants. If your program needs only `mouseDowns` and `keyDowns`, for example, you might use the following call:

```
EventRecord event;

WaitNextEvent( (mDownMask | keyDownMask), &event, sleep,
              mouseRgn );
```

In this case, `WaitNextEvent()` will return only `mouseDown`, `keyDown`, or `nullEvent` information. As you'll remember, `nullEvents` are just the Mac OS telling you that nothing happened, so `nullEvents` are never masked out. To handle all possible events, pass the predefined constant `everyEvent` as the `eventMask` parameter. *Inside Macintosh* recommends you use `everyEvent` as your event mask in all of your applications unless there's a specific reason not to.

```
mDownMask = 2
mUpMask = 4
keyDownMask = 8
keyUpMask = 16
autoKeyMask = 32
updateMask = 64
diskMask = 128
activMask = 256
highLevelEventMask = 1024
osMask = -32768
```

**Figure 4.5** Event Masks.

The second parameter of `WaitNextEvent()` is `event`, declared as an `EventRecord`. Here's `EventRecord`'s type definition:

```
typedef struct EventRecord
{
    short    what;
    long     message;
    long     when;
    Point    where;
    short    modifiers;
}
```

Let's look at each of the fields:

- **what:** What type of event just occurred? Was it a `nullEvent`, `keyDown`, `mouseDown`, or `updateEvt`?
- **message:** This part of the `EventRecord` is specific to the event. For `keyDown` events, the `message` field contains information about the actual key that was pressed (the key code) and the character that key represents (the character code). For `activateEvs` and `updateEvs`, the `message` field contains a pointer to the affected window.
- **when:** When did the event occur? The Event Manager tells you, in ticks, when the system was last turned on (or restarted).
- **where:** Where was the mouse when the event occurred? This information is specified in global coordinates (see Chapter 3).
- **modifiers:** This part of the `EventRecord` describes the state of the mouse button and the modifier keys (the **Shift**, **Option**, **Control**, **Command**, and **Caps Lock** keys) when the event occurred.



The third parameter to `WaitNextEvent()` is the `sleep` parameter. `sleep` is a long integer that specifies the amount of time (in ticks) your application is willing *not* to perform any background processing while waiting for an event.



`sleep` should be set to `MAXLONG` (defined in the `#include` file `Values.h`) if your application doesn't need to do things in the background. If you pass a value of `0L` for `sleep`, you're telling `WaitNextEvent()` to hog the processor, which is not a neighborly thing to do.

The fourth parameter to `WaitNextEvent()` is the `mouseRgn` parameter, used to simplify cursor tracking. If your application requires different cursors, depending on which part of the screen the cursor is in, the `mouseRgn` parameter may be helpful. With it, you can specify the screen region appropriate to the current cursor. Whenever the mouse is outside the region appropriate to the current cursor, the Event Manager queues up a `mouseMoved` event. When your program receives the `mouseMoved` event, you change the region to the one that the new mouse position is within, then pass it as a parameter to the next `WaitNextEvent()` call. If your program doesn't need more than one cursor region, or if you plan to keep track of the different regions yourself, just pass `nil` in the `mouseRgn` parameter.



Calling `WaitNextEvent()` with a `sleep` value of `60` and a `mouseRgn` of `nil` is exactly equivalent to calling `SystemTask()` and `GetNextEvent()` using earlier versions of the Mac OS.

Once you have retrieved an event with `WaitNextEvent()`, you dispatch it to the appropriate procedure. For example, if the user clicked the mouse button, `DoEvent()` calls a `HandleMouseDown()` routine designed to handle `mouseDown` events. When a `keyDown` occurs, call `HandleKeyDown()`. You get the idea.

```
DoEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown();
            break;

        case keyDown:
            HandleKeyDown();
            break;

        .
        .
        .
    }
}
```

Our first program in this chapter will show you this process in detail.

## Apple Events

Now that you've seen the event-handling process, there is one more event that needs discussing: the **Apple event**. An Apple event is a special event that allows your application to communicate with other applications, such as the Finder.

You see Apple events in action all the time. For example, when you double-click on a document icon in the Finder, the Finder launches the application that created the document. Once the application is open (or, if it *was* already open), the Finder sends the application an Apple event, asking it to open the specified document. When you ask the Finder to print a document (by clicking on the document icon and selecting **Print**), the Finder sends an Apple event to the appropriate application, asking it to print the specified document.

There are four Apple events that your application must support. Together, these are known as the **required Apple Events**. They are `kAEOpenApplication`, `kAEOpenDocuments`, `kAEPrintDocuments`, and `kAEQuitApplication`. `kAEOpenApplication` asks your application to perform whatever functions it normally performs when it first starts up. `kAEOpenDocuments` and `kAEPrintDocuments` ask your application to open or print a specified document. Finally, `kAEQuitApplication` asks your program to close all open documents and quit.

Your programs, at a minimum, will need to understand how to handle these four required Apple events. To do this, we'll need to adjust the event loop model presented earlier.

## A New Structure for Macintosh Programming (Part 2)

---

Our original event loop model was based on the value in the `EventRecord`'s `what` field. If `event.what` contains a `mouseDown`, we call the function `HandleMouseDown()`. If `event.what` contains a `keyDown`, jump to some code that handles a `keyDown`.

The new model adds a new twist. *Before* you enter the main event loop, you'll tell the Event Manager which Apple events you plan to handle, and which routines you plan to handle them with. You might pair up the Apple event `AEOpenApplication` with the routine `DoAEOpen()`. Once you assign a routine to an Apple event, the Event Manager takes over. From then on, whenever that Apple event occurs, the Event Manager automatically calls the routine you assigned to that Apple event. A routine assigned to a particular Apple event is known as an **Apple event handler**.

Use the routine `AEInstallEventHandler()` to pair up an Apple event with its handler. Since, at the very least, you'll handle the four required Apple events, you'll call `AEInstallEventHandler()` at least four times. We'll get to the specifics of calling `AEInstallEventHandler()` shortly.

Once you've installed your Apple event handlers, you'll enter the main event loop. Once inside, you'll process events as usual, using the value in `event.what` to determine your next course of action. In addition to the 15 events presented above, a sixteenth event constant has been added.

Whenever an Apple event occurs, the Event Manager places the constant `kHighLevelEvent` in `event.what`. When this happens, all you do is pass the event on to the routine `AEProcessAppleEvent()`. `AEProcessAppleEvent()` automatically calls the event handler you've installed for that Apple event.



Why `kHighLevelEvent`? In general, the 15 events described earlier are known as **low-level events**. Low-level events tend to be physical in nature (a mouse is clicked, a key is pressed) and describe the relationship between the user and your application. **High-level events**, on the other hand, are designed to support communication between applications. Apple events are a subset of high-level events. Once you get the hang of Apple event programming, you might want to take a crack at designing your own high-level event protocol. To learn more about high-level events, take a look at (VI:5-8).



EventLoop() is the same as before.

```
EventLoop()
{
    EventRecord    event;
    gDone = false;
    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event,
                           sleepValue, mouseRgn ) )
        {
            DoEvent( &event );
        }
    }
}
```

The first thing DoEvent() does is check to see if the event is an Apple event. If so, it passes the event on to AEPProcessAppleEvent(). Assuming you've already installed a handler for this event, AEPProcessAppleEvent() will call your handler for you. If the event wasn't an Apple event, DoEvent() handles the event as it did before.

#### Old Event Loop

```
DoEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown();
            break;

        case keyDown:
            HandleKeyDown();
            break;
        .
        .
        .
    }
}
```

#### New Event Loop with Apple Events

```
DoEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
        case kHighLevelEvent:
            AEPProcessAppleEvent
            ( eventPtr );
            break;

        case mouseDown:
            HandleMouseDown();
            break;
        case keyDown:
            HandleKeyDown();
            break;
        .
        .
        .
    }
}
```

## Calling AEInstallEventHandler()

To install a handler for an Apple event, you call `AEInstallEventHandler()`:

```
OSErr AEInstallEventHandler( AEEEventClass theAEEEventClass,
                             AEEEventID theAEEEventID,
                             EventHandlerProcPtr handler,
                             long handlerRefcon, Boolean
                             isSysHandler );
```

The first parameter, `theAEEEventClass`, specifies the **class** of events your handler will handle. All Apple events are grouped into classes according to functionality. For example, the four required Apple events all belong to the same class, known as the **core event class**. When installing a handler for one of the required Apple events, pass the predefined constant `kCoreEventClass` as `theAEEEventClass`.

Within an event class, the **event ID** serves to distinguish one type of Apple event from another. The constants `kAEOpenApplication`, `kAEOpenDocuments`, `kAEPrintDocuments`, and `kAEQuitApplication` are the event IDs for the four required Apple events. You'll pass one of these as the second parameter to `AEInstallEventHandler()`. These `#defines` represent the 4-byte values 'oapp', 'odoc', 'pdoc', and 'quit', respectively.

The third parameter is a function pointer, the name of the function you wrote to handle this particular type of Apple event. The fourth parameter is a reference constant for use by your handler (we just pass a value of 0L, since we don't make any special use of this parameter). The last parameter is a `Boolean` that is `true` if your handler will be used by the operating system, `false` otherwise. Typically, `isSystemAE` is set to `false`.



If you find yourself getting a little confused at this point, don't worry, that's normal. You're trying to absorb a lot in one take. Keep on reading. By the time you get through the first program, the concepts will fall into place.

Now that you've been through the Event Manager in detail, let's look at some code that implements the new model. The next section describes **EventTracker**, a program that displays the inner workings of the Event Manager.

## EventTracker

---

EventTracker uses a WIND resource to create a new window, then enters the main event loop. As events occur, EventTracker describes them in the window, drawing one line of text for each event.

EventTracker:

- Initializes the EventTracker window.
- Calls `Gestalt()` to see if Apple events are available. If not, an appropriate message is drawn in the EventTracker window.
- Continually retrieves events from the event queue, displaying a text string describing each event as it occurs. A `mouseDown` in the window's close box causes EventTracker to exit.

### Resources

Create a folder called `EventTracker` inside your Development folder. Next, launch ResEdit and click the mouse to bring up the **Open File** dialog box. Click on the **New** button. When the **New File** dialog box appears, create a resource file named `EventTracker.π.rsrc` inside the `EventTracker` folder.

### Creating a WIND Resource

EventTracker requires a single WIND resource. Select **Create New Resource** from the **Resource** menu. When prompted for a resource type, enter `WIND` and click the **OK** button. A WIND editing window will appear. Edit your WIND to match the specifications shown in Figure 4.6. If you don't see the **Height:** and **Width:** fields, select **Show Height & Width** from the **WIND** menu. Be sure to select the second window type from the left in the WIND editor's top row.

Next, select **Set 'WIND' Characteristics...** from the **WIND** menu and set the **Window title:** field to **OS Events**. Next, select **Get Resource Info** from the **Resource** menu and set the WIND's resource ID to 128. Close the window displaying the WIND resource.

You've finished creating the resources you'll need for EventTracker. Select **Quit** from the **File** menu and save your changes.

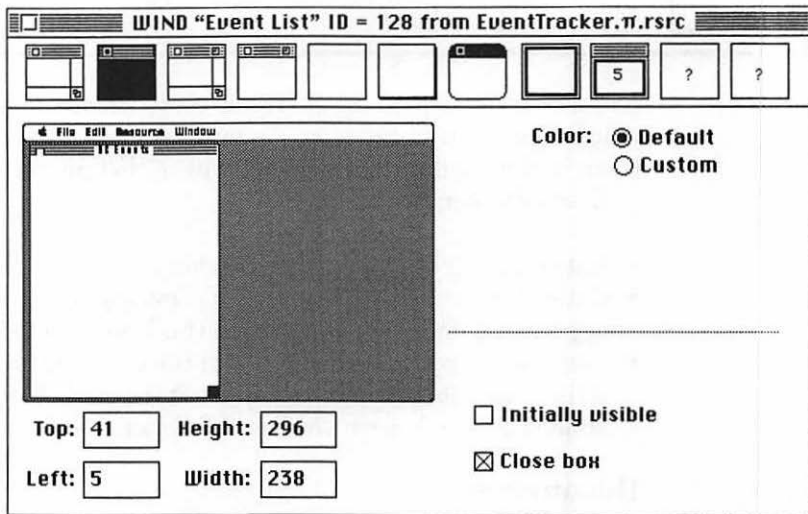


Figure 4.6 EventTracker's WIND resource.

## Setting Up the Project

Now, start up THINK C. When prompted, create a new project inside the EventTracker folder. Call the project EventTracker.π. Select **New** from the **File** menu to create a new source code file. Type in the following code:

```
#include <AppleEvents.h>
#include <GestaltEqu.h>
#include <Values.h>

#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L
#define kSleep          MAXLONG

#define kLeftMargin     4
#define kRowStart       285
#define kFontSize       9
#define kRowHeight      (kFontSize + 2)
#define kHorizontalOffset 0

#define kGestaltMask    1L
```



```

/*****/
/*  Globals  */
/*****/

Boolean          gDone;

/*****/
/*  Functions  */
/*****/

void             ToolBoxInit( void );
void             WindowInit( void );
void             EventInit( void );
void             EventLoop( void );
void             DoEvent( EventRecord *eventPtr );
pascal OSErr     DoOpenApp( AppleEvent theAppleEvent,
                           AppleEvent reply, long refCon );
pascal OSErr     DoOpenDoc( AppleEvent theAppleEvent,
                           AppleEvent reply, long refCon );
pascal OSErr     DoPrintDoc( AppleEvent theAppleEvent,
                           AppleEvent reply, long refCon );
pascal OSErr     DoQuitApp( AppleEvent theAppleEvent,
                           AppleEvent reply, long refCon );
void             DrawEventString( Str255 eventString );
void             HandleMouseDown( EventRecord *eventPtr );

/***** main *****/

void main( void )
{
    ToolBoxInit();
    WindowInit();
    EventInit();

    EventLoop();
}

/***** ToolBoxInit */

void ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
}
```

```

        InitMenus();
        TEInit();
        InitDialogs( nil );
        InitCursor();
    }

    /***** WindowInit *****/

void WindowInit( void )
{
    WindowPtr    window;
    Rect          windRect;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 ); /* Couldn't load the WIND
                        resource!!! */
        ExitToShell();
    }

    SetPort( window );
    TextSize( kFontSize );

    ShowWindow( window );
}

    /***** EventInit *****/

void EventInit( void )
{
    OSErr err;
    long  feature;

    err = Gestalt( gestaltAppleEventsAttr, &feature );

    if ( err != noErr )
    {
        DrawEventString( "\pProblem in calling Gestalt!" );
        return;
    }
    else
    {

```

```

        if ( !( feature & ( kGestaltMask <<
                                gestaltAppleEventsPresent ) ) )
        {
            DrawEventString
                ( "\pApple events not available!" );
            return;
        }
    }

    err = AEInstallEventHandler ( kCoreEventClass,
        kAEOpenApplication, DoOpenApp, 0L, false );
    if ( err != noErr ) DrawEventString
        ( "\pkAEOpenApplication Apple event not available!" );

    err = AEInstallEventHandler( kCoreEventClass,
        kAEOpenDocuments, DoOpenDoc, 0L, false );
    if ( err != noErr ) DrawEventString
        ( "\pkAEOpenDocuments Apple event not available!" );

    err = AEInstallEventHandler( kCoreEventClass,
        kAEPrintDocuments, DoPrintDoc, 0L, false );
    if ( err != noErr ) DrawEventString
        ( "\pkAEPrintDocuments Apple event not available!" );

    err = AEInstallEventHandler( kCoreEventClass,
        kAEQuitApplication, DoQuitApp, 0L, false );
    if ( err != noErr ) DrawEventString
        ( "\pkAEQuitApplication Apple event not available!" );
}

/***** EventLoop *****/

void    EventLoop( void )
{
    EventRecord  event;

    gDone = false;
    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event,
                            kSleep, nil ) )
            DoEvent( &event );
    }
}

```

```

        /*else DrawEventString( "\pnulEvent" );*/
        /* Uncomment the previous line for a burst of
           flavor! */
    }
}

/***** DoEvent */

void DoEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
        case kHighLevelEvent:
            DrawEventString( "\pHigh level event: " );
            AEProcessAppleEvent( eventPtr );
            break;
        case mouseDown:
            DrawEventString( "\pmouseDown" );
            HandleMouseDown( eventPtr );
            break;
        case mouseUp:
            DrawEventString( "\pmouseUp" );
            break;
        case keyDown:
            DrawEventString( "\pkeyDown" );
            break;
        case keyUp:
            DrawEventString( "\pkeyUp" );
            break;
        case autoKey:
            DrawEventString( "\pautoKey" );
            break;
        case updateEvt:
            DrawEventString( "\pupdateEvt" );
            BeginUpdate( (WindowPtr)eventPtr->message );
            EndUpdate( (WindowPtr)eventPtr->message );
            break;
        case diskEvt:
            DrawEventString( "\pdiskEvt" );
            break;
        case activateEvt:
            DrawEventString( "\pactivateEvt" );
            break;
    }
}

```

```

        case networkEvt:
            DrawEventString( "\pnetworkEvt" );
            break;
        case driverEvt:
            DrawEventString( "\pdriverEvt" );
            break;
        case applEvt:
            DrawEventString( "\papplEvt" );
            break;
        case app2Evt:
            DrawEventString( "\papp2Evt" );
            break;
        case app3Evt:
            DrawEventString( "\papp3Evt" );
            break;
        case osEvt:
            DrawEventString( "\posEvt: " );
            if ( (eventPtr->message & suspendResumeMessage)
                == resumeFlag )
                DrawString( "\pResume event" );
            else
                DrawString( "\pSuspend event" );
            break;
    }
}

/***** DoOpenApp */
pascal OSErr DoOpenApp( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon )
{
    DrawString( "\pApple event: kAEOpenApplication" );
}

/***** DoOpenDoc */
pascal OSErr DoOpenDoc( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon )
{
    DrawString( "\pApple event: kAEOpenDocuments" );
}

```

```

/***** DoPrintDoc */

pascal OSErr DoPrintDoc( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon )
{
    DrawString( "\pApple event: kAEPrintDocuments" );
}

/***** DoQuitApp */

pascal OSErr DoQuitApp( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon )
{
    DrawString( "\pApple event: kAEQuitApplication" );
}

/***** DrawEventString */

void DrawEventString( Str255 eventString )
{
    RgnHandle    tempRgn;
    WindowPtr    window;

    window = FrontWindow();
    tempRgn = NewRgn();
    ScrollRect( &window->portRect, kHorizontalOffset,
                -kRowHeight, tempRgn );
    DisposeRgn( tempRgn );

    MoveTo( kLeftMargin, kRowStart );
    DrawString( eventString );
}

/***** HandleMouseDown */

void HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr    window;
    long          thePart;

    thePart = FindWindow( eventPtr->where, &window );

    switch ( thePart )
    {

```

```

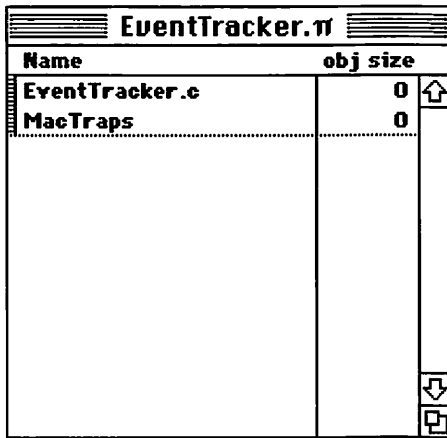
        case inSysWindow :
            SystemClick( eventPtr, window );
            break;
        case inDrag :
            DragWindow( window, eventPtr->where,
                        &screenBits.bounds );

            break;
        case inGoAway :
            gDone = true;
            break;
    }
}

```

## Running EventTracker

Save your source code inside the EventTracker folder as EventTracker.c. Select **Add** from the **Source** menu to add EventTracker.c to the project (don't forget to add MacTraps, as well). The **Project** window should now look like Figure 4.7.



**Figure 4.7** Project window for EventTracker.

There's one more thing you'll need to do for the EventTracker project before you run it. You'll need to create a 'SIZE' resource. The 'SIZE' resource tells the operating system how well your application interacts with the outside world. Can your program run in the background? Is it 32-bit compatible? Will it handle Apple events?

While most resources are created using ResEdit, THINK C will generate this resource for you. To create it, select **Set Project Type...** from THINK C's **Project** menu. You should see the dialog box in Figure 4.8.

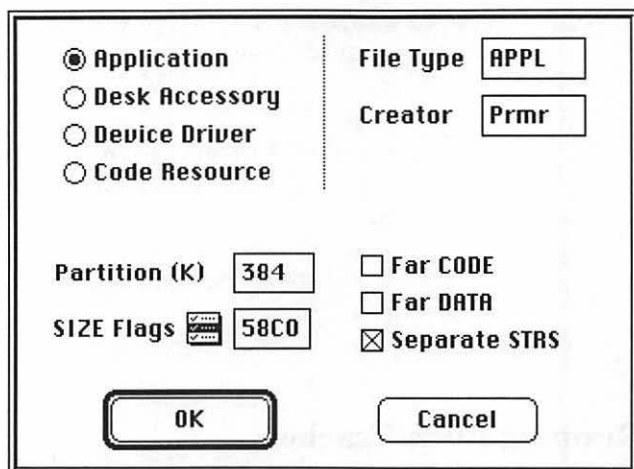


Figure 4.8 Set Project Type... dialog box.

Click on the pop-up menu area to the right of **SIZE Flags** in the dialog box. If you hold the button down, you'll see the menu in Figure 4.9. Set the check marks as shown and click on **OK** to save your changes.



When you set those flags, you're telling the Mac OS that EventTracker uses `OSEvts` and Apple events. The 'SIZE' resource flags will be discussed in more detail in Chapter 8.

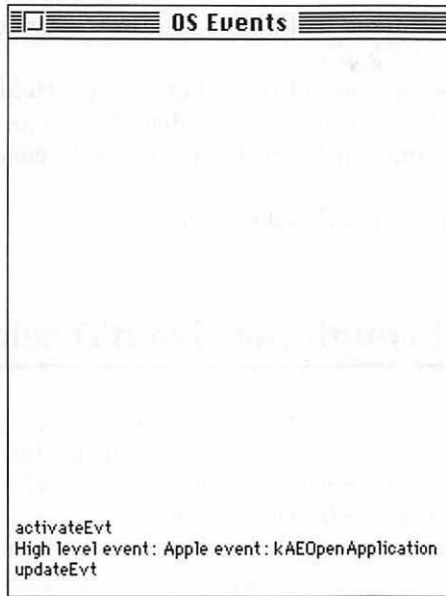


Figure 4.9 SIZE Flags.



Now you're ready. Select **Run** from the **Project** menu. Click **Yes** when asked to **Bring the project up to date?** If everything went well, the EventTracker window should appear on the desktop.

Figure 4.10 shows how the EventTracker window should look:



**Figure 4.10** Project window for EventTracker.

It should already list three events:

- `activateEvt`: This event occurs when you make the window visible with `ShowWindow()`.
- High level event: Apple event: `kAEOpenApplication`: Whenever the Finder launches an application, it sends the application a `kAEOpenApplication` Apple event.
- `updateEvt`: When the Window Manager creates a window, it draws the window, then generates an `updateEvt` for the window.



When the Window Manager draws a window, it first draws the window frame. The window frame includes the border, as well as a drag region, zoom box, and a close box, if appropriate. Next, it generates an `updateEvt` for the window, asking the application to draw the window contents.

Press the mouse button in the middle of the EventTracker window. Now release the mouse button. You should see a `mouseDown` and then a `mouseUp` event. Next, click on another application, or click on the desktop to bring the Finder to the foreground. The EventTracker window will be deactivated and placed in the background; it should display an `osEvt: Suspend` event. Click on the EventTracker window again. The window will be brought to the front; it should display an `osEvt: Resume` event.

Experiment with some of the other events. Hold down a key for a few seconds (`autoKey`), or insert a diskette (`diskEvt`). When you're done experimenting, click on EventTracker's close box to exit the program.

Let's look at the EventTracker code.

## Walking Through the EventTracker Code

EventTracker starts with a few `#includes`. The file `<AppleEvents.h>` has the Apple event declarations we'll need. `<GestaltEqu.h>` is necessary to call `Gestalt()`, an important Toolbox routine that we'll discuss in the code. `<Values.h>` contains a number of useful constants; the only one we're using in EventTracker is the `#define` for `MAXLONG` used in `WaitNextEvent()`.

```
#include <AppleEvents.h>
#include <GestaltEqu.h>
#include <Values.h>
```

Some of the `#defines` should be familiar. We'll discuss each constant as it appears in the code:

```
#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L
#define kSleep              MAXLONG

#define kLeftMargin         4
#define kRowStart           285
#define kFontSize           9
#define kRowHeight          (kFontSize + 2)
#define kHorizontalOffset   0

#define kGestaltMask        1L
```

gDone is checked each time through the main event loop. If anything sets gDone to true, the program exits.

```
Boolean      gDone;
```

Here are the **EventTracker** prototypes:

```
void          ToolBoxInit( void );
void          WindowInit( void );
void          EventInit( void );
void          EventLoop( void );
void          DoEvent( EventRecord *eventPtr );
pascal OSErr  DoOpenApp( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon );
pascal OSErr  DoOpenDoc( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon );
pascal OSErr  DoPrintDoc( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon );
pascal OSErr  DoQuitApp( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon );
void          DrawEventString( Str255 eventString );
void          HandleMouseDown( EventRecord *eventPtr );
```

**EventTracker's** `main()` function starts by initializing the Toolbox and executing the window initialization routine. Next, it calls `EventInit()` to install the event handlers for the four required Apple events. Finally, **EventTracker** enters the event loop by calling `EventLoop()`.

```
/****** main *****/

void main( void )
{
    ToolBoxInit();
    WindowInit();
    EventInit();

    EventLoop();
}
```

`ToolBoxInit()` is the same as before.

```

/***** ToolBoxInit */

void ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

`WindowInit()` starts by loading a window from the resource file.

```

void WindowInit( void )
{
    WindowPtr    window;
    Rect          windRect;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

```

If the `WIND` resource isn't found, the program beeps and exits. If this happens, check the name of your resource file.

```

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND
                           resource!!! */
        ExitToShell();
    }

```

The window is made the current port, and the text size is set to `kFontSize`.

```

    SetPort( window );
    TextSize( kFontSize );

```

Finally, `ShowWindow()` makes the window visible.

```

    ShowWindow( window );
}

```

`EventInit()`, the routine which installs the Apple event handlers, starts with a call to `Gestalt()`. `Gestalt()` is passed a constant `gestaltAppleEventsAttr` that is defined in `GestaltEqu.h`. The parameter returned by `Gestalt()`, `feature`, contains information about hardware or software features on your Macintosh, specifically information about Apple events. If `Gestalt()` returns an error, the request for this information has not been granted. In this case, you can assume that Apple events are not available and return to `main()`.

```

/***** EventInit *****/

void EventInit( void )
{
    OSErr err;
    long feature;

    err = Gestalt( gestaltAppleEventsAttr, &feature );

    if ( err != noErr )
    {
        DrawEventString( "\pProblem in calling Gestalt!" );
        return;
    }
}

```

If `Gestalt()` succeeds, `feature` defines the level of support your Macintosh has for Apple events. `feature` is compared with another `#define` from `GestaltEqu.h` to determine if Apple events are available. If Apple events aren't available, use `DrawEventString()` to display that information in the window.

```

else
{
    if ( !( feature & ( kGestaltMask <<
                      gestaltAppleEventsPresent ) ) )
    {
        DrawEventString
            ( "\pApple events not available!" );
        return;
    }
}

```

Next, install an event handler for the `kAEOpenApplication` event ('oapp'). If an error occurs, `DrawEventString()` displays an error string in the `EventTracker` window. Otherwise, `DoOpenApp()` will automatically get called each time `EventTracker` receives a `kAEOpenApplication` event.

```
err = AEInstallEventHandler( kCoreEventClass,
    kAEOpenApplication, DoOpenApp, 0L, false );
if ( err != noErr ) DrawEventString
    ( "\pkAEOpenApplication Apple event not available!" );
```

The `kAEOpenDocuments` event ('odoc') is paired with the procedure `DoOpenDoc()`,

```
err = AEInstallEventHandler( kCoreEventClass,
    kAEOpenDocuments, DoOpenDoc, 0L, false );
if ( err != noErr ) DrawEventString
    ( "\pkAEOpenDocuments Apple event not available!" );
```

The `kAEPrintDocuments` event ('odoc') is paired with the procedure `DoPrintDoc()`,

```
err = AEInstallEventHandler( kCoreEventClass,
    kAEPrintDocuments DoPrintDoc, 0L, false );
if ( err != noErr ) DrawEventString
    ( "\pkAEPrintDocuments Apple event not available!" );
```

and the `kAEQuitApplication` event ('quit') is paired with the procedure `DoQuitApp()`.

```
err = AEInstallEventHandler( kCoreEventClass,
    kAEQuitApplication, DoQuitApp, 0L, false );
if ( err != noErr ) DrawEventString
    ( "\pkAEQuitApplication Apple event not available!" );
}
```

`EventLoop()` starts by initializing `gDone`. Then, while `gDone` is false, `WaitNextEvent()` is called. If no event was available, `WaitNextEvent()` will generate a nullEvt and return false. If you uncomment the call to `DrawEventString()`, you'll get an idea of how frequently this happens. Otherwise, `DoEvent()` is called to process the event.



Gestalt() looks like this:

```
err = Gestalt( OSType selectorCode, long
               &response );
```

To use Gestalt(), you pass it a **selector code** in the first parameter that checks for the availability of a specific feature. In the second parameter, Gestalt() returns a response that you can then interpret and use.

Selector codes have different suffixes, each of which signals a different type of response.

Suffix	Meaning
Attr	Response consists of 32 bitflags that must be compared against constants defined in Gestalt.h.
Count	Response is the total of the kind of type specified.
Table	Response is the base address to a table.
Type	Response is an index describing a particular type of feature.
Version	Response is a version number.

See *Inside Macintosh* Volume VI, 3-35 to 3-53 for more information about Gestalt().

```
/***** EventLoop *****/
```

```
void EventLoop( void )
{
    EventRecord    event;

    gDone = false;
    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event,
                           kSleep, nil ) )
            DoEvent( &event );
    }
}
```

```

        /*else DrawEventString( "\pnullEvent" );*/
        /* Uncomment the previous line for a burst of
           flavor! */
    }
}

```



`nullEvents` are used in the program `WorldClock` in the next chapter; they offer an excellent opportunity to deal with such things as cursor tracking and internal housekeeping.

`DoEvent()` uses `eventPtr->what` to pass control to the appropriate handler. If the event is an Apple event, the string `High level event:` is drawn in the window, and `AEProcessAppleEvent()` is called. `AEProcessAppleEvent()` passes the Apple event to the event handler installed in `EventInit()`. At this point, the only Apple event you'll see is the `kAEOpenApplication` Apple event sent by the Finder when the application started up.

```

/***** DoEvent *****/

void DoEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
        case kHighLevelEvent:
            DrawEventString( "\pHigh level event: " );
            AEProcessAppleEvent( eventPtr );
            break;

```

If the mouse button has been clicked, or keys have been depressed, the appropriate string is drawn. In addition, `HandleMouseDown()` is called when a `mouseDown` has been detected.

```

        case mouseDown:
            DrawEventString( "\pmouseDown" );
            HandleMouseDown( eventPtr );
            break;
        case mouseUp:
            DrawEventString( "\pmouseUp" );
            break;

```



```
case keyDown:
    DrawEventString( "\\pkeyDown" );
    break;
case keyUp:
    DrawEventString( "\\pkeyUp" );
    break;
case autoKey:
    DrawEventString( "\\pautoKey" );
    break;
```

If there's an `updateEvt`, the `EventTracker` window needs to be redrawn. Since you're not concerned with maintaining the window contents in this program, just call `BeginUpdate()` and `EndUpdate()` to remove the `updateEvt` from the event queue.



If you were to comment out the calls to `BeginUpdate()` and `EndUpdate()`, you'd get an unending stream of `updateEvs` for the window. The Window Manager, thinking you were ignoring the ones you'd already retrieved, would just keep generating them.

```
case updateEvt:
    DrawEventString( "\\pupdateEvt" );
    BeginUpdate( (WindowPtr)eventPtr->message );
    EndUpdate( (WindowPtr)eventPtr->message );
    break;
```

Continue drawing the event strings in the window.

```
case diskEvt:
    DrawEventString( "\\pdiskEvt" );
    break;
case activateEvt:
    DrawEventString( "\\pactivateEvt" );
    break;
case networkEvt:
    DrawEventString( "\\pnetworkEvt" );
    break;
case driverEvt:
    DrawEventString( "\\pdriverEvt" );
    break;
```

```

case app1Evt:
    DrawEventString( "\papp1Evt" );
    break;
case app2Evt:
    DrawEventString( "\papp2Evt" );
    break;
case app3Evt:
    DrawEventString( "\papp3Evt" );
    break;

```

Finally, if there's an `osEvt`, call `DrawString()` to display the mode of the event. Resume event is displayed in the window if `EventTracker` has been brought to the foreground, and Suspend event if `EventTracker` has been sent to the background.

```

case osEvt:
    DrawEventString( "\posEvt: " );
    if ( ( eventPtr->message &
          suspendResumeMessage ) == resumeFlag )
        DrawString( "\pResume event" );
    else
        DrawString( "\pSuspend event" );
    break;
}
}

```

Next are the handlers for the four required Apple events. Each routine handles the exciting job of drawing a text description of itself in the `EventTracker` window.

```

/***** DoOpenApp */

pascal OSErr DoOpenApp( AppleEvent theAppleEvent,
                       AppleEvent reply, long refCon )
{
    DrawString( "\pApple event: kAEOpenApplication" );
}

/***** DoOpenDoc */

pascal OSErr DoOpenDoc( AppleEvent theAppleEvent,
                       AppleEvent reply, long refCon )
{
    DrawString( "\pApple event: kAEOpenDocuments" );
}

```

```

/***** DoPrintDoc */

pascal OSErr DoPrintDoc( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon )
{
    DrawString( "\pApple event: kAEPrintDocuments" );
}

```

```

/***** DoQuitApp */

pascal OSErr DoQuitApp( AppleEvent theAppleEvent,
                       AppleEvent reply, long refCon )
{
    DrawString( "\pApple event: kAEQuitApplication" );
}

```

**DrawEventString()** positions and draws the text strings in the window. First, **FrontWindow()** returns a pointer to the **EventTracker** window.

```

/***** DrawEventString */

void DrawEventString( Str255 eventString )
{
    RgnHandle    tempRgn;
    WindowPtr    window;

    window = FrontWindow();

```

Then, **ScrollRect()** scrolls the contents of the current **GrafPort** within the rectangle specified in the first parameter. The rectangle is scrolled to the right by the number of pixels specified in the second parameter (**kHorizontalOffset**, which is 0; we're just scrolling up, not across) and down by the number of pixels specified in the third parameter (**-kRowHeight**). Because you specified a negative **kRowHeight**, the contents of the window scroll up.

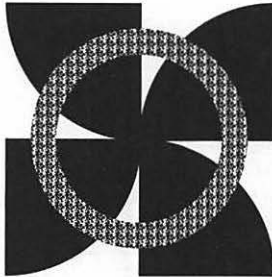
```

    tempRgn = NewRgn();
    ScrollRect( &window->portRect, kHorizontalOffset,
               -kRowHeight, tempRgn );
    DisposeRgn( tempRgn );

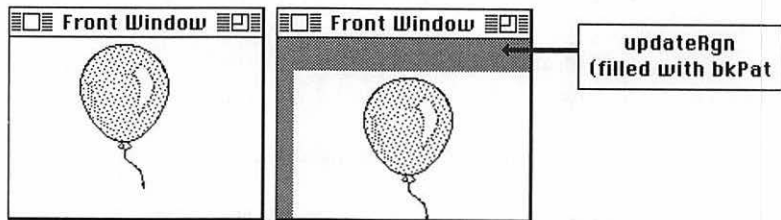
```



The last parameter to `ScrollRect()` is a `RgnHandle`, or a handle to a **region**. Regions are collections of drawn lines, shapes, and curves, as shown in Figure 4.11 (we discussed them briefly in Chapter 3). After the pixels in the rectangle are scrolled, `ScrollRect()` fills the vacated area of the rectangle with the `GrafPort`'s background pattern. These new areas are collected into the region handled by `updateRgn` (Figure 4.12).



**Figure 4.11** A region.



**Figure 4.12** Front Window's `updateRgn` after `ScrollRect(&r, 10, 20, updateRgn)`.



Many programs use this region as a guide to redrawing the window so that they don't have to redraw the entire window. This is especially useful if your window is extremely complex and takes a long time to redraw. In that case, a handle to the window's `updateRgn` is passed to `ScrollRect()`. Whenever the Window Manager detects that a window's `updateRgn` is nonempty, the Window Manager generates an `updateEvt` for the window. As part of its processing, `BeginUpdate()` sets the specified window's `updateRgn` to the empty region. Since we're not redrawing our window in response to an `updateEvt` (yet), `EventTracker` just disposes of the `tempRgn`.

`MoveTo()` positions the `QuickDraw` pen in the bottom left-hand corner of the scrolled window. Finally, `DrawString()` draws the string describing the event in the window.

```
MoveTo( kLeftMargin, kRowStart );
DrawString( eventString );
}
```

Now let's look at the `HandleMouseDown()` routine, where all the action is.

## Handling mouseDown Events in EventTracker

When you receive a `mouseDown` event, the first thing to do is find out which window the mouse was clicked in. Do this by calling the Toolbox routine `FindWindow()`. `FindWindow()` takes, as input, a point on the screen. It then returns, in the parameter `window`, a `WindowPtr` to the window containing the point. In addition, `FindWindow()` returns an integer part code that describes the part of the window in which the point was located.

Once you have your part code, compare it to the predefined Toolbox part codes. (You can find a list of legal part codes in I:287.)

```
/* ***** HandleMouseDown */

void HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr    window;
    long          thePart;
```

```

thePart = FindWindow( eventPtr->where, &window );

switch ( thePart )
{

```

The part code `inSysWindow` means that the mouse was clicked in a system window, very likely a desk accessory. (Because `EventTracker` doesn't support desk accessories, you probably won't see any `inSysWindow` mouseDowns.) In this case, pass the event and the `WindowPtr` to the system so that it can handle the event. Do this with the Toolbox routine `SystemClick()`.

```

    case inSysWindow :
        SystemClick( eventPtr, window );
        break;

```

The part code `inDrag` indicates a mouse click in the window's drag region. Handle this with a call to the Toolbox routine `DragWindow()`. `DragWindow()` needs a `WindowPtr` (the point on the screen where the mouse was clicked), and a boundary rectangle. `DragWindow()` will allow the user to drag the window anywhere on the screen as long as it's within the boundary rectangle. Use `screenBits.bounds`, which will let you drag the window anywhere on the desktop.

```

    case inDrag :
        DragWindow( window, eventPtr->where,
                    &screenBits.bounds );
        break;

```

The part code `inGoAway` indicates a mouse click in the close box of the window. `gDone` is set to `true`, so that `EventTracker` quits. Quitting an application by clicking in the close box isn't exactly kosher, but we'll need menus to quit the right way. You'll see them in Chapter 5.

```

    case inGoAway :
        gDone = true;
        break;
}
}

```

## Building the EventTracker Application

Since you'll need EventTracker later in this chapter as a standalone application, let's build it now. Open the EventTracker project and select the **Set Project Type...** menu item in the **Project** menu. When the dialog is displayed, type `Prmr` into the **Creator** field, as shown in Figure 4.13.

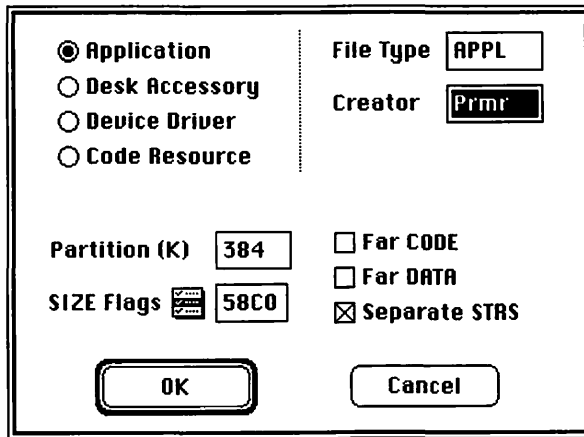


Figure 4.13 Changing the Creator.

Now select the **Build Application...** menu item in the **Project** menu. As shown in Figure 4.14, name the application EventTracker and click on the **Save** button.

Hang onto your EventTracker application—you'll need it later in this chapter.

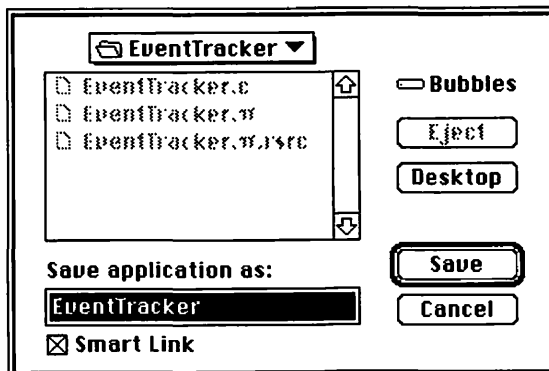


Figure 4.14 Building the EventTracker application.

Figuring out how to properly redraw a window when it has been obscured by other windows can be tricky. You see this demonstrated in Updater, the program in the next section.

---

## Updater: The Return of ShowPICT

---

Chapter 3's ShowPICT program showed you how to draw a PICT resource in a window. As soon as you clicked the mouse, the program exited. Our next project is a little more ambitious.

**Updater** starts with two PICT resources, drawn in two separate windows. Updater allows you to drag, resize, and zoom each of the windows. Click on the back one, it moves to the front. Updater also demonstrates proper handling of update and activate events.

### Building Updater

Create a folder called Updater inside your Development folder. Next, launch ResEdit and click the mouse to bring up the **Open File** dialog box. Click on the **New** button. When the **New File** dialog box appears, open the Updater folder and create a resource file named Updater. $\pi$ .rsrc inside the Updater folder.

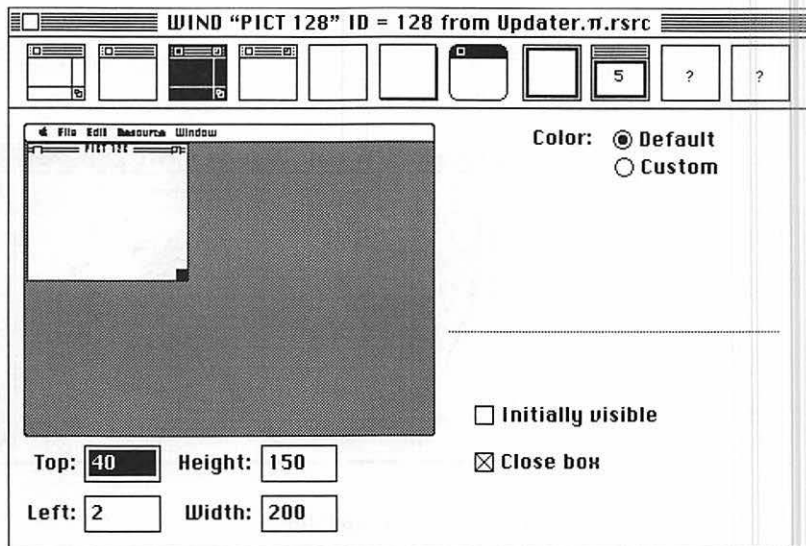
### Updater Resources

Updater needs two WIND resources. Select **Create New Resource** from the **Resource** menu. When prompted for a resource type, enter WIND and click the **OK** button. A WIND editing window will appear. Edit your WIND to match the specifications shown in Figure 4.15. Be sure to select the third window type from the left in the WIND editor's top row.

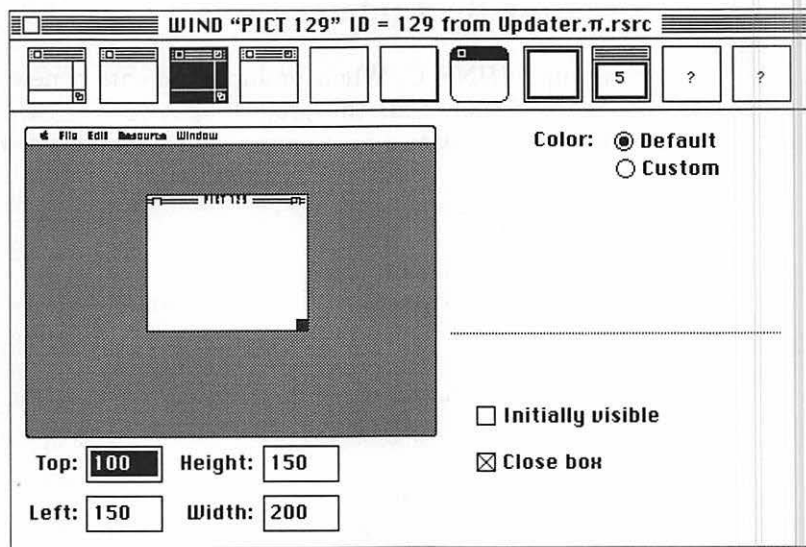
Next, select **Set 'WIND' Characteristics...** from the **WIND** menu and set the **Window title:** field to **PICT 128**. If necessary, select **Get Resource Info** from the **Resource** menu and set the WIND's resource ID to 128.

Create a second WIND resource in the same way, using the specifications in Figure 4.16. Set the title of this window to **PICT 129** and the resource ID to 129.



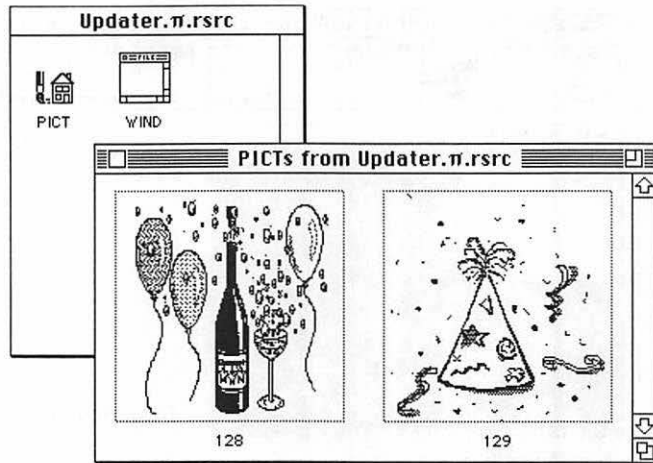


**Figure 4.15** First window for Updater.



**Figure 4.16** Second window for Updater.

Finally, you'll add two PICTs to your resource file. It's easiest to copy and paste each one directly from the Scrapbook into Updater.π.rsrc, as we did in Figure 4.17. Make sure the PICT resource IDs are 128 and 129.



**Figure 4.17** PICTs for Updater.

When you've finished with the two PICT resources, save the resource file and quit ResEdit. It's time to code!

## Creating the Updater Project

Start up THINK C. When prompted, create a new project inside the Updater folder. Call the project Updater.π. Select **New** from the **File** menu to create a new source code file, and type in the code below:

```
#include <Values.h>

#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

#define kScrollbarAdjust (16-1)
#define kLeaveWhereItIs  false
#define kNormalUpdates   true

#define kMinWindowHeight 50
#define kMinWindowWidth  80

/*****
/*  Globals
*****/

Boolean      gDone;
```

```

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
void    WindowInit( void );
void    EventLoop( void );
void    DoEvent( EventRecord *eventPtr );
void    HandleMouseDown( EventRecord *eventPtr );
void    DoUpdate( EventRecord *eventPtr );
void    DoActivate( WindowPtr window, Boolean
                    becomingActive );
void    DoPicture( WindowPtr window, PicHandle picture );
void    CenterPict( PicHandle picture, Rect *destRectPtr );


/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();

    EventLoop();
}


/***** ToolBoxInit */

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}


/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;
```

```

window = GetNewWindow( kBaseResID, nil, kMoveToFront );

if ( window == nil )
{
    SysBeep( 10 );    /* Couldn't load the WIND
                       resource!!! */
    ExitToShell();
}

SetWRefCon ( window, (long)kBaseResID );
ShowWindow( window );

window = GetNewWindow( kBaseResID+1, nil, kMoveToFront);

if ( window == nil )
{
    SysBeep( 10 );    /* Couldn't load the WIND
                       resource!!! */
    ExitToShell();
}

SetWRefCon ( window, (long)( kBaseResID+1 ) );
ShowWindow( window );
}

/***** EventLoop *****/

void    EventLoop( void )
{
    EventRecord  event;

    gDone = false;
    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, MAXLONG,
                           nil ) )
            DoEvent( &event );
    }
}

```

```

/***** DoEvent */

void      DoEvent( EventRecord *eventPtr )
{
    Boolean becomingActive;

    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
        case updateEvt:
            DoUpdate( eventPtr );
            break;
        case activateEvt:
            becomingActive = ( (eventPtr->modifiers &
                               activeFlag) == activeFlag );

            DoActivate( (WindowPtr)eventPtr->message,
                        becomingActive );
            break;
    }
}

/***** HandleMouseDown */

void      HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr    window;
    short int    thePart;
    GrafPtr      oldPort;
    long         windSize;
    Rect         growRect;

    thePart = FindWindow( eventPtr->where, &window );

    switch ( thePart )
    {
        case inSysWindow :
            SystemClick( eventPtr, window );
            break;
        case inContent:
            SelectWindow( window );
            break;
    }
}
```

```

        case inDrag :
            DragWindow( window, eventPtr->where,
                        &screenBits.bounds );

            break;
        case inGoAway :
            if ( TrackGoAway( window, eventPtr->where ) )
                gDone = true;

            break;
        case inGrow:
            growRect.top = kMinWindowHeight;
            growRect.left = kMinWindowWidth;
            growRect.bottom = MAXSHORT;
            growRect.right = MAXSHORT;

            windSize = GrowWindow( window,
                                   eventPtr->where, &growRect );
            if ( windSize != 0 )
            {
                GetPort( &oldPort );
                SetPort( window );
                EraseRect( &window->portRect );
                SizeWindow( window, LoWord( windSize ),
                           HiWord( windSize ), kNormalUpdates );
                InvalRect( &window->portRect );
                SetPort( oldPort );
            }

            break;
        case inZoomIn:
        case inZoomOut:
            if ( TrackBox( window, eventPtr->where,
                           thePart ) )
            {
                GetPort( &oldPort );
                SetPort( window );
                EraseRect( &window->portRect );
                ZoomWindow( window, thePart,
                           kLeaveWhereItIs );
                InvalRect( &window->portRect );
                SetPort( oldPort );
            }

            break;
    }
}

```

```

/***** DoUpdate */

void      DoUpdate( EventRecord *eventPtr )
{
    short      pictureID;
    PicHandle   picture;
    WindowPtr   window;

    window = (WindowPtr)eventPtr->message;

    BeginUpdate( window );
    pictureID = GetWRefCon ( window );
    picture = GetPicture( pictureID );

    if ( picture == nil )
    {
        SysBeep( 10 );    /* Couldn't load the PICT
                           resource!!! */
        ExitToShell();
    }

    DoPicture( window, picture );
    EndUpdate( window );
}

/***** DoActivate */

void      DoActivate( WindowPtr window, Boolean
                     becomingActive )
{
    DrawGrowIcon( window );
}

/***** DoPicture *****/

void      DoPicture( WindowPtr window, PicHandle picture )
{
    Rect      drawingClipRect, windowRect;
    GrafPtr   oldPort;
    RgnHandle  tempRgn;

    GetPort( &oldPort );
    SetPort( window );

```

```

    tempRgn = NewRgn();
    GetClip( tempRgn );
    EraseRect( &window->portRect );

    DrawGrowIcon( window );

    drawingClipRect = window->portRect;
    drawingClipRect.right -= kScrollbarAdjust;
    drawingClipRect.bottom -= kScrollbarAdjust;

    windowRect = window->portRect;
    CenterPict( picture, &windowRect );
    ClipRect( &drawingClipRect );
    DrawPicture( picture, &windowRect );

    SetClip( tempRgn );
    DisposeRgn( tempRgn );
    SetPort( oldPort );
}

/***** CenterPict *****/

void    CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

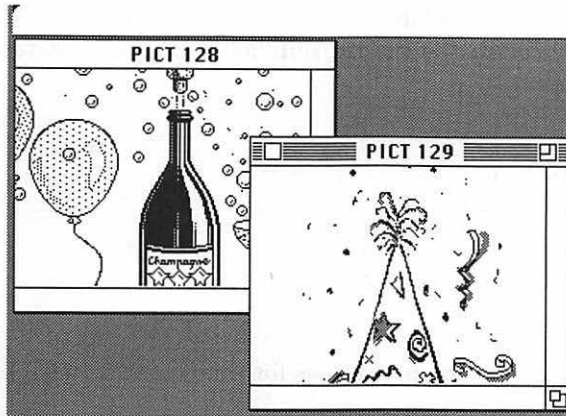
    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top );
    OffsetRect( &pictRect, (windRect.right -
                pictRect.right)/2, (windRect.bottom -
                pictRect.bottom)/2 );
    *destRectPtr = pictRect;
}

```

## Running Updater

Now that your source code is in, you're ready to run Updater. Save your source code in the Updater folder as Updater.c. Select **Add** from the **Source** menu to add Updater.c to the project (don't forget to add MacTraps, as well). Select **Run** from the **Project** menu. When asked to **Bring the project up to date?** click **Yes**. Two new windows should appear, each displaying a PICT resource, as shown in Figure 4.18.





**Figure 4.18** The two Updater windows.

Updater demonstrates just how much the Window Manager does for you. For starters, the Window Manager draws each window, starting with the window frame. The window frame includes the border, as well as a drag region, zoom box, and a close box, if appropriate. Next, it generates an `updateEvt`, asking Updater to draw the window contents.

Try clicking in a window's zoom box. Clicking the zoom box should vary the window size between the maximum size of your monitor and the last size you used. The picture should remain centered in the window. Resize a window by clicking and dragging the grow box.

As you play with Updater, notice that no matter how you manipulate the window by dragging it or resizing it, the contents are always redrawn properly. Let's take a look at the code and see how it's done.

## Walking Through the Updater Code

Updater starts with `#defines`, some of them the regular bunch of suspects. We'll discuss each constant as it appears in the code:

```
#include <Values.h>

#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

#define kScrollbarAdjust (16-1)
#define kLeaveWhereItIs  false
#define kNormalUpdates   true

#define kMinWindowHeight 50
#define kMinWindowWidth  80
```

As we saw before, `gDone` is initialized to false and checked each time through the main event loop. If `gDone` is set to true, the program exits.

```

/*****
/*  Globals  */
*****/

Boolean      gDone;

```

Next come the prototypes for the Updater routines.

```

/*****
/*  Functions  */
*****/

void  ToolBoxInit( void );
void  WindowInit( void );
void  EventLoop( void );
void  DoEvent( EventRecord *eventPtr );
void  HandleMouseDown( EventRecord *eventPtr );
void  DoUpdate( EventRecord *eventPtr );
void  DoActivate( WindowPtr window, Boolean
                becomingActive );
void  DoPicture( WindowPtr window, PicHandle picture );
void  CenterPict( PicHandle picture, Rect *destRectPtr );

```

Updater's main procedure starts by calling `ToolBoxInit()` and the window initialization routine. Next, Updater enters the main event loop by calling `EventLoop()`.

```

/***** main *****/

void  main( void )
{
    ToolBoxInit();
    WindowInit();

    EventLoop();
}

```

Here's the familiar `ToolboxInit()`,

```

/***** ToolboxInit */

void ToolboxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

`WindowInit()` starts by loading a `WIND` resource to create a new window. If the resource could not be loaded, beep once, then exit.

```

/***** WindowInit *****/

void WindowInit ( void )
{
    WindowPtr window;
    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND
                           resource!!! */
        ExitToShell();
    }
}

```

Next, we'll tuck away the resource ID of the `PICT` we want drawn in this window. Later, when it comes time to draw the window's contents, we'll retrieve this ID and draw the `PICT`.

The Window Manager reserves a cache of 4 bytes for your own personal use as part of each window you create. To place a value in those 4 bytes, call `SetWRefCon()`, passing it a `WindowPtr` and a long. To retrieve the value, call `GetWRefCon()`.

```
SetWRefCon ( window, (long)kBaseResID );
```



The `refCon`, short for **reference constant**, is found in many Toolbox data structures. It's just a place where you can stash some information away safely. The `refCon` is an invaluable tool. Take advantage of it!

Once the reference constant is in place, make the window visible.

```
ShowWindow( window );
```

The same thing is done for the second window, except that we use a different window resource and reference constant.

```

window = GetNewWindow( kBaseResID+1, nil,
                      MoveToFront );

if ( window == nil )
{
    SysBeep( 10 );    /* Couldn't load the WIND
                      resource!!! */
    ExitToShell();
}

SetWRefCon ( window, (long)( kBaseResID+1 ) );
ShowWindow( window );
}

```

EventLoop() loops on DoEvent() until gDone is set to true.

```

void    EventLoop( void )
{
    EventRecord    event;

    gDone = false;
    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, MAXLONG,
                          nil ) )
            DoEvent( &event );
    }
}

```

As before, DoEvent() starts with a call to WaitNextEvent(). Unlike EventTracker, however, Updater handles only mouseDown, update, and activate events.

```

void    DoEvent( EventRecord *eventPtr )
{
    Boolean    becomingActive;

    switch ( eventPtr->what )
    {

```

```

case mouseDown:
    HandleMouseDown( eventPtr );
    break;
case updateEvt:
    DoUpdate( eventPtr );
    break;

```

Although we don't take advantage of it in this program, the Boolean `becomingActive` plays a key role in proper `activateEvt` handling. The event's `activeFlag` is set (and `becomingActive` set to true) if the `activateEvt` is in response to a window becoming active (moving to the front). If the window is being deactivated, `becomingActive` will be set to false.



Do you need `becomingActive`? You do if your windows change as they activate or deactivate. For example, if you write on a word processor, you'll want the front-most window to feature a blinking text cursor, and highlighted text. As a window activates, you'll turn on these features. As the window deactivates, you'll turn them back off.

```

case activateEvt:
    becomingActive = ( (eventPtr->modifiers &
                        activeFlag) ==
                        activeFlag );

    DoActivate( (WindowPtr)eventPtr->message,
                becomingActive );

    break;
}

```

`DoUpdate()` starts by retrieving a `windowPtr` from the event structure. As you can see from the code, the Event Manager stores a pointer to the affected window in the EventRecord's message field.

```

void DoUpdate( EventRecord *eventPtr )
{
    short        pictureID;
    PicHandle     picture;
    WindowPtr     window;

    window = (WindowPtr)eventPtr->message;

```

Then, `BeginUpdate()` tells the Event Manager that you're about to take care of the condition that caused the update.

```
BeginUpdate( window );
```

Now it's time to use that reference constant that we stored safely away in `InitWindows()`. `GetWRefCon()` is used to retrieve the `refCon`.

```
pictureID = GetWRefCon ( window );
```

Call `GetPicture()` to load the PICT resource. If the PICT could not be loaded, beep once, then exit.

```
picture = GetPicture( pictureID );

if ( picture == nil )
{
    SysBeep( 10 );    /* Couldn't load the PICT
                      resource!!! */
    ExitToShell();
}
```

If the PICT loaded OK, pass it to the `DoPicture()` routine to display it.

```
DoPicture( window, picture );
```



If the picture doesn't show up in one or both of the windows, check your resource files. Most likely, one of the PICT resources is messed up, or the resource ID is wrong. It's also possible that the PICT may not load if there is not enough memory available. If you suspect this is the case, try a smaller picture.

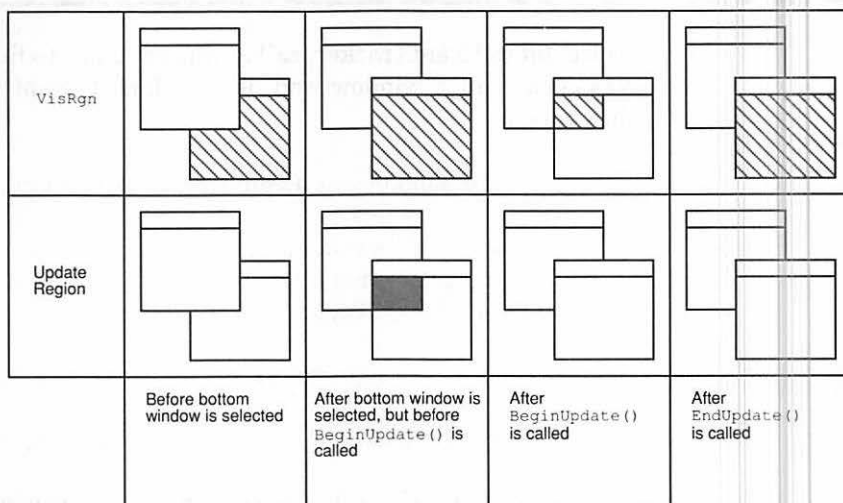
Finally, `EndUpdate()` tells the Mac to empty the window's update region.

```
EndUpdate( window );
}
```



Every window has an update region associated with it. When a previously covered section of a window is uncovered, the uncovered area is added to the window's update region. The Window Manager is constantly on the lookout for windows with non-empty update regions. When it finds one, it generates an `updateEvt` for that window. `BeginUpdate()`, as part of its processing, replaces the update region of the specified window with the empty region. Therefore, if you don't call `BeginUpdate()`, you'll never empty the window's update region, and the Window Manager will never stop generating `updateEvs` for the window.

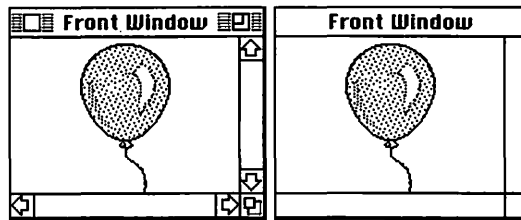
Figure 4.19 shows how these regions relate to each other: Before `BeginUpdate()` empties the update region, it replaces the visible region of the window (called the `visRgn`) with the intersection of the `visRgn` and the update region. The application then redraws the contents of the window. If it wants to, it can use this newly cropped `visRgn` to help reduce the amount of drawing necessary. For now, you'll just redraw the entire contents of the window. Finally, `EndUpdate()` is called. `EndUpdate()` replaces the original version of the `visRgn`. A call to `BeginUpdate()` without a corresponding call to `EndUpdate()` will leave your window in an unpredictable state. When your program is in this state, it should not drive or operate heavy machinery.



**Figure 4.19** `BeginUpdate()` in action.

`DoActivate()` is simple by comparison. `DrawGrowIcon()` redraws the grow box and the empty scroll bar areas. The grow box looks different depending on whether the window was activated or deactivated (Figure 4.20). `DrawGrowIcon()` is smart enough to draw the grow box correctly.

```
void      DoActivate( WindowPtr window, Boolean
                      becomingActive )
{
    DrawGrowIcon( window );
}
```



**Figure 4.20** The grow box, activated and deactivated.

## Handling mouseDown Events in Updater

As we did in `EventTracker`, call `FindWindow()` to find out if the mouse was clicked in a window and, if so, which part of which window the click was in.

```
void  HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr    window;
    short int     thePart;
    GrafPtr       oldPort;
    long          windSize;
    Rect          growRect;

    thePart = FindWindow( eventPtr->where, &window );
```

As before, `inSysWindow` means the mouse was clicked in a system window, very likely a desk accessory. (Because `Updater` doesn't support desk accessories, you probably won't see any `inSysWindow` mouseDowns, but you will see them in Chapter 5).



```
switch ( thePart )
{
    case inSysWindow :
        SystemClick( eventPtr, window );
        break;
```

The `inContent` part code represents the part of the window in which you draw. When you detect a mouse click inContent, call `SelectWindow()`. If the mouse click was not in the front-most window, `SelectWindow()` deactivates the front-most window and activates the clicked-in window. A call to `SelectWindow()` usually results in a pair of `activateEvs`.

```
case inContent:
    SelectWindow( window );
    break;
```

`inDrag` indicates a mouse click in window's drag region. Call `DragWindow()` with `screenBits.bounds`, which lets you drag the window anywhere you want.

```
case inDrag :
    DragWindow( window, eventPtr->where,
               &screenBits.bounds );
    break;
```

A click in the close box of either window will result in `gDone`'s being set to true. This will cause the program to exit. As we said earlier, this isn't the proper way to exit a Mac program, but we'll do a better job in Chapter 5.

```
case inGoAway :
    if ( TrackGoAway( window, eventPtr->where ) )
        gDone = true;
    break;
```

A click in the grow region is handled by a call to `GrowWindow()`. `GrowWindow()` allows the user to resize the specified window, limited by the fields in the `Rect growRect`. The top and left fields specify the minimum window height and width allowed. `GrowWindow()` won't allow the user to shrink the window beyond these values. bottom and right specify the maximum height and width. By setting these fields to `MAXSHORT`, you're allowing users to grow the window as big as possible, limited only by the size of their monitors.

```

case inGrow:
    growRect.top = kMinWindowHeight;
    growRect.left = kMinWindowWidth;
    growRect.bottom = MAXSHORT;
    growRect.right = MAXSHORT;

```

GrowWindow() returns a long integer composed of two words (2 bytes each) that define the number of pixels the window will grow or shrink in each direction. These words are passed to SizeWindow(), causing the window to be resized accordingly. The last parameter to SizeWindow() tells the Window Manager to accumulate any newly created content region into the update region. This means the Window Manager will generate an update event whenever the window is made either taller or wider.

```

windSize = GrowWindow( window,
                      eventPtr->where, &growRect );
if ( windSize != 0 )
{
    GetPort( &oldPort );
    SetPort( window );
    EraseRect( &window->portRect );
    SizeWindow( window, LoWord( windSize ),
               HiWord( windSize ), kNormalUpdates );
}

```

Next, we call InvalRect(), asking it to accumulate the entire content region into the update region. Here's why. When the window is resized, the picture will be redrawn, centered in the window. Because the picture is centered, its new position (in most cases) won't coincide with its old position. Our strategy? Erase the old picture and redraw it in its new position.

Between calls to BeginUpdate() and EndUpdate(), the Window Manager limits drawing to the *update region only*. Without the call to InvalRect(), we'd be able to draw the picture only in the newly expanded area of the window. To really understand this important point, comment out the call to InvalRect(), then recompile and run Updater. When the windows appear, grow the front window about 100 pixels in each direction. Experiment. When you're done, don't forget to uncomment the InvalRect() call.

```

    InvalRect( &window->portRect );
    SetPort( oldPort );
}
break;

```



Updater's update event strategy is fairly simple. Use the routine `InvalRect()` to add the entire contents of the window to the window's `updateRgn`, guaranteeing that an `updateEvt` will be generated whether or not the window was grown. When you plan your applications, spend some time working out an appropriate update strategy. If redrawing the contents of your windows will be fairly easy and won't take too long, you may want to use the `InvalRect()` approach. However, if the contents of your window are potentially complex (as is true of many drawing and CAD packages), you'll probably want to avoid the call to `InvalRect()` and, instead, use the shape of the update region to aid you in updating your window efficiently.

If the mouse is clicked in the zoom box, respond by calling `TrackBox()`. `TrackBox()` will return `true` if the mouse button is released while the mouse is still in the zoom box.

```
case inZoomIn:
case inZoomOut:
    if ( TrackBox( window, eventPtr->where,
                  thePart ) )
    {
```

Next, we save the old port and set the current port to window. Then, just as we did with `GrowWindow()`, we erase the contents of the window and call `ZoomWindow()`.

```
GetPort( &oldPort );
SetPort( window );
EraseRect( &window->portRect );
```

`ZoomWindow()` zooms the window in or out, depending on the part code passed as a parameter. The constant `kLeaveWhereItIs` tells `ZoomWindow()` to leave the window in front if it was in front when the zoom box was pressed, and in back if the window was in back when the zoom box was pressed. Just as you did with `SizeWindow()`, call `InvalRect()` to guarantee that an `updateEvt` is generated when the window is zoomed in or out. Finally, reset the old `GrafPort`.

```

ZoomWindow( window, thePart,
            kLeaveWhereItIs );
InvalRect( &window->portRect );
SetPort( oldPort );
    }
    break;
}
}

```

`DoPicture()` draws the window contents, clipping the drawing so that the scroll bar and grow areas aren't overwritten. Start by saving a pointer to the current `GrafPort` in `oldPort` so you can restore it when you're done drawing. Next, make window the current `GrafPort` so the picture will be drawn in the correct window:

```

void DoPicture( WindowPtr window, PicHandle picture )
{
    Rect        drawingClipRect, windowRect;
    GrafPtr     oldPort;
    RgnHandle    tempRgn;

    GetPort( &oldPort );
    SetPort( window );

```

Just as we squirreled away a copy of the current `GrafPort`, we'll also save a copy of the current clip region to restore later. To avoid drawing on top of our scroll bars, we'll set the clip region to the window's content region, then shrink the height and width to account for the two scroll bars.

First, call `NewRgn()` to allocate memory for a minimum-sized region. `GetClip()` resizes the region to accommodate the current clip region.

```

tempRgn = NewRgn();
GetClip( tempRgn );

```



If you created a region in the shape of a star and used `SetClip()` to set the clip region to your star region, all drawing in that window would be clipped in the shape of a star. You can read more about regions in *Inside Macintosh* (I:141–142 and I:166–167).

Next, erase the whole window with a call to `EraseRect()`. You've just erased the grow icon, so call `DrawGrowIcon()` to redraw it.

```
EraseRect( &window->portRect );

DrawGrowIcon( window );
```

Next, set up your clipping `Rect`, `drawingClipRect`, so that it excludes the right and bottom scroll bar areas (and, as a result, the grow area). Then, set `windowRect` to the `drawingWindow` `portRect`. You'll use `windowRect` as a parameter to `CenterPict()`, where it will be adjusted to reflect the size of the picture, centered in the input `Rect`.

```
drawingClipRect = window->portRect;
drawingClipRect.right -= kScrollbarAdjust;
drawingClipRect.bottom -= kScrollbarAdjust;

windowRect = window->portRect;
CenterPict( picture, &windowRect );
```

At this point, you have not changed the clip region of `drawingWindow`. To do this, call `ClipRect()` to set the clipping region to the rectangle defined by `drawingClipRect`. Now, draw the picture with `DrawPicture()`.

```
ClipRect( &drawingClipRect );
DrawPicture( picture, &windowRect );
```

Finally, reset the `ClipRect` to the region saved in `tempRgn`, release the memory allocated to `tempRgn`, and set the current `GrafPort` back to the original setting.

```
SetClip( tempRgn );
DisposeRgn( tempRgn );
SetPort( oldPort );
}
```

`CenterPict()` is the same as in Chapter 3's `ShowPICT` program:

```
void    CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (**( picture )).picFrame;
```

```
OffsetRect( &pictRect, windRect.left - pictRect.left,  
            windRect.top - pictRect.top);  
OffsetRect( &pictRect, (windRect.right -  
            pictRect.right)/2,  
            (windRect.bottom - pictRect.bottom)/2);  
*destRectPtr = pictRect;  
}
```

The first two programs of Chapter 4, *EventTracker* and *Updater*, demonstrated how you should handle the standard events generated by the Event Manager. The next program, **EventTrigger**, lets you send the required Apple events to *EventTracker*.

---

## EventTrigger: Sending Apple Events

---

In *EventTracker*, you learned how to install an Apple event handler, a routine designed to respond to a specific Apple event. When you launched *EventTracker*, the only Apple event you saw was the `kAEOpenApplication` sent by the Finder. *EventTrigger* was designed to add a little more life to *EventTracker*. Each time you run it, *EventTrigger* sends the four required Apple events to *EventTracker*. If you keep an eye on the *EventTracker* window, you'll see the event handlers execute, one at a time, drawing the event name in the event window.



*EventTrigger* and *EventTracker* demonstrate the basics of Apple event handling. For example, when *EventTracker* receives a `kAEQuitApplication` event, it doesn't quit. When it receives a `kAEOpenDocuments` Apple event, it doesn't open any documents. You'll learn a lot more about Apple events when you get to Chapter 8.

## Sending Apple Events

To send an Apple event, you have to tell the Apple Event Manager where you'd like the event sent (the **target address**) and what type of event you'd like sent. Call the routine `AECreatDesc()` to create a **descriptor record**, a complete description of the target of your Apple event. Then call `AECreatAppleEvent()` to create the Apple event itself. Finally, call `AESend()` to pass the event on to *EventTracker*. Let's look at each one of these routines in turn.

## AECreatDesc()

AECreatDesc() uses data describing the target application and creates a descriptor record used by other Apple Event Manager routines:

```
OSErr AECreatDesc( DescType typeCode, Ptr dataPtr,  
                  Size dataSize, AEDesc &result);
```

The first parameter describes the receiver of the Apple event. Since we are sending an Apple event to an application, we'll use the type typeApplSignature. This tells the Apple Event Manager we'd like to send the event to an application having a particular signature, or creator ID.

We'll pass the signature in the second parameter. When we built the standalone EventTracker application earlier in the chapter, we used **Set Project Type...** to set the **Creator** to 'Prmr'. To send an Apple event to EventTracker, we'll pass 'Prmr' in AECreatDesc()'s second parameter.

The third parameter is the number of bytes passed in the second parameter. The resulting descriptor record, the fourth parameter, is used by AECreatAppleEvent() in determining the target for the new Apple event.

## AECreatAppleEvent()

AECreatAppleEvent() is used to create the Apple event itself.

```
OSErr AECreatAppleEvent( AEEEventClass theAEEEventClass,  
                        AEEEventID theAEEEventID,  
                        AEAddressDesc target, short  
                        returnID, long transactionID,  
                        AppleEvent &result);
```

To create an Apple event, specify the class and ID of the desired Apple event. For example, if you wanted to send an Apple event to print a document, you would use the event class kCoreEventClass and an event ID of kAEPrintDocuments. The third parameter contains the target application information you got from AECreatDesc(). The fourth and fifth parameters are used to track the Apple event's progress toward the target application. The final parameter is the new Apple event.

Once you have created an Apple event, you can send it by way of AESend().

## AESend()

`AESend()` takes the Apple event you defined in `AECreatAppleEvent()` and sends it to the application targeted by `AECreatDesc()`.

```
OSErr AESend( AppleEvent theAppleEvent, AppleEvent &reply,  
              AESendMode sendMode, AESendPriority  
              sendPriority, long timeOutInTicks,  
              IdleProcPtr idleProc, EventFilterProcPtr  
              filterProc);
```

The first parameter is the Apple event you're sending. The second parameter is the Apple event reply that is received if a return reply has been requested. The third parameter, `sendMode`, allows you to request a return reply, or to specify the level of interaction you'd like to have with the target application. The fourth parameter is a flag that determines whether your request is put in the back of the event queue (`kAENormalPriority`) or at the front of the event queue (`kAEHighPriority`). The fifth parameter, `timeOutInTicks`, is the time in ticks (sixtieths of a second) that your application is willing to wait for a reply before giving up. The sixth parameter, `idleProc`, points to a procedure that should be called while you're waiting for a reply. The final parameter, `filterProc`, points to a procedure that can sort through Apple event return replies and decide which ones to process.



Apple events are monstrously cool! Once you understand `EventTracker` and `EventTrigger`, go through the sample code in Chapter 8. Once you've conquered these, go out there and beg, borrow, or steal a copy of *Inside Macintosh*, Volume VI, and read through Chapter 6.

Master Apple events—it's the right thing to do.

Now that you've looked at the routines that allow you to send Apple events, let's make it work in code.

---

## The EventTrigger Algorithm

---

`EventTrigger` is the last program in this chapter. It demonstrates how you can send Apple events from one application to another. `EventTrigger` consists of six steps:



- Initializes the Toolbox and tests for Apple events.
- Creates, describes, and sends the `kAEOpenApplication` Apple event.
- Creates, describes, and sends the `kAEOpenDocuments` Apple event.
- Creates, describes, and sends the `kAEPrintDocuments` Apple event.
- Creates, describes, and sends the `kAEQuitApplication` Apple event.
- Quits.

## Setting Up the EventTrigger Project

Start by creating a new project folder, called `EventTrigger`, inside your Development folder. `EventTrigger` has no user interface and needs no resources, so start up THINK C. When prompted, create a new project inside the `EventTrigger` folder. Call the project `EventTrigger.π`. Select **New** from the **File** menu to create a new source code file. Type the code listing in and save the file inside the `EventTrigger` folder as `EventTrigger.c`. Select **Add** (not **Add...**) from the **Project** menu to add `EventTrigger.c` to the project (don't forget MacTraps).

Here's the source code for `EventTrigger`:

```
#include <AppleEvents.h>
#include <GestaltEqu.h>

#define kGestaltMask 1L

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
void    EventsInit( void );
void    SendEvent( AEEventID theAEEventID );

/***** main *****/

void    main( void )
{
    ToolBoxInit();
    EventsInit();

    SendEvent( kAEOpenApplication );
    SendEvent( kAEOpenDocuments );
}
```

```

        SendEvent( kAEPrintDocuments );
        SendEvent( kAEQuitApplication );
    }

/***** ToolboxInit */

void    ToolboxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** EventsInit */

void    EventsInit( void )
{
    long    feature;
    OSErr    err;

    err = Gestalt( gestaltAppleEventsAttr, &feature );

    if ( err != noErr )
    {
        SysBeep( 10 );    /* Error calling Gestalt!!! */
        ExitToShell();
    }

    if ( !( feature & ( kGestaltMask <<
        gestaltAppleEventsPresent ) ) )
    {
        SysBeep( 10 );    /* AppleEvents not supported!!! */
        ExitToShell();
    }
}

```

```
/****** SendEvent *****/

void      SendEvent( AEEEventID theAEEEventID )
{
    OSErr          err;
    AEAddressDesc  address;
    OSType          appSig;
    AppleEvent      appleEvent, reply;

    appSig = 'Prmr';

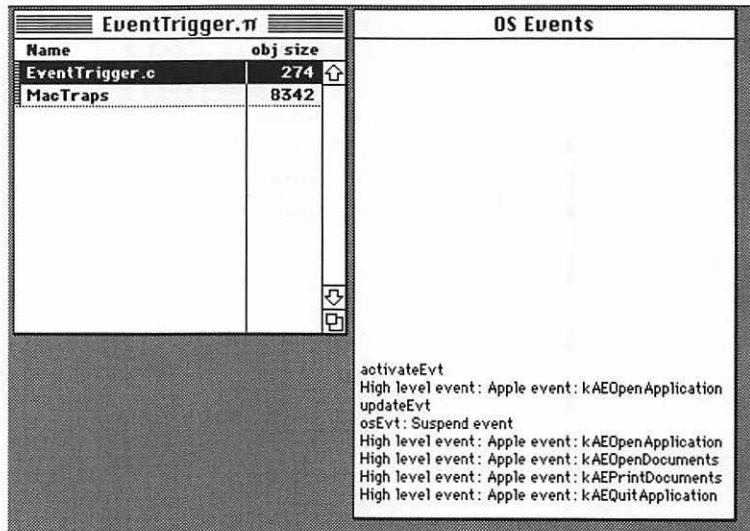
    err = AECreateDesc( typeApplSignature, (Ptr)(&appSig),
                        (Size)sizeof( appSig ), &address );

    err = AECreateAppleEvent( kCoreEventClass,
                              theAEEEventID, &address,
                              kAutoGenerateReturnID, 1L,
                              &appleEvent );

    err = AESend( &appleEvent, &reply, kAENoReply +
                  kAECanInteract, kAENormalPriority,
                  kAEDefaultTimeout, nil, nil );
}
```

## Running EventTrigger

Now that your source code is entered, you're ready to run EventTrigger. First, however, double-click on the EventTracker standalone application you built at the beginning of this chapter. When both the EventTracker OS window and the EventTrigger project window are displayed—but not overlapping—select **Run** from the **Project** menu. THINK C will start the compilation process. Figure 4.21 shows the result. If you like, run EventTrigger several times. You won't hurt anything. Each time you run it, four new event strings should appear in the EventTracker window.



**Figure 4.21** EventTrigger sends Apple events to EventTracker.



Some things to look for if EventTracker isn't getting its messages. Check your code again closely. Each parameter in the Apple event functions must be just right for this to work. If they look right and EventTrigger still doesn't work, try recompiling EventTracker, and make sure the **Creator** in the **Set Project Type...** dialog is set to 'Prmr'. Of course, if EventTracker is not actually running, EventTrigger will never succeed. Apple events can be sent only to applications that are already running.

If you don't have enough memory to run THINK C, your project, and EventTracker at the same time, build EventTrigger as a standalone application first. Then quit THINK C, launch EventTracker, and then EventTrigger.

Now, let's take a look at the code.

## Walking Through the EventTrigger Code

EventTrigger starts with the `#includes` for Apple events and `Gestalt()`, as was done in EventTracker:

```
#include <AppleEvents.h>
#include <GestaltEqu.h>

#define kGestaltMask 1L
```

**EventTrigger** has only three functions:

```
void    ToolBoxInit( void );
void    EventsInit( void );
void    SendEvent( AEEventID theAEEventID );
```

**main()** calls **ToolBoxInit()**, as always. Then it checks to see if your Mac supports Apple events by calling **EventsInit()**. If Apple events are supported, call **SendEvent()**, once for each of the four required Apple events.

```
/****** main *****/

void    main( void )
{
    ToolBoxInit();
    EventsInit();

    SendEvent( kAEOpenApplication );
    SendEvent( kAEOpenDocuments );
    SendEvent( kAEPrintDocuments );
    SendEvent( kAEQuitApplication );
}
```

**Here's a familiar face.**

```
/****** ToolBoxInit */

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}
```

`EventsInit()` calls `Gestalt()`, as in `EventTracker`, to make sure that both `Gestalt()` and Apple events are up and running on your Mac. If there is a problem, instead of displaying the error string as in `EventTracker`, you just beep and quit.

```

/***** EventsInit */

void    EventsInit( void )
{
    long    feature;
    OSErr    err;

    err = Gestalt( gestaltAppleEventsAttr, &feature );

    if ( err != noErr )
    {
        SysBeep( 10 ); /* Error calling Gestalt!!! */
        ExitToShell();
    }

    if ( !( feature & ( kGestaltMask <<
                       gestaltAppleEventsPresent ) ) )
    {
        SysBeep( 10 ); /* Apple events not
                       supported!!! */
        ExitToShell();
    }
}

```

`SendEvents()` calls `AECreatDesc()`, `AECreatAppleEvent()`, and `AESend()` to send the Apple event with the ID specified in `theAEEEventID`. The event is sent to the application with the creator ID 'Prmr'.

```

void    SendEvent( AEEEventID theAEEEventID )
{
    OSErr        err;
    AEAddressDesc address;
    OSType        appSig;
    AppleEvent    appleEvent, reply;

    appSig = 'Prmr';

    err = AECreatDesc( typeAppSignature, (Ptr)(&appSig),
                       (Size)sizeof( appSig ), &address );

```

```
err = AECreatAppleEvent( kCoreEventClass,
                        theAEEEventID, &address,
                        kAutoGenerateReturnID, 1L, &appleEvent );

err = AESend( &appleEvent, &reply, kAENoReply +
             kAECanInteract, kAENormalPriority,
             kAEDefaultTimeout, nil, nil );
}
```



The definitive reference on Apple events is the *Apple Event Registry*, available from APDA. See Chapter 9 for information about APDA.

---

## In Review

---

At the heart of every Macintosh application is the main event loop. Mac applications are built around this loop. Each pass through the main event loop consists of the retrieval of an event from the event queue and the processing of the event.

Apple events are events that are sent and received between applications. Macintosh applications are expected to respond to the four required Apple events: `kAEOpenApplication`, `kAEOpenDocuments`, `kAEPrintDocuments`, and `kAEQuitApplication`. In Chapter 8, you'll learn about the required Apple events in more detail.

The Window Manager plays an important role in the handling of events by generating `updateEvts` as a means of getting the application to draw (or update) the contents of a window. In addition, Window Manager routines, such as `FindWindow()`, offer a mechanism for linking an event to a window.

In Chapter 5, you'll learn all about menus. You'll learn how to design and implement regular menus, hierarchical menus, and pop-up menus.

---

---

# Menu Management

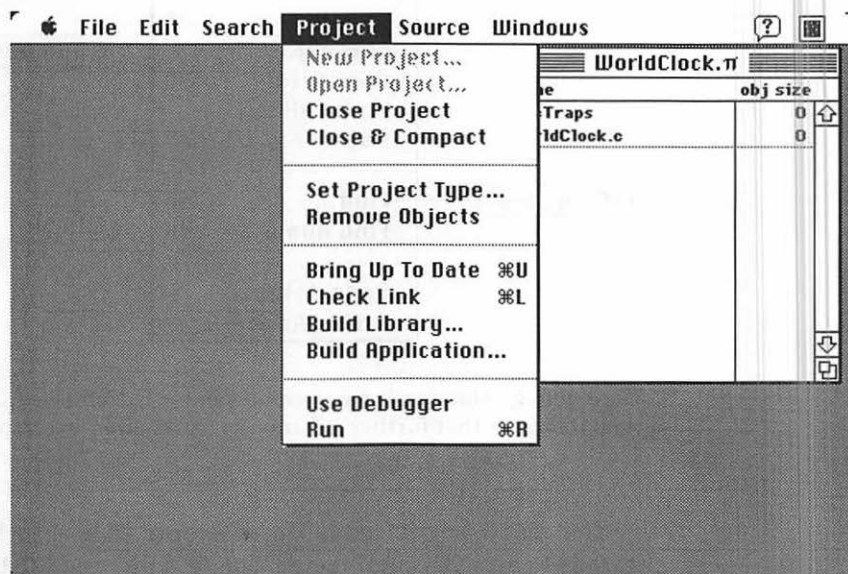
*This chapter explains the techniques you'll need to know to add menus to your programs. We'll show you how to create MBAR and MENU resources that specify your program's menus, then use those resources to bring the menus to life.*

---

---



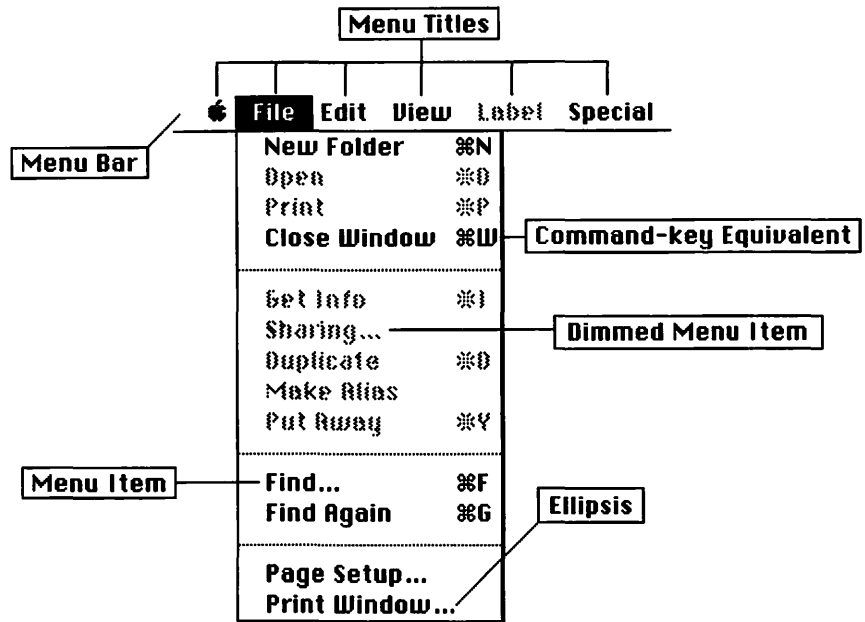
THE EARLIEST MACINTOSH incarnations offered a single choice when it came to menus: the classic, **pull-down menu**—the strip at the top of the screen with options that, when clicked on, displayed the possibilities available to each program (Figure 5.1). When Apple rolled out the Macintosh SE and the Mac II, the situation changed for the better with two additional menu types: the **hierarchical menu** and the **pop-up menu**. We'll discuss and illustrate both. But first, let's look at the standard parts of all menu systems.




**Figure 5.1** The pull-down menu.

## Menu Components

Before we get into the details of menu implementation, let's discuss the parts that make up a menu. Figure 5.2 shows the components of the Macintosh pull-down menuing system. The menu bar displayed at the top of the Mac screen is normally 20 pixels high. All menu text is drawn using the system font and size (usually 12-point **Chicago**).



**Figure 5.2** Macintosh menu components. The ellipsis (...) following a menu item indicates that further information is required to complete the command.

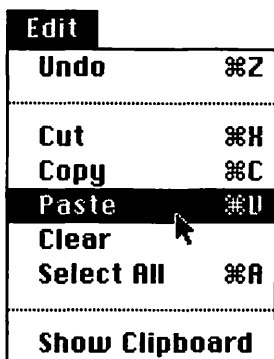
The menu bar is made up of **menu titles**, each of which corresponds to an individual menu. The , **File**, and **Edit** menus are found in most Macintosh applications. A menu's title is dimmed, or disabled, when none of its options is available.

**Menu items** are the choices available in a given menu. When the mouse is clicked in a menu's title in the menu bar, the menu title is inverted and the menu itself appears below the inverted title. The menu lists each of the menu's items.

Take a good look at Figure 5.2. In this figure, the mouse was clicked in the **File** menu title, causing the **File** menu to appear. The characters ⌘N to the right of the **New Folder** item are together known as a **Command-key Equivalent**.





If the mouse is dragged over a menu item, the item is highlighted. In Figure 5.3, the **Paste** item in the **Edit** menu is highlighted. If the mouse button is released while an item is highlighted, that item is **selected**, and the application takes the action associated with that item.

Like menu titles, individual items may also be disabled (dimmed). An icon or a check mark can be placed to the left of an item's text. The font and size of the item may be varied; Command-key equivalents

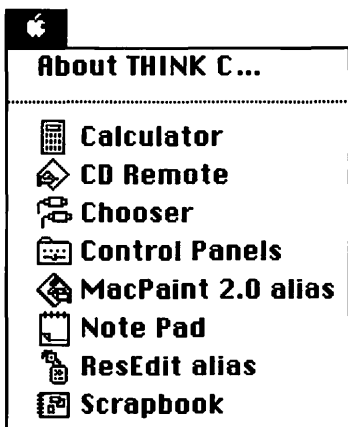


**Figure 5.3** The Finder's **Edit** menu with the **Paste** item selected.

may be placed to the right of a menu item. If a menu item list becomes too long for the screen, which is not uncommon on the smaller Macs, the last item that would normally be seen is replaced with a downward-pointing arrow (▼). If the user drags the mouse cursor down farther, more menu items will scroll into view.

The  menu is different in several respects from the other menus in the menu bar. By convention, the first item in the  menu is used by your application to display information (an **about box**) describing itself. The remaining menu items list the files found inside the Apple Menu Items folder inside your System Folder (Figure 5.4). For the most part, you'll only need to worry about the first item in the  menu (the **About** item). With some help from your application, System 7 will handle the rest of the  menu for you.

Let's take a look at some of the other available menu types.

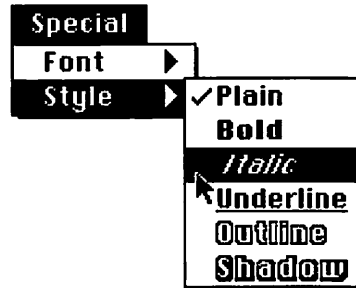


**Figure 5.4** The  menu.

## The Hierarchical Menu

The **hierarchical menu** was added to the Toolbox in 1987. It was needed for the new, complex programs that had become available for the Mac. As more bells and whistles were added to Mac applications, it became harder to find a place for them on the menu bar. Hierarchical menus allow a programmer to attach an entire menu to a single menu item. For example, Figure 5.5 shows a regular pull-down menu sporting two hierarchical menu items. When a hierarchical menu item is highlighted, its associated submenu appears to its right. You can then select an item from the submenu.

Hierarchical menu items always have a small, right-pointing triangle (▶) on their right side.



**Figure 5.5** The menu of styles appears when the **Style** hierarchical menu item is selected.

## The Pop-up Menu

The **pop-up menu** is the only menu that can be placed anywhere on the screen. This menu is similar to a pull-down menu, except that pop-up menus can be placed inside windows, dialog boxes, even on the desktop.

A pop-up menu appears when a `mouseDown` occurs in an area defined by an application. This area is known as the **pop-up rectangle**. Once the pop-up menu appears, the user can select an item just as they would from a pull-down menu. When the mouse button is released, the selection is processed.

The pop-up menu in Figure 5.6 appears in THINK C's **Add...** dialog box. The down-pointing triangle (on the right), the menu text, and the



**Figure 5.6** This pop-up menu label is from THINK C's **Add...** dialog box.

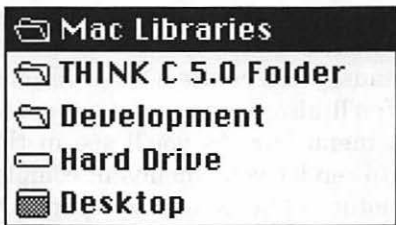
1-pixel drop shadow (below and to the right) are important pop-up menu elements. This pop-up is in its **inactive state**, the state it's in before the user clicks on it.



The icon representing the open folder on the left side of the pop-up menu in Figure 5.6 shows that you can add any graphic elements you like to your pop-up. The down-pointing triangle, the pop-up menu text, and the drop shadow are drawn for you by the Toolbox. Any other elements are your responsibility.

Frequently, the text in an inactive pop-up incorporates the last item selected from the pop-up. For example, Figure 5.7 shows the pop-up menu from Figure 5.6 in its **active state**. Notice that the pop-up text in Figure 5.6 corresponds to the item highlighted in Figure 5.7.

A more traditional pop-up menu is demonstrated in this chapter's WorldClock program. WorldClock displays the current time in a window and refreshes the time once every second. The pop-up menu shown in Figure 5.8 allows you to select a time zone. Selecting **Moscow**, for example, tells WorldClock to display the current time in Moscow.

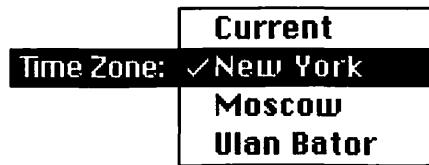


**Figure 5.7** The same pop-up after the mouse button was clicked.

Time Zone: **New York ▼**

**Figure 5.8** WorldClock's pop-up menu.

Notice that the pop-up in Figure 5.8 contains a down-pointing triangle, a 1-pixel drop shadow, and text showing the last selection made from the pop-up (**New York**). In addition, this pop-up has a **title**, text that does not appear in the pop-up menu itself. Typically, a pop-up's title ends in a colon (:). Figure 5.9 shows this pop-up menu in its active state.



**Figure 5.9** Making a selection from WorldClock's pop-up menu.

By the end of this chapter, you'll know how to add pull-down, hierarchical, and pop-up menus to your programs.

---

## Adding Menus to Your Programs

---

The Toolbox's Menu Manager provides everything you'll need to add menus to your programs. Although you can create all your menus from scratch, the usual approach is to base your menus on resources, just as you did for windows.

For pull-down menus, you'll create a MENU resource for each of your program's menus. You'll also need an MBAR resource to link all your MENUS into a single menu bar. As you'll see in the following pages, ResEdit allows you to render your menus in complete detail. You can link one MENU to another MENU's item to create a hierarchical menu. You can customize your menus, adding color, styled text, icons, and special symbols (such as a check mark) to individual items or, in the case of color, to the entire menu itself.

Pop-up menus require just a bit more work. In addition to the MENU resource that defines the pop-up's menu, you'll need to create a CNTL resource that defines the pop-up's bounding rectangle and outside




label text. As you'll see in the next chapter, the CNTL resource plays a big role in the Toolbox. In addition to pop-up menus, the CNTL resource is used to define scroll bars, push buttons, radio buttons, and more. For the moment, we'll stick with the pop-up CNTL.

---


## WorldClock

---

The best way to learn about the Menu Manager is to see it in action. With that in mind, we present WorldClock, a program that combines pull-down, hierarchical, and pop-up menus. As mentioned earlier, WorldClock displays the current time in a window and refreshes the time once every second.

Just as every Macintosh application should, WorldClock supports the standard , **File**, and **Edit** menus. The  menu is fully functional, giving you access to all the items you normally expect to see in the  menu. The **File** menu contains a single item, **Quit**.

WorldClock's **Edit** menu contains the minimum standard set of **Edit** commands (Figure 5.10). While you may want to add commands to the bottom of the **Edit** menu, you'll always want to support the items listed in Figure 5.10. The uniformity of the **Edit** menu across Mac applications is one of the many reasons Macintosh applications are so easy to use.

In addition to the standard , **File**, and **Edit** menus, WorldClock supports a **Special** menu. The **Special** menu has two hierarchical submenus, which allow you to change the clock's font and style.

Finally, WorldClock uses a pop-up menu that allows you to specify the current time zone.

Edit	
Undo	⌘Z
<hr/>	
Cut	⌘H
Copy	⌘C
Paste	⌘V
Clear	

**Figure 5.10** WorldClock's **Edit** menu.

## The WorldClock Algorithm

WorldClock uses the same basic event-loop structure as the programs presented in Chapter 4. Here's how WorldClock works:

- Initializes the WorldClock window;
- Creates the menu bar by loading the MBAR and MENU resources;
- Enters the event loop, refreshing the clock window once every second.

## Resources

Create a folder called WorldClock inside your Development folder. Next, launch ResEdit and, when the Jack-in-the-box appears, click the mouse to bring up the **Open File** dialog box. Click on the **New** button. When the **New File** dialog box appears, navigate into the WorldClock folder and create a resource file named WorldClock.π.rsrc in the WorldClock folder.

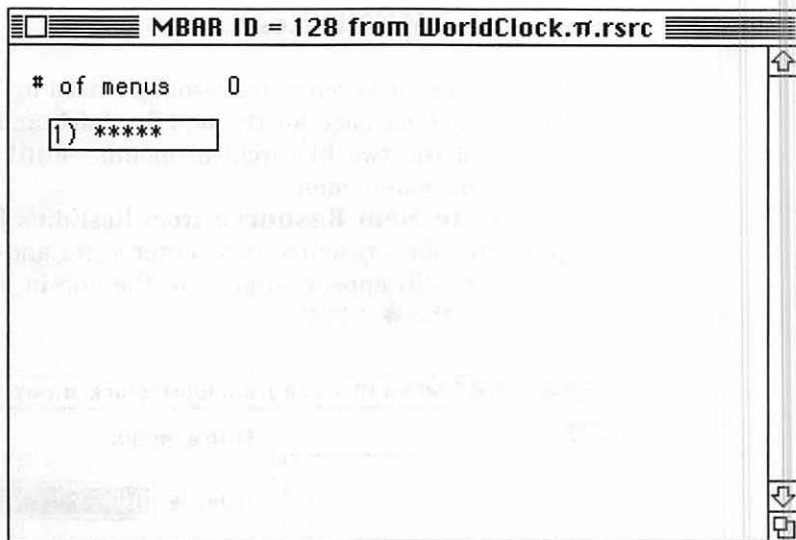
## Creating the MBAR Resource

Select **Create New Resource** from ResEdit's **Resource** menu. When prompted for a resource type, type MBAR, then click **OK** (be sure to spell MBAR exactly as it appears, in all upper-case letters). An MBAR editing window will appear. We'll need to add four menus to this menu bar, one for each menu title that appears in the menu bar itself: **⌘**, **File**, **Edit**, and **Special**. Notice that the two hierarchical menus (**Font** and **Style**) and the pop-up menu are not part of the MBAR resource. We'll deal with those later.

Click on the row of asterisks (\*) that appears in the window. A rectangle will appear around them to show they are selected (Figure 5.11). Select **Insert New Field(s)** from the **Resource** menu. A field labeled **Menu res ID** will appear underneath the first \* row, and a second \* row will appear beneath the new field. Click inside the **Menu res ID** field and type 128. This represents the resource ID of the first MENU in the menu bar. We'll create the MENU resources after we finish with the MBAR.

Next, click on the second \* row and again select **Insert New Field(s)** from the **Resource** menu. A second field and a third \* row will appear. Type 129 in the second field. Click on the third \* row and select **Insert New Field(s)** from the **Resource** menu. A third field and a fourth \* row will appear. Type 130 in the third field. OK, one more time. Click on the fourth \* row and select **Insert New Field(s)** from the **Resource** menu. A fourth field and a fifth \* row will appear.

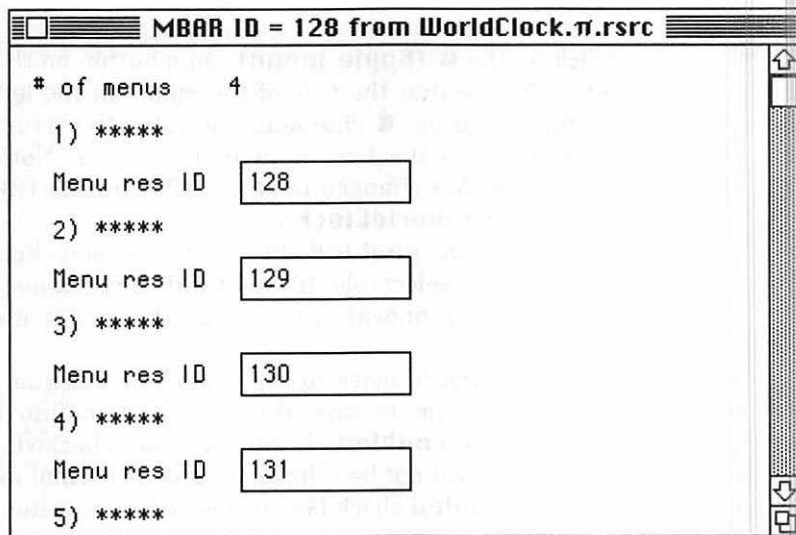




**Figure 5.11** An MBAR editing window.

Type 131 in the fourth field. Your MBAR resource should look like the one in Figure 5.12.

Click on the close box of the MBAR editing window. Next click on the close box of the MBAR listing window. The only window left open should be the window labeled `WorldClock.π.rsrc`.

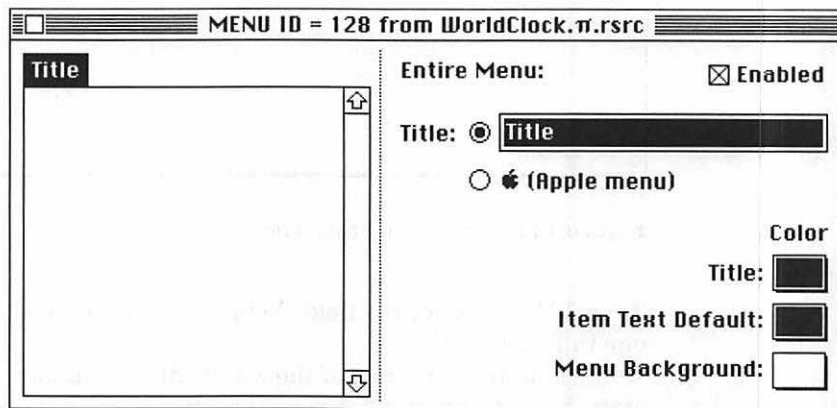


**Figure 5.12** The completed MBAR resource.

## Creating the MENU Resources

Next, we'll create the seven **MENU** resources used by WorldClock. Why seven? We'll need one each for the **Apple**, **File**, **Edit**, and **Special** menus, one for each of the two hierarchical menus—**Font** and **Style**—and one more for the pop-up menu.

Select **Create New Resource** from ResEdit's **Resource** menu. When prompted for a resource type, enter **MENU** and click **OK**. A **MENU** editing window will appear, similar to the one in Figure 5.13. We'll start by editing the **Apple** MENU.

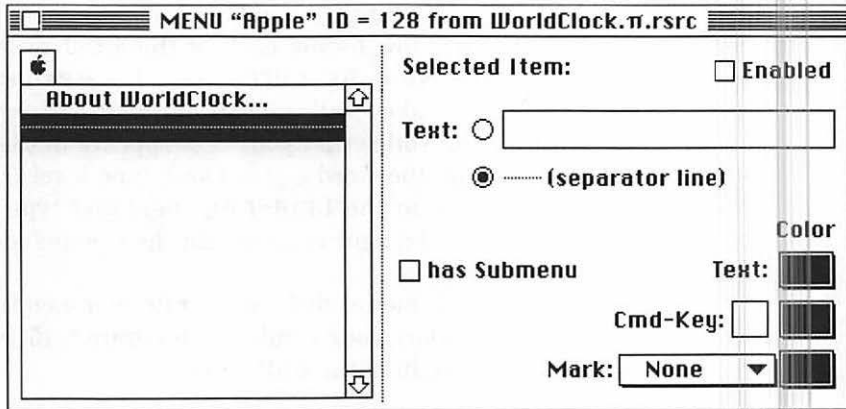


**Figure 5.13** A new **MENU** editing window.

Click on the **Apple (Apple menu)** radio button on the right side of the window. Notice that the title of the menu on the left side of the window changed to the **Apple** character. Now, hit the return key. The editor will move on to the first item in this **MENU**. Notice that the field labeled **Title:** has changed to **Text:**. Click in the **Text:** field and type the text **About WorldClock....**

Notice that the **Enabled** check box has been checked for you. This makes the item selectable. If the **Enabled** check box was not checked, the item would appear dimmed in the menu and would not be selectable.

Type a return to move to the next item. Click on the **(separator line)** radio button to turn this second item into a separator line. Notice that the **Enabled** check box is not checked. This means that the separator will not be selectable. This is normal for separator lines. Leave the **Enabled** check box unchecked for this item.



**Figure 5.14** The completed MENU.

At this point, your MENU should look like the MENU in Figure 5.14. To try out your new MENU, click on the on the right side of the menu bar. The menu that appears should have two items in it. The first item, **About WorldClock...**, should be selectable. The second item, the separator line, should not be selectable.

Next, select **Get Resource Info** from the **Resource** menu. Make sure the **ID:** field says 128. Close the Resource Info window. Now select **Edit Menu & MDEF ID...** from the **MENU** menu. Make sure the **Menu ID:** field says 128 and that the **MDEF ID:** field says 0, then click **OK**.



Unlike most resources, a menu's resource ID is part of the menu data structure itself. For example, a `WindowRecord` doesn't contain a `WIND` resource ID. The `MenuInfo` data structure uses the MENU's ID to link an application's menus together. That's why a MENU's resource ID appears in two different places and can be set to two different values—once in the Get Resource Info window and once in the **Edit Menu & MDEF ID...** dialog box.

*Always make these two values agree.*

Close the MENU editing window. You should see the MENU picker window (the window listing each of the MENU resources in this file). Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Without clicking the mouse, type the word **File**. Notice that your text appears in the **Title:** field. Hit a return and type the word **Quit**. Don't type a return after the **Quit**. Click the mouse in the **Cmd-Key:** field and type the letter **Q**. This tells the Menu Manager to associate the Command-key sequence **⌘Q** with this item.

To try out this menu, click on the **File** menu on the *right* side of the menu bar. Compare your results with Figure 5.15. Notice the **⌘Q** that appears to the right of the **Quit** item.



**Figure 5.15** Testing the **File** menu in ResEdit.

Finally, ensure that the resource ID for this MENU is set to 129. Make sure you check this by way of both the **Get Resource Info** and the **Edit Menu & MDEF ID...** menu items.

Close the MENU editing window. You should see the MENU picker window (this time it should show two MENUS). Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Enter **Edit** in the **Title:** field.

Hit a return and type the word **Undo**. Before returning, type the letter **Z** in the **Cmd-Key** field. Hit a return and click on the **(separator line)** radio button. The **Enabled** check box should *not* be checked for this item.

Hit a return and type the word **Cut**. Type the letter **H** in the **Cmd-Key** field. Hit a return and type the word **Copy**. Type the letter **C** in the **Cmd-Key** field. Hit a return and type the word **Paste**. Type the letter **U** in the **Cmd-Key** field. Hit one last return and type the word **Clear**. The command **Clear** doesn't have a Command-key equivalent.

To try out this menu, click on the **Edit** menu on the *right* side of the menu bar. Compare your results with Figure 5.16.

Make sure the resource ID for this MENU is set to 130. Check this by way of both the **Get Resource Info** and the **Edit Menu & MDEF ID...** menu items.

Close the MENU editing window. You should see the MENU picker window (this time it should show three MENUS). Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Enter **Special** in the **Title:** field.

Edit	
Undo	⌘Z
<hr/>	
Cut	⌘H
Copy	⌘C
Paste	⌘V
Clear	

**Figure 5.16** Testing the **Edit** menu in ResEdit.

Hit a return and type the word **Font**. Click on the **has Submenu** checkbox. Once the checkbox is checked, a new field, **ID:**, will appear. This field contains the resource ID of the **MENU** to appear as a submenu. Enter 100 in the **ID:** field. We'll create **MENU 100** in a minute.



Submenu IDs are limited to values between 0 and 255. We use 100 as the base resource ID for hierarchical menus. Accordingly, **MENU** IDs from 100 to 127 are reserved for hierarchical **MENUS**. If you need more than 28 submenus, or if you just don't like starting with 100, feel free to use your own numbering scheme. Just make sure you keep your submenu IDs between 0 and 255.

Once the **ID:** field is filled in, type a return and type the word **Style**. Once again, check the **has Submenu** checkbox. Enter 101 in the **ID:** field.

To try this menu, click on the **Special** menu on the *right* side of the menu bar. Compare your results with Figure 5.17. Notice the right-pointing triangles to the right of both the **Font** and **Style** items.



One reason the appropriate submenus don't appear when the **Font** and **Style** menus are selected in the sample **Special** menu is that we haven't created them yet. An even better reason is that ResEdit doesn't show the submenus in its sample menus. To see the submenus, you'll have to wait until WorldClock is complete. Don't worry, they'll be there.



**Figure 5.17** Testing the **Special** menu in ResEdit.

Make sure the resource ID for the **Special** MENU is set to 131. Check this by way of both the **Get Resource Info** and the **Edit Menu & MDEF ID...** menu items.

Close **Special**'s MENU editing window. You should see the MENU picker window (this time it should show four MENUS). Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Enter **Font** in the **Title:** field.

You won't add any items to the **Font** MENU at this point. One of the first things WorldClock will do is ask the Menu Manager to add a list of all available fonts to this menu. This will allow WorldClock to keep up with whatever fonts are installed on the machine it's being run on.

Click on the **Font** menu on the *right* side of the menu bar. Compare your results with Figure 5.18. Kind of boring, huh?

## Font

**Figure 5.18** Testing the **Font** menu in ResEdit. At this point, the **Font** menu has no items.

Make sure the resource ID for the **Font** MENU is set to 100. Select **Get Resource Info** from the **Resource** menu and **Edit Menu & MDEF ID...** from the **MENU** menu. You'll need to make the change in both places. Don't forget!

Close **Font**'s MENU editing window. You should see the MENU picker window (this time it should show five MENUS). Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Enter **Style** in the **Title:** field.

Hit a return and type the word **Plain**. Hit another return and type the word **Bold**. Select **Bold** from ResEdit's **Style** menu. Be sure to select **Bold** from ResEdit's **Style** menu and not from the fake **Style** menu ResEdit builds for you. The word **Bold** should appear **Bold**, and the **Style** menu's **Bold** item should have a check mark (✓) next to it.

Hit a carriage return and type the word **Italic**. Select **Italic** from ResEdit's **Style** menu. The word "Italic" should change to **Italic**. Hit

another return and type Underline. Select **Underline** from ResEdit's **Style** menu.

Hit return and type Outline. Select **Outline** from ResEdit's **Style** menu. One more to go. Hit return and type Shadow. Select **Shadow** from ResEdit's **Style** menu. Compare your **Style** menu with the one shown in Figure 5.19.

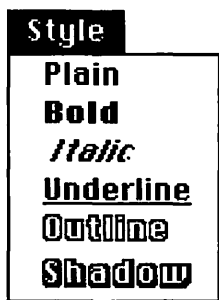


Figure 5.19 Testing the **Style** menu in ResEdit.

Make sure the resource ID for the **Style** MENU is set to 101. Select **Get Resource Info** from the **Resource** menu and **Edit Menu & MDEF ID...** from the **MENU** menu. Make the change in both places!

The last MENU we'll create is for the pop-up menu of time zones that will appear in the clock window. Close **Style**'s MENU editing window. You should see the MENU picker window (this time it should show six MENUS). Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Enter Time Zone in the **Title:** field.

Hit a carriage return and type the word Current. Hit a second return and type the text New York. Hit a third return and type Moscow. Hit a fourth return and type Ulan Bator. Compare your **Time Zone** menu with the one shown in Figure 5.20.



Figure 5.20 Testing the **Time Zone** menu in ResEdit.

Make sure the resource ID for the **Time Zone** pop-up MENU is set to 132. Select **Get Resource Info** from the **Resource** menu and **Edit Menu & MDEF ID...** from the **MENU** menu. Remember that the pop-up MENU uses resource ID 132. We'll need this information when we create our next resource.

## Creating a Pop-up Menu CNTL Resource

Prior to System 7, pop-up menus were somewhat difficult to implement. For starters, you had to draw the pop-up label, making sure the label reflected the current value of the pop-up. When a `mouseDown` occurred, you had to calculate whether said click occurred within the boundaries of the label. If so, you had to invert the label, then call a Toolbox function to bring the menu to life. Next, you had to reinvert, then update the label to reflect its new value. In a nutshell, pop-up menus were a lot of work.

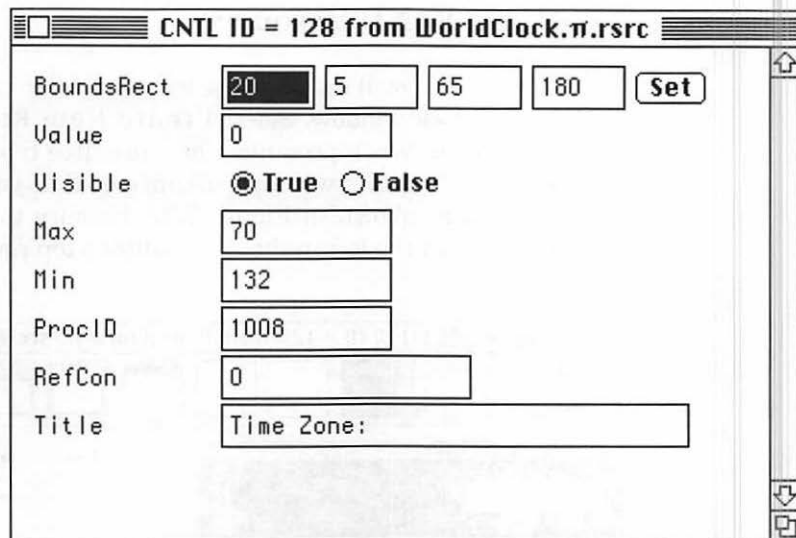
System 7 changed all that by working with the **Control Manager** to create a pop-up menu control. On the Mac, a control is a device that can change between a range of values. You've seen Macintosh controls before. Scroll bars, push buttons, check boxes, and radio buttons are all examples of controls. Push buttons, check boxes, and radio buttons are known as **simple controls**. They can take on one of two values: ON or OFF. A scroll bar can take on many values, depending on its initial settings. We'll get into the details of the Control Manager later in the book. For now, we just need to know enough to implement the new pop-up control.

Just as a WIND resource is used as a template to create a window, a CNTL resource is used to create a control. Back in ResEdit, close the MENU window and the MENU picker window, leaving only the window labeled `WorldClock.π.rsrc`. Select **Create New Resource** from the **Resource** menu. Specify the resource type CNTL and click **OK**. A CNTL editing window will appear. Fill in the CNTL fields exactly as specified in Figure 5.21.

The `BoundsRect` field specifies the top, left, bottom, and right sides of the pop-up menu rectangle. The Control Manager will draw the pop-up menu and title inside the rectangle. The `Value` field specifies how the title should appear in relation to the pop-up box itself. A `Value` of 0 indicates that the pop-up title will appear in the system font, drawn on the left side of the pop-up.

The `Visible` field determines whether the control is initially visible or hidden, just as it does in the WIND resource. The `Max` field specifies the portion of the `BoundsRect` to be allocated to the pop-up title. In this case, the left side of the pop-up box will appear 70 pixels





BoundsRect	20	5	65	180	Set
Value	0				
Visible	<input checked="" type="radio"/> True <input type="radio"/> False				
Max	70				
Min	132				
ProcID	1008				
RefCon	0				
Title	Time Zone:				

**Figure 5.21** Specifications for the pop-up CNTL.

from the left edge of the `BoundsRect`. The pop-up title, **Time Zone:**, will be drawn, left-justified, in those 70 pixels.

The `Min` field should look familiar to you. It contains the resource ID of the pop-up `MENU` resource. The `ProcID` serves the same purpose as in the `WIND` resource—it determines the type of control being specified. In this case, 1008 corresponds to the pop-up menu control.

The `RefCon` field can be used by your application as a 4-byte scratchpad area. For now, specify a value of 0. Finally, the `Title` field is used to draw the pop-up menu's title text. Remember to include the colon (:) at the end of your title.

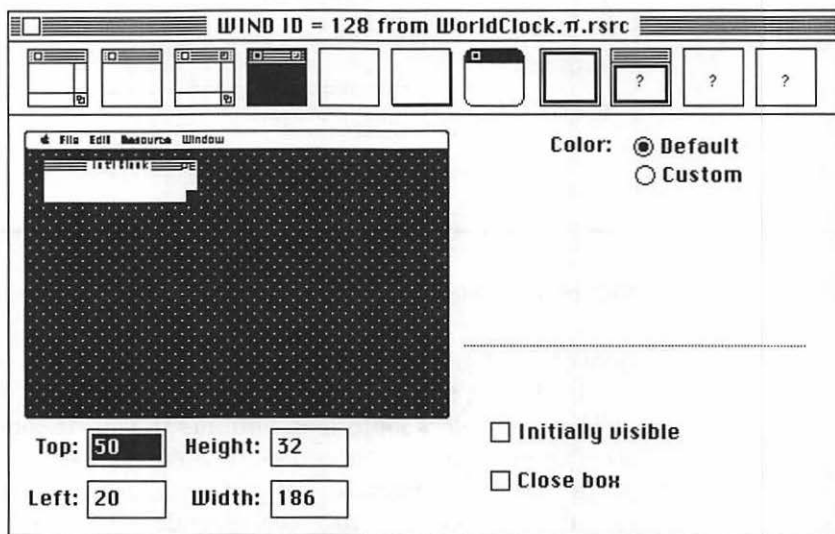
Finally, select **Get Resource Info** from the **Resource** menu and set the CNTL's resource ID to 128. When you're done, close the resource info window, the CNTL editing window, and the CNTL picker window, leaving only the window labeled `WorldClock.π.rsrc`.



For more information on the pop-up menu control, check out pages 3-16 through 3-19 of *Inside Macintosh*, Volume VI. These pages describe the pop-up control in detail, listing all of the important constants and data structures.

## Creating a WIND Resource

The last resource you'll create is the **WIND** resource used as a template to create the clock window. Select **Create New Resource** from the **Resource** menu. When prompted for a resource type, enter **WIND** and click **OK**. A **WIND** editing window will appear. Edit your **WIND** to match the specifications shown in Figure 5.22. Be sure to select the fourth window type from the left in the **WIND** editor's top row.



**Figure 5.22** WorldClock's **WIND** resource specifications.

Next, select **Set 'WIND' Characteristics...** from the **WIND** menu and set the **Window title:** field to **WorldClock**. Finally, select **Get Resource Info** from the **Resource** menu and set the **WIND**'s resource ID to 128.

That's it! Your resource file should consist of one **CNTL**, one **MBAR**, seven **MENUS**, and one **WIND**. Select **Quit** from the **File** menu and save your changes.

## Setting Up the Project

Launch **THINK C** and create a new project named **WorldClock.π** in the **WorldClock** folder, where you created your resource file. Add **MacTraps** to the project. Next, create a new source code file, save it as **WorldClock.c**, and add it to the project. Here's the source code for **WorldClock.c**:

```
#include <Packages.h>
#include <GestaltEqu.h>

#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L
#define kUseDefaultProc      (void *)-1L
#define kSleep               20L
#define kLeaveWhereItIs       false

#define kIncludeSeconds      true
#define kTicksPerSecond      60
#define kSecondsPerHour      3600L

#define kAddCheckMark        true
#define kRemoveCheckMark     false

#define kPopupControlID      kBaseResID
#define kNotANormalMenu      -1

#define mApple               kBaseResID
#define iAbout                1

#define mFile                 kBaseResID+1
#define iQuit                 1

#define mFont                 100

#define mStyle                101
#define iPlain                1
#define iBold                 2
#define iItalic               3
#define iUnderline            4
#define iOutline              5
#define iShadow               6

#define kPlainStyle           0

#define kExtraPopupPixels     25

#define kClockLeft            12
#define kClockTop             25
#define kClockSize            24

#define kCurrentTimeZone      1
#define kNewYorkTimeZone      2
```

```

#define kMoscowTimeZone      3
#define kUlanBatorTimeZone  4

#define TopLeft( r )         (*(Point *) &(r).top)
#define BottomRight( r )     (*(Point *) &(r).bottom)

#define IsHighBitSet( longNum )  ( (longNum >> 23) & 1 )
#define SetHighByte( longNum )   ( longNum |= 0xFF000000 )
#define ClearHighByte( longNum ) ( longNum &= 0x00FFFFFF )

/*****/
/*  Globals  */
/*****/

Boolean  gDone, gHasPopupControl;
short    gLastFont = 1, gCurrentZoneID = kCurrentTimeZone;
Style    gCurrentStyle = kPlainStyle;
Rect     gClockRect;

/*****/
/*  Functions  */
/*****/

void      ToolBoxInit( void );
void      WindowInit( void );
void      MenuBarInit( void );
void      EventLoop( void );
void      DoEvent( EventRecord *eventPtr );
void      HandleNull( EventRecord *eventPtr );
void      HandleMouseDown( EventRecord *eventPtr );
void      SetUpZoomPosition( WindowPtr window, short
                                zoomInOrOut );
void      HandleMenuChoice( long menuChoice );
void      HandleAppleChoice( short item );
void      HandleFileChoice( short item );
void      HandleFontChoice( short item );
void      HandleStyleChoice( short item );
void      DoUpdate( EventRecord *eventPtr );
long      GetZoneOffset( void );

```

```
/****** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();
    MenuBarInit();

    EventLoop();
}

/****** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/****** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 ); /* Couldn't load the WIND
                        resource!!! */

        ExitToShell();
    }

    SetPort( window );
    TextSize( kClockSize );
}
```

```

        gClockRect = window->portRect;

        ShowWindow( window );
    }

/***** MenuBarInit *****/

void      MenuBarInit( void )
{
    Handle          menuBar;
    MenuHandle      menu;
    ControlHandle   control;
    OSErr           myErr;
    long            feature;

    menuBar = GetNewMBar( kBaseResID );
    SetMenuBar( menuBar );

    menu = GetMHandle( mApple );
    AddResMenu( menu, 'DRVR' );

    menu = GetMenu( mFont );
    InsertMenu( menu, kNotANormalMenu );
    AddResMenu( menu, 'FONT' );

    menu = GetMenu( mStyle );
    InsertMenu( menu, kNotANormalMenu );
    CheckItem( menu, iPlain, true );

    DrawMenuBar();

    HandleFontChoice( gLastFont );

    myErr = Gestalt( gestaltPopupAttr, &feature );

    gHasPopupControl = ((myErr == noErr) && (feature &
        (1 << gestaltPopupPresent)));

    if ( gHasPopupControl )
        control = GetNewControl( kPopupControlID,
            FrontWindow() );
}

```

```

/***** EventLoop *****/

void    EventLoop( void )
{
    EventRecord  event;

    gDone = false;

    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, kSleep,
                           nil ) ) DoEvent( &event );
        else
            HandleNull( &event );
    }
}

/***** DoEvent *****/

void    DoEvent( EventRecord *eventPtr )
{
    char    theChar;

    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
        case keyDown:
        case autoKey:
            theChar = eventPtr->message & charCodeMask;
            if ( (eventPtr->modifiers & cmdKey) != 0 )
                HandleMenuChoice( MenuKey( theChar ) );
            break;
        case updateEvt:
            DoUpdate( eventPtr );
            break;
    }
}

```

```

/***** HandleNull *****/

void HandleNull( EventRecord *eventPtr )
{
    static long    lastTime = 0;

    if ( (eventPtr->when / kTicksPerSecond) > lastTime )
    {
        InvalRect( &gClockRect );
        lastTime = eventPtr->when / kTicksPerSecond;
    }
}

/***** HandleMouseDown *****/

void      HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr      whichWindow;
    GrafPtr        oldPort;
    short          thePart;
    long           menuChoice;
    ControlHandle   control;
    short          ignored;

    thePart = FindWindow( eventPtr->where, &whichWindow );
    switch ( thePart )
    {
        case inMenuBar:
            menuChoice = MenuSelect( eventPtr->where );
            HandleMenuChoice( menuChoice );
            break;
        case inSysWindow:
            SystemClick( eventPtr, whichWindow );
            break;
        case inContent:
            SetPort( whichWindow );
            GlobalToLocal( &eventPtr->where );

            if ( FindControl( eventPtr->where,
                             whichWindow, &control ) )
            {
                ignored = TrackControl( control,
                                       eventPtr->where,
                                       kUseDefaultProc );
            }
    }
}

```



```

        gCurrentZoneID = GetCtlValue( control );
    }
    break;
case inDrag:
    DragWindow( whichWindow, eventPtr->where,
                &screenBits.bounds );
    break;
case inZoomIn:
case inZoomOut:
    if ( TrackBox( whichWindow, eventPtr->where,
                    thePart ) )
    {
        SetUpZoomPosition( whichWindow, thePart );
        ZoomWindow( whichWindow, thePart,
                     kLeaveWhereItIs );
    }
    break;
}
}

```

/\*\*\*\*\*\* SetUpZoomPosition \*\*\*\*\*/

```

void      SetUpZoomPosition( WindowPtr window, short
                             zoomInOrOut )
{
    WindowPeek      wPeek;
    WStateData      *wStatePtr;
    Rect             windowRect;
    Boolean          isBig;
    short            deltaPixels;

    wPeek = (WindowPeek) window;
    wStatePtr = (WStateData *) *(wPeek->dataHandle);

    windowRect = window->portRect;
    LocalToGlobal( &TopLeft( windowRect ) );
    LocalToGlobal( &BottomRight( windowRect ) );

    wStatePtr->stdState = windowRect;
    wStatePtr->userState = wStatePtr->stdState;

    if ( gHasPopupControl )
    {

```

```

        isBig = (windowRect.bottom - windowRect.top) >
                (gClockRect.bottom - gClockRect.top);

        if ( isBig )
            deltaPixels = -kExtraPopupPixels;
        else
            deltaPixels = kExtraPopupPixels;

        if ( zoomInOrOut == inZoomIn )
            wStatePtr->userState.bottom += deltaPixels;
        else
            wStatePtr->stdState.bottom += deltaPixels;
    }
    else
        SysBeep( 20 );
}

/***** HandleMenuChoice *****/

void    HandleMenuChoice( long menuChoice )
{
    short    menu;
    short    item;

    if ( menuChoice != 0 )
    {
        menu = HiWord( menuChoice );
        item = LoWord( menuChoice );

        switch ( menu )
        {
            case mApple:
                HandleAppleChoice( item );
                break;
            case mFile:
                HandleFileChoice( item );
                break;
            case mFont:
                HandleFontChoice( item );
                break;
            case mStyle:
                HandleStyleChoice( item );
                break;
        }
    }
}

```

```
        HiliteMenu( 0 );
    }
}

/***** HandleAppleChoice *****/

void    HandleAppleChoice( short item )
{
    MenuHandle    appleMenu;
    Str255        accName;
    short         accNumber;

    switch ( item )
    {
        case iAbout: /* We'll put up an about box next
                        chapter.*/
            SysBeep( 20 );
            break;
        default:
            appleMenu = GetMHandle( mApple );
            GetItem( appleMenu, item, accName );
            accNumber = OpenDeskAcc( accName );
            break;
    }
}

/***** HandleFileChoice *****/

void    HandleFileChoice( short item )
{
    switch ( item )
    {
        case iQuit :
            gDone = true;
            break;
    }
}

/***** HandleFontChoice *****/

void    HandleFontChoice( short item )
{

```

```

        short          fontNumber;
        Str255          fontName;
        MenuHandle      menuHandle;

        menuHandle = GetMHandle( mFont );

        CheckItem( menuHandle, gLastFont, kRemoveCheckMark );
        CheckItem( menuHandle, item, kAddCheckMark );

        gLastFont = item;

        GetItem( menuHandle, item, fontName );
        GetFNum( fontName, &fontNumber );

        TextFont( fontNumber );
    }

/***** HandleStyleChoice *****/

void      HandleStyleChoice( short item )
{
    MenuHandle menuHandle;

    switch( item )
    {
        case iPlain:
            gCurrentStyle = kPlainStyle;
            break;
        case iBold:
            if ( gCurrentStyle & bold )
                gCurrentStyle -= bold;
            else
                gCurrentStyle |= bold;
            break;
        case iItalic:
            if ( gCurrentStyle & italic )
                gCurrentStyle -= italic;
            else
                gCurrentStyle |= italic;
            break;
        case iUnderline:
            if ( gCurrentStyle & underline )
                gCurrentStyle -= underline;
            else

```

```

        gCurrentStyle |= underline;
        break;
    case iOutline:
        if ( gCurrentStyle & outline )
            gCurrentStyle -= outline;
        else
            gCurrentStyle |= outline;
        break;
    case iShadow:
        if ( gCurrentStyle & shadow )
            gCurrentStyle -= shadow;
        else
            gCurrentStyle |= shadow;
        break;
}

menuHandle = GetMHandle( mStyle );

CheckItem( menuHandle, iPlain, gCurrentStyle ==
            kPlainStyle );
CheckItem( menuHandle, iBold, gCurrentStyle & bold );
CheckItem( menuHandle, iItalic, gCurrentStyle &
            italic );
CheckItem( menuHandle, iUnderline, gCurrentStyle &
            underline );
CheckItem( menuHandle, iOutline, gCurrentStyle &
            outline );
CheckItem( menuHandle, iShadow, gCurrentStyle &
            shadow );

TextFace( gCurrentStyle );
}

/***** DoUpdate *****/

void    DoUpdate( EventRecord *eventPtr )
{
    WindowPtr    window;
    Str255        timeString;
    unsigned long    curTimeInSecs;

    window = (WindowPtr)eventPtr->message;

```

```

        BeginUpdate( window );

        GetDateTime ( &curTimeInSecs );
        curTimeInSecs += GetZoneOffset();

        IUTimeString( (long)curTimeInSecs, kIncludeSeconds,
                      timeString );

        EraseRect( &gClockRect );
        MoveTo( kClockLeft, kClockTop );
        DrawString( timeString );

        DrawControls( window );

        EndUpdate( window );
    }

/***** GetZoneOffset *****/

long GetZoneOffset( void )
{
    MachineLocation  loc;
    long             delta, defaultZoneOffset;

    ReadLocation( &loc );
    defaultZoneOffset = ClearHighByte
        ( loc.gmtFlags.gmtDelta );

    if ( IsHighBitSet( defaultZoneOffset ) )
        SetHighByte( defaultZoneOffset );

    switch ( gCurrentZoneID )
    {
        case kCurrentTimeZone :
            delta = defaultZoneOffset;
            break;
        case kNewYorkTimeZone :
            delta = -5L * kSecondsPerHour ;
            break;
        case kMoscowTimeZone :
            delta = 3L * kSecondsPerHour;
            break;
        case kUlanBatorTimeZone :
            delta = 8L * kSecondsPerHour;

```

```
        break;
    }
    delta -= defaultZoneOffset;

    return delta;
}
```

## Running WorldClock

Now that your source code is in, you're ready to run WorldClock. Save your changes, then select **Run** from the **Project** menu. When asked to **Bring the project up to date**, click **Yes**. If everything went well, the WorldClock menus should appear in the menu bar and the WorldClock window should appear on the desktop. The current time should appear in the WorldClock window (Figure 5.23), updating once every second.



Figure 5.23 WorldClock in action.

Four menus should appear in the menu bar: **Apple**, **File**, **Edit**, and **Special**. The first item under the **Apple** menu should read **About WorldClock...**. When you select this item, you should hear a beep (or whatever your system is using for a beep sound). Try selecting another item from the **Apple** menu. Desk accessories and, under System 7, other **Apple** menu items should behave normally.

Next, pull down the **Special** menu. Two hierarchical menu items, **Font** and **Style**, should appear. When you highlight the **Font** item, a submenu listing all of the currently available fonts should appear. The first font should have a check mark (✓) next to it. Select a different font from this menu. The font in the clock window should change to match the selected font. Also, if you pull down the **Font** submenu again, the selected font should now have a check mark next to it.

Pull down the **Style** submenu. A check mark should appear next to the **Plain** item. Select **Bold**, then select **Outline**. The text in the clock window should appear bold and outlined. Also, check marks should appear next to the **Bold** and **Outline** items in the **Style** submenu. Select **Bold** again. The check mark next to it should disappear and the clock's style should change accordingly. Select **Outline** again. The check mark next to **Outline** should disappear

and, since there are no other styles selected, a check mark should appear next to the **Plain** style.



If things didn't go as planned, double-check each of the WorldClock resources, as well as the source code in `WorldClock.c`. Make sure your project and resource files are named correctly.

Now, let's test out the pop-up menu. Click on the zoom box in the upper right corner of the clock window. The window should grow longer, revealing a pop-up menu in the bottom half. The pop-up menu should say **Current**, as it does in Figure 5.24.



**Figure 5.24** The WorldClock window with a pop-up menu.



If the window beeps at you and refuses to change size, WorldClock could not find the pop-up menu control. Are you running System 7? If not, go out and get yourself a copy. While not everyone in the world will be running System 7, as a developer, you owe it to yourself to always run the most recent version of the operating system.

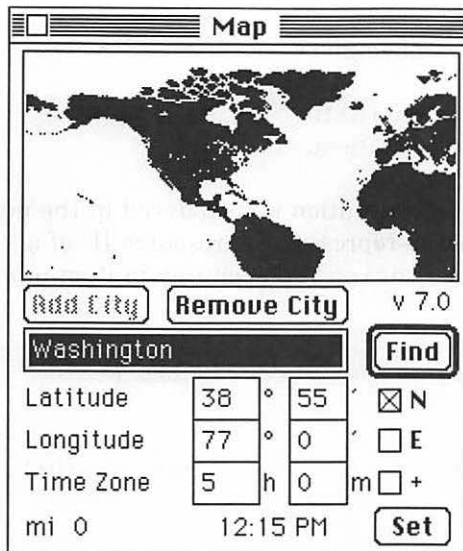
Click on the pop-up menu. Notice the check mark next to **Current**. Select **New York**. The time in the clock window tells you what time it is in New York.



Does the time difference make sense to you? For example, if you are in the United States, on the East Coast, the **Current** time should be the same as **New York** time. If not, your Macintosh may be set to the wrong time zone. To fix this, run the Map Control Panel that comes with the Macintosh system software (Figure 5.25). Click on the map (drag left or right if you have to) on your current location and click the **Set** button.

Finally, type **⌘Q** to quit WorldClock.





**Figure 5.25** The Map control panel.

## Walking Through the WorldClock Code

WorldClock starts with a pair of `#includes`. The file `Packages.h` contains the declarations you'll need to call `IUTimeString()`. `GestaltEqu.h` is necessary to call `Gestalt()`.

```
#include <Packages.h>
#include <GestaltEqu.h>
```

Several of these `#defines` should be familiar to you. Those that aren't should become clear from the sections of code in which they are used.

```
#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L
#define kUseDefaultProc      (void *)-1L
#define kSleep               20L
#define kLeaveWhereItIs       false

#define kIncludeSeconds      true
#define kTicksPerSecond      60
#define kSecondsPerHour      3600L
```

```
#define kAddCheckMark      true
#define kRemoveCheckMark  false

#define kPopupControlID    kBaseResID
#define kNotANormalMenu    -1
```

A new naming convention is introduced in the next few #defines. A #define used to represent the resource ID of a MENU starts with a lower-case *m*. A #define representing an item number starts with a lower-case *i*.

```
#define mApple              kBaseResID
#define iAbout              1

#define mFile               kBaseResID+1
#define iQuit               1

#define mFont               100

#define mStyle              101
#define iPlain              1
#define iBold               2
#define iItalic             3
#define iUnderline          4
#define iOutline            5
#define iShadow             6

#define kPlainStyle         0

#define kExtraPopupPixels   25

#define kClockLeft          12
#define kClockTop           25
#define kClockSize          24

#define kCurrentTimeZone    1
#define kNewYorkTimeZone    2
#define kMoscowTimeZone     3
#define kUlanBatorTimeZone  4
```

The macros `TopLeft()` and `BottomRight()` convert a Rect into a Point, either the top-left or bottom-right corner.

```
#define TopLeft( r )        (*(Point *) &(r).top)
#define BottomRight( r )    (*(Point *) &(r).bottom)
```

These three macros turn some hard-to-read, bit-manipulating code into simple functions.

```
#define IsHighBitSet( longNum )    ( (longNum >> 23) & 1 )
#define SetHighByte( longNum )    ( longNum |= 0xFF000000 )
#define ClearHighByte( longNum )  ( longNum &= 0x00FFFFFF )
```

The global `gDone` is initialized to false but becomes true when **Quit** is selected from the **File** menu. `gHasPopupControl` is set to true if the special pop-up menu control definition procedure (the Toolbox function that knows how to work with pop-up CNTLs) is installed.

```
Boolean    gDone, gHasPopupControl;
```

`gLastFont` specifies which item in the **Font** menu should have a check mark next to it; that is, which is the current font. `gCurrentZoneID` specifies which of the time zones in the pop-up menu was selected last. `gCurrentStyle` specifies the current style of the text in the clock window. `gCurrentStyle` can be a combination of several styles. It starts as the plain style. `gClockRect` will be set to the rectangle that needs to be updated once every second—a rectangle surrounding the clock but not including the pop-up menu.

```
short      gLastFont = 1, gCurrentZoneID = kCurrentTimeZone;
Style      gCurrentStyle = kPlainStyle;
Rect       gClockRect;
```

Next come WorldClock's function prototypes:

```
void      ToolBoxInit( void );
void      WindowInit( void );
void      MenuBarInit( void );
void      EventLoop( void );
void      DoEvent( EventRecord *eventPtr );
void      HandleNull( EventRecord *eventPtr );
void      HandleMouseDown( EventRecord *eventPtr );
void      SetUpZoomPosition( WindowPtr window, short
                           zoomInOrOut );
void      HandleMenuChoice( long menuChoice );
void      HandleAppleChoice( short item );
void      HandleFileChoice( short item );
void      HandleFontChoice( short item );
void      HandleStyleChoice( short item );
void      DoUpdate( EventRecord *eventPtr );
long      GetZoneOffset( void );
```

`main()` initializes the Toolbox, sets up the clock window, initializes the menus, and enters the event loop.

```

/***** main *****/

```

```

void    main( void )
{
    ToolboxInit();
    WindowInit();
    MenuBarInit();

    EventLoop();
}

```

`ToolBoxInit()` should be familiar by now:

```

/***** ToolBoxInit *****/

```

```

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

`WindowInit()` starts by loading the WIND resource used as a basis for the clock window.

```

/***** WindowInit *****/

```

```

void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

```

As before, if the WIND resource isn't found, the program beeps and exits. If this happens, check the name of your resource file.

```

    if ( window == nil )
    {

```

```

        SysBeep( 10 );    /* Couldn't load the WIND
                           resource!!! */
        ExitToShell();
    }

```

Once loaded, the window is made the current port and the text size is set to 24. As an experiment, try creating a giant clock. Start by changing `kClockSize` to 72. You'll need to change the `WIND`'s height and width, as well as the `BoundsRect` of the pop-up `CNTL`.

```

SetPort( window );
TextSize( kClockSize );

```

Next, `gClockRect` is set to the window's bounding `Rect`. To update the time once every second, `WorldClock` erases `gClockRect`, then draws the new time. This happens in the function `DoUpdate()`.

```

gClockRect = window->portRect;

```

Finally, `WindowInit()` makes the window visible.

```

ShowWindow( window );
}

```

`MenuBarInit()` starts by calling `GetNewMBar()` to load the `MBAR` resource. `GetNewMBar()` loads each of the `MENUS` specified in the `MBAR` resource, creating a data structure known as a **menu list**. `GetNewMBar()` returns a `Handle` to the menu list.

The menu list `Handle` returned by `GetNewMBar()` is passed to `SetMenuBar()`. `SetMenuBar()` copies the specified menu list into the master menu list for your application. If you wanted to offer several different menu bar configurations for your application (*à la* Microsoft Word), you'd use `SetMenuBar()` to switch between them.


```

/***** MenuBarInit *****/


void    MenuBarInit( void )
{
    Handle        menuBar;
    MenuHandle     menu;
    ControlHandle  control;
    OSErr          myErr;
    long           feature;

    menuBar = GetNewMBar( kBaseResID );
    SetMenuBar( menuBar );

```



`GetMHandle()` takes a MENU resource ID and returns a Handle to that MENU's data in the menu list. If the specified MENU is not in the menu list, `GetMHandle()` returns nil. This call to `GetMHandle()` returns a Handle to the  menu:



```
menu = GetMHandle( mApple );
```


`AddResMenu()` adds the names of all resources of a given type to the specified menu. In this case, we are asking `AddResMenu()` to add all resources of type 'DRVR' to the  menu:

```
AddResMenu( menu, 'DRVR' );
```



In the days before System 7, the only items you'd find in an  menu would be an **About xxx...** item, a separator line, and a list of desk accessories. Since desk accessories come packed into resources of type 'DRVR', `AddResMenu()` was used to add the names of all available desk accessories to the  menu.

Though System 7 opened up the  menu to more than just DAs, the techniques used for DA handling work fine for all types of  menu items.

The **Font** and **Style** menus require a slightly different strategy. While the , **File**, **Edit**, and **Special** menus appear in the menu bar, the **Font** and **Style** menus only appear as submenus. If we included the **Font** and **Style** menus in the MBAR resource, the words **Font** and **Style** would appear as fifth and sixth titles in the menu bar, right alongside the other four.

The solution here is to load the two hierarchical menus into the menu list by hand, marking each as a nonmenu bar menu. `GetMenu()` loads the specified MENU into memory, but does not add the menu to the menu list.

```
menu = GetMenu( mFont );
```

`InsertMenu()` adds the menu to the menu list, and normally adds it to the menu bar. By passing -1 as the second parameter, we're telling `InsertMenu()` to mark the menu as hierarchical, so it won't appear in the menu bar.

```
InsertMenu( menu, kNotANormalMenu );
```

This call to `AddResMenu()` adds the names of all the 'FONT' resources to the end of the specified menu. Since the **Font** MENU was empty to begin with, we've created a menu made up completely of font names.

```
AddResMenu( menu, 'FONT' );
```

The same basic technique was used for the **Style** MENU, though no font names were added. In this case, the function `CheckItem()` was called to place a check mark next to the first item in the **Style** menu, **Plain**.

```
menu = GetMenu( mStyle );  
InsertMenu( menu, kNotANormalMenu );  
CheckItem( menu, iPlain, true );
```

Next, `DrawMenuBar()` draws the menu bar. Call `DrawMenuBar()` any time you make a change to the menu titles in the menu bar. If you're just changing a menu's item, there's no need to call `DrawMenuBar()`.

```
DrawMenuBar();
```

`HandleFontChoice()`, defined below, handles a selection from the **Font** menu. In this case, we've simulated a selection of the first font in the **Font** menu (`gLastFont` was initialized to 1).

```
HandleFontChoice( gLastFont );
```

We first used `Gestalt()` to check for the availability of `AppleEvents` in Chapter 4. In this case, we're checking to see if the pop-up control mechanism is available. The global `gHasPopupControl` is set to true if the pop-up control mechanism is installed.

```
myErr = Gestalt( gestaltPopupAttr, &feature );  
  
gHasPopupControl = ((myErr == noErr) && (feature &  
    (1 << gestaltPopupPresent)));
```

If the pop-up control is available, call `GetNewControl()` to load the CNTL from the resource file. `GetNewControl()` takes two parameters. The first specifies the resource ID of the CNTL. The second specifies the window in which the control should be displayed. Since `WorldClock` uses only one window, it's safe to assume that `FrontWindow()` will return the correct one.

At this point, we won't get into too much detail on the ins and outs of the Control Manager. For now, focus on the steps you'll need to take to add a System 7-savvy pop-up menu to your own applications. We'll get to the Control Manager in Chapter 6.

```

    if ( gHasPopupControl )
        control = GetNewControl( kPopupControlID,
                                FrontWindow() );
}

```

EventLoop() is almost the same as the version found in the first two programs in Chapter 4. The difference lies in the handling of the Boolean value returned by WaitNextEvent(). In Chapter 4, a false return value was ignored. If there are no events pending in the event queue, WaitNextEvent() returns a **null event** in the event parameter and returns a value of false. In the case of a null event, we pass the event on to a function called HandleNull(), which is covered a little bit further on in this chapter.

```

/***** EventLoop *****/

```

```

void    EventLoop( void )
{
    EventRecord    event;

    gDone = false;

```

As mentioned earlier, gDone will be set to true when **Quit** is selected from the **File** menu.

```

    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, kSleep,
                            nil ) )
            DoEvent( &event );
        else
            HandleNull( &event );
    }
}

```

DoEvent() is similar to the version presented in Chapter 4. The field eventPtr->what tells you what type of event is being processed.



```

/***** DoEvent *****/

```

```

void      DoEvent( EventRecord *eventPtr )
{
    char      theChar;

    switch ( eventPtr->what )
    {

```

In the case of a mouseDown, pass the event on to HandleMouseDown().

```

        case mouseDown:
            HandleMouseDown( eventPtr );
            break;

```

A keyDown is generated as soon as a key is pressed. An autoKey is generated when a key is held down longer than the autoKey threshold (stored in the system global KeyThresh). The fields in the EventRecord are the same in either case.

In a keyDown or autoKey event, the lowest byte of the message field contains the character code of the pressed key. For example, if the a key were pressed, the low byte of the message field contains an ASCII 'a'. This value is placed in theChar.

```

        case keyDown:
        case autoKey:
            theChar = eventPtr->message & charCodeMask;

```

If the Command key (⌘) was pressed when the event occurred, pass theChar to MenuKey(). MenuKey() looks through the menu list to find which menu item, if any, had the Command-key equivalent corresponding to theChar. For example, if theChar held an ASCII 'Q' or 'q' (case is not distinguished), MenuKey() would locate the File menu's Quit item.

If it finds a matching item, MenuKey() highlights the menu's title, then returns a 4-byte value containing the menu ID (high 2 bytes) and the item number (low 2 bytes). As you'll see in HandleMouseDown(), this is exactly what happens when you call MenuSelect().

```

        if ( (eventPtr->modifiers & cmdKey) != 0 )
            HandleMenuChoice( MenuKey( theChar ) );
        break;

```

If an `updateEvt` occurs, pass the event on to `DoUpdate()`.

```

        case updateEvt:
            DoUpdate( eventPtr );
            break;
    }
}

```

`HandleNull()` gets called whenever `WaitNextEvent()` is called and there are no events in the queue. The static variable `lastTime` is just like a global variable in that it keeps its value even after `HandleNull()` exits. We'll use it to buffer the time stored in the field `eventPtr->when`. The `EventRecord`'s `when` field tells you exactly when the event occurred. The time is described in ticks (60ths of a second) since system startup.

```

/***** HandleNull *****/

```

```

void HandleNull( EventRecord *eventPtr )
{
    static long lastTime = 0;

```

By dividing the number of ticks since startup by 60, we calculate the number of seconds since startup. Skip down a few lines and you'll see that we buffer the number of seconds since startup in `lastTime`. The `if` statement checks to see if the number of seconds has changed since the last time the clock was updated. If so, we use `InvalRect()` to force an update on the clock area of the window, then save the time in `lastTime`.

```

        if ( (eventPtr->when / kTicksPerSecond) > lastTime )
        {
            InvalRect( &gClockRect );
            lastTime = eventPtr->when / kTicksPerSecond;
        }
}

```

`HandleMouseDown()` handles all mouseDown events.

```

/***** HandleMouseDown *****/

```

```

void HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr      whichWindow;
    GrafPtr        oldPort;

```

```
short          thePart;
long           menuChoice;
ControlHandle  control;
short          ignored;
```

Just as we did in Chapter 4, we use `FindWindow()` to determine in which window, and which area within the window, the `mouseDown` occurred.

```
thePart = FindWindow( eventPtr->where, &whichWindow );
switch ( thePart )
{
```

If the `mouseDown` was in the menu bar, call the function `MenuSelect()`. `MenuSelect()` takes the mouse's location as a parameter and tracks the mouse, pulling down menus and highlighting items until the mouse button is released. If an item is selected from a menu, `MenuSelect()` will leave the menu's title highlighted, just as `MenuKey()` did earlier. `MenuSelect()` also returns the same 4-byte value as `MenuKey()`, with the menu ID in the upper 2 bytes and the item number in the lower 2 bytes. This value is passed on to `HandleMenuChoice()`, described below.

```
case inMenuBar:
    menuChoice = MenuSelect( eventPtr->where );
    HandleMenuChoice( menuChoice );
    break;
```

When a `mouseDown` occurs in a system window (most likely a desk accessory), call `SystemClick()` to handle the event properly.

```
case inSysWindow:
    SystemClick( eventPtr, whichWindow );
    break;
```

If the `mouseDown` occurred in the window's content region, first make sure it's the current port, then convert the location from global to the local coordinates of the window.

```
case inContent:
    SetPort( whichWindow );
    GlobalToLocal( &eventPtr->where );
```

Next, use `FindControl()` to determine whether the `mouseDown` was in the pop-up menu control (the only control inside the clock window's content region). If the `mouseDown` was in a control, call

`TrackControl()`. In the case of the pop-up menu control, `TrackControl()` inverts the title, pops up the menu, and highlights items until the mouse button is released.

Once the mouse button is released, call `GetCtlValue()` to get the control's current value. In this case, the control's value corresponds to the number of the last item selected from the pop-up. This value is stored in `gCurrentZoneID`.

You'll learn more about `GetCtlValue()` in Chapter 6.

```
if ( FindControl( eventPtr->where,
                  whichWindow, &control ) )
{
    ignored = TrackControl( control,
                           eventPtr->where,
                           kUseDefaultProc );
    gCurrentZoneID = GetCtlValue( control );
}
break;
```

If the `mouseDown` was in the drag region, call `DragWindow()` to drag the window around the screen.

```
case inDrag:
    DragWindow( whichWindow, eventPtr->where,
                &screenBits.bounds );
    break;
```

If the `mouseDown` was in the zoom box, start by calling `TrackBox()` to animate the zoom box. If the cursor was in the zoom box when the mouse button was released, `TrackBox()` returns true.

```
case inZoomIn:
case inZoomOut:
    if ( TrackBox( whichWindow, eventPtr->where,
                  thePart ) )
    {
```

In that case, call `SetUpZoomPosition()` to set up the window's zoomed-in and zoomed-out coordinates, then call `ZoomWindow()` to alternate between the clock window and the clock and pop-up menu window positions.

```
        SetUpZoomPosition( whichWindow, thePart );
        ZoomWindow( whichWindow, thePart,
                    kLeaveWhereItIs );
```

```

    }
    break;
}
}

```

SetUpZoomPosition() takes advantage of a little-known Macintosh feature, the **zoom state rectangles**. When a WindowRecord is created, the Window Manager checks to see if the window contains a zoom box. If so, a Handle to a pair of Rects is embedded in the WindowRecord. These Rects correspond to the position of the window as it toggles between the user and standard state.

Typically, the standard state represents the size and position of the window as it first appears. The user state corresponds to the window's size and position after the user starts mucking with it, dragging it around the screen, and changing its size. The zoom box allows the user to toggle between these two states.

WorldClock interprets these two states in a slightly different manner. One state corresponds to the position and size of the window as SetUpZoomPosition() gets called. The other state corresponds to the same position, but differs in the height of the window. If the window currently displays the clock only, the other state corresponds to the window displaying both the clock and pop-up menu. This will become clearer in a bit.

```

/***** SetUpZoomPosition *****/

void      SetUpZoomPosition( WindowPtr window, short
                           zoomInOrOut )
{
    WindowPeek      wPeek;
    WStateData      *wStatePtr;
    Rect            windowRect;
    Boolean          isBig;
    short           deltaPixels;

```

For starters, we need to cast the WindowPtr to the type WindowPeek. Take a look at the declaration of the WindowRecord on (I:276). A WindowPtr is defined to be a GrafPtr, a pointer to a GrafPort. This means that a WindowPtr only allows you access to the fields within the port field of the WindowRecord. On the other hand, a WindowPeek is defined as a pointer to a WindowRecord. By casting the WindowPtr to a WindowPeek, we gain access to the rest of the fields within the WindowRecord.

```

wPeek = (WindowPeek) window;

```

Next, we set `wStatePtr` to point to the two `Rects` embedded in the `WindowRecord`'s `DataHandle`. Since a `Handle` is a pointer to a pointer, we set `wStatePtr` to `*DataHandle`.



As we've stated before, there's more to handles than meets the eye. Volume II of the *Primer* goes into detail on the proper usage of handles.

```
wStatePtr = (WStateData *) *(wPeek->dataHandle);
```

Next, the window's `portRect` is copied into `windowRect`. At this point, `windowRect` is in the window's local coordinates. The two macros `TopLeft()` and `BottomRight()` (defined at the top of the file) are used along with `LocalToGlobal()` to convert the `Rect` from local to global coordinates. Although it looks kind of funky, this method is fully approved by Apple.

```
windowRect = window->portRect;
LocalToGlobal( &TopLeft( windowRect ) );
LocalToGlobal( &BottomRight( windowRect ) );
```

Next, both `stdState` and `userState` are set to the globalized `windowRect`.

```
wStatePtr->stdState = windowRect;
wStatePtr->userState = wStatePtr->stdState;
```

If the pop-up menu control mechanism is installed, `isBig` is set to true if the window is set to "clock and pop-up menu" size, false otherwise.

```
if ( gHasPopupControl )
{
    isBig = (windowRect.bottom - windowRect.top) >
            (gClockRect.bottom - gClockRect.top);
}
```

If the window is big, a click in the zoom box should make it shorter. If the window is small, a click in the zoom box should make it longer.

```
if ( isBig )
    deltaPixels = -kExtraPopupPixels;
else
    deltaPixels = kExtraPopupPixels;
```

The parameter `zoomInOrOut` tells us whether we're zoomed in, about to zoom out, or zoomed out, about to zoom in. This is the same as asking: are we in the user state, or are we in the standard state? Use this info to set the destination state to the correct length.

```
if ( zoomInOrOut == inZoomIn )
    wStatePtr->userState.bottom += deltaPixels;
else
    wStatePtr->stdState.bottom += deltaPixels;
}
```

If the pop-up menu control mechanism is not installed, call `SysBeep()`. This is a clue that you are trying to run this program under an old operating system.



It's important to note that WorldClock will run under pre-System 7 operating systems. Its System 7 specific features will be disabled, but it will not crash and burn. If possible, do the same for your programs.

```
else
    SysBeep( 20 );
}
```

`HandleMenuChoice()` starts by unpacking the 4-byte value passed to it into two shorts.

```
/****** HandleMenuChoice *****/
```

```
void HandleMenuChoice( long menuChoice )
{
    short    menu;
    short    item;

    if ( menuChoice != 0 )
    {
```

`menu` gets the high 2 bytes and `item` gets the low 2 bytes. `HiWord()` and `LoWord()` are both Toolbox functions.

```

menu = HiWord( menuChoice );
item = LoWord( menuChoice );

```

Next, the item is passed to the appropriate handler for the specified menu.

```

switch ( menu )
{
    case mApple:
        HandleAppleChoice( item );
        break;
    case mFile:
        HandleFileChoice( item );
        break;
    case mFont:
        HandleFontChoice( item );
        break;
    case mStyle:
        HandleStyleChoice( item );
        break;
}

```

Once the menu command is processed, HiliteMenu() is called to uninvert any inverted menu titles.

```

        HiliteMenu( 0 );
    }
}

```

For now, if the **About WorldClock...** item is selected, we'll just beep. Chapter 6 builds a proper about box for its application.

```

/***** HandleAppleChoice *****/

void HandleAppleChoice( short item )
{
    MenuHandle    appleMenu;
    Str255        accName;
    short         accNumber;

    switch ( item )
    {
        case iAbout: /* We'll put up an about box next
                        chapter.*/
            SysBeep( 20 );
            break;
    }
}

```



If another item is selected, it is treated as if it were a desk accessory. `GetItem()` converts the `MenuHandle` and item number to the item name, embedded in a Pascal string. Next, `OpenDeskAcc()` is called to launch the DA.

```

        default:
            appleMenu = GetMHandle( mApple );
            GetItem( appleMenu, item, accName );
            accNumber = OpenDeskAcc( accName );
            break;
    }
}

```

`HandleFileChoice()` sets `gDone` to true if **Quit** was selected.

```

/***** HandleFileChoice *****/

void    HandleFileChoice( short item )
{
    switch ( item )
    {
        case iQuit :
            gDone = true;
            break;
    }
}

```

`HandleFontChoice()` uses `CheckItem()` to remove the check mark from the last font selected and then add the check mark to the font just selected.

```

/***** HandleFontChoice *****/

void    HandleFontChoice( short item )
{
    short          fontNumber;
    Str255         fontName;
    MenuHandle     menuHandle;

    menuHandle = GetMHandle( mFont );

    CheckItem( menuHandle, gLastFont, kRemoveCheckMark );
    CheckItem( menuHandle, item, kAddCheckMark );
}

```

Next, `gLastFont` is updated to reflect this latest font selection.

```
gLastFont = item;
```

`GetItem()` is called to get a Pascal string with the new font's name in it. This string is passed to `GetFNum()`, which turns the font name into a number. Finally, the font number is passed to `TextFont()` to set the port's font, used the next time the clock is redrawn.

```
GetItem( menuHandle, item, fontName );
```

```
GetFNum( fontName, &fontNumber );
```

```
TextFont( fontNumber );
```

```
}
```

`HandleStyleChoice()` uses the item selected from the **Style** menu to update `gCurrentStyle`. `gCurrentStyle` is a bitmap containing a bit for each possible style. `gCurrentStyle` has a value of 0 (`kPlainStyle`) if no styles are set.

```
/****** HandleStyleChoice *****/
```

```
void HandleStyleChoice( short item )
```

```
{
```

```
    MenuHandle menuHandle;
```

```
    switch( item )
```

```
    {
```

```
        case iPlain:
```

```
            gCurrentStyle = kPlainStyle;
```

```
            break;
```

For each style, if the style's bit is currently set, the bit must be cleared. If the bit is cleared, it must be set.

```
        case iBold:
```

```
            if ( gCurrentStyle & bold )
```

```
                gCurrentStyle -= bold;
```

```
            else
```

```
                gCurrentStyle |= bold;
```

```
            break;
```

```
        case iItalic:
```

```
            if ( gCurrentStyle & italic )
```

```
                gCurrentStyle -= italic;
```

```

        else
            gCurrentStyle |= italic;
        break;
    case iUnderline:
        if ( gCurrentStyle & underline )
            gCurrentStyle -= underline;
        else
            gCurrentStyle |= underline;
        break;
    case iOutline:
        if ( gCurrentStyle & outline )
            gCurrentStyle -= outline;
        else
            gCurrentStyle |= outline;
        break;
    case iShadow:
        if ( gCurrentStyle & shadow )
            gCurrentStyle -= shadow;
        else
            gCurrentStyle |= shadow;
        break;
}

```

Once the proper bit has been adjusted, `CheckItem()` is used to place or remove the check mark next to each style, depending on whether the style's bit is set or cleared.

```

menuHandle = GetMHandle( mStyle );

CheckItem( menuHandle, iPlain, gCurrentStyle ==
            kPlainStyle );
CheckItem( menuHandle, iBold, gCurrentStyle & bold );
CheckItem( menuHandle, iItalic, gCurrentStyle &
            italic );
CheckItem( menuHandle, iUnderline, gCurrentStyle &
            underline );
CheckItem( menuHandle, iOutline, gCurrentStyle &
            outline );
CheckItem( menuHandle, iShadow, gCurrentStyle &
            shadow );

```

Finally, `gCurrentStyle` is used to set the text style for the clock window.

```

    TextFace( gCurrentStyle );
}

```

`DoUpdate()` handles the clock window's updateEvts.

```

/***** DoUpdate *****/

```

```

void DoUpdate( EventRecord *eventPtr )
{
    WindowPtr      window;
    Str255          timeString;
    unsigned long   curTimeInSecs;

```

First, the `WindowPtr` is pulled out of the `EventRecord`'s message field. Next, `BeginUpdate()` is called.

```

    window = (WindowPtr)eventPtr->message;

```

```

    BeginUpdate( window );

```

`GetDateTime()` is called to retrieve the current time in seconds. `GetZoneOffset()` returns the number of seconds needed to adjust for a change in time zones.

```

    GetDateTime ( &curTimeInSecs );
    curTimeInSecs += GetZoneOffset();

```

`IUTimeString()` converts `curTimeInSecs` to a Pascal string. `IUTimeString()` determines the string's format from International Resource 1 (I:500). Basically, the International Resources enable your programs to run on any Macintosh, in any country, without change. In this case, the time in the clock window will appear in a form that makes sense for the current country.

```

    IUTimeString( (long)curTimeInSecs, kIncludeSeconds,
                  timeString );

```

To draw the time, the old time is first erased. Next, the pen is moved to the proper location, and the time string is drawn.

```

    EraseRect( &gClockRect );
    MoveTo( kClockLeft, kClockTop );
    DrawString( timeString );

```

Next, `DrawControls()` is called to draw the controls for this window. Since the only control in this window is the pop-up menu control, this has the effect of updating the pop-up menu label.

```
DrawControls( window );
```

As always, every call to `BeginUpdate()` must be matched with a call to `EndUpdate()`.

```
EndUpdate( window );
}
```

`GetZoneOffset()` calls the System 7 utility `ReadLocation()` (VI:14:49) to fetch information about this Mac's geographic location.



Apple designed the Macintosh with the global marketplace in mind. For example, the first Macintosh Toolbox came equipped with the International Utilities Package, designed to give programmers access to country-specific number, currency, date, and time formats.

With the introduction of System 6.07, Apple added another element to its globalization strategy. `ReadLocation()` and `WriteLocation()` join the International Utilities as part of the Script Manager. A **script** is a language-specific writing system, such as Japanese, Arabic, or Hebrew. By working with the Script Manager, you can design programs that can easily be ported to countries using different script systems. If you plan on developing for the global market, Volume VI, Chapter 14 of *Inside Macintosh* is a must read.

For now, we'll focus on the part of the Script Manager that lets you access information regarding your Macintosh's geographic location and time zone.

```
/***** GetZoneOffset *****/
```

```
long GetZoneOffset( void )
{
    MachineLocation  loc;
    long             delta, defaultZoneOffset;
```

`ReadLocation()` retrieves the current geographic and time-zone information from **parameter RAM**.



Parameter RAM (or P-RAM) is a portion of memory, backed up by a battery, used to store long-term information about your Macintosh. For example, when you use the General Control Panel to edit the current time, you are actually editing a location in P-RAM. Since P-RAM is backed up by a battery, you don't have to reset the time every time you power off your Mac.

```
ReadLocation( &loc );
```

The next few lines demonstrate the proper method for calculating the time-zone offset, the number of seconds the clock needs to be adjusted to make up for any differences in time zone.

```
defaultZoneOffset = ClearHighByte
                    ( loc.gmtFlags.gmtDelta );
```

```
if ( IsHighBitSet( defaultZoneOffset ) )
    SetHighByte( defaultZoneOffset );
```

`delta` is set to the number of seconds needed to make up for a difference in time zone of the requested city.

```
switch ( gCurrentZoneID )
{
    case kCurrentTimeZone :
        delta = defaultZoneOffset;
        break;
    case kNewYorkTimeZone :
        delta = -5L * kSecondsPerHour ;
        break;
    case kMoscowTimeZone :
        delta = 3L * kSecondsPerHour;
        break;
    case kUlanBatorTimeZone :
        delta = 8L * kSecondsPerHour;
        break;
}
```

Next, delta is offset to adjust for the current time zone, then returned.

```
    delta -= defaultZoneOffset;

    return delta;
}
```

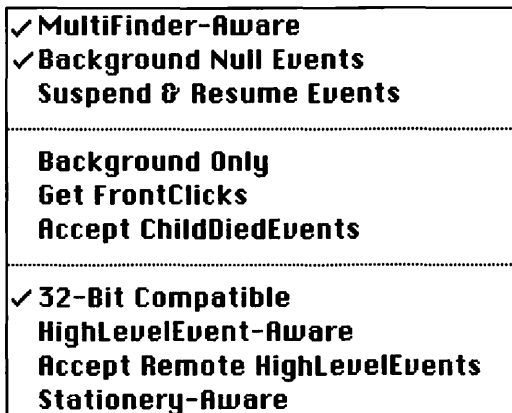
## Variants

There are several interesting things you can do with WorldClock. As suggested earlier, try changing the clock's size. You'll need to change the font size and you'll also need to adjust the size of the clock window and the position of the pop-up menu.

Try adding a few more cities to the WorldClock pop-up menu. To maximize WorldClock's effectiveness, pick a city in each of the world's 24 time zones.

Next, make WorldClock run in the background. To see why, run WorldClock. Then, with WorldClock still running, bring another program to the foreground. You may have to move the WorldClock window around to see this effect but, when you get things set up right, you should notice that the clock window stops updating when it is in the background. This is because WorldClock is not set up to receive **Background Null Events**.

To change this, **Quit** WorldClock and, once back in THINK C, select **Set Project Type...** from the **Project** menu. Next, click on the **SIZE flags** pop-up menu and select **Background Null Events** (Figure 5.26). Click **OK**.



**Figure 5.26** The pop-up menu from the **Set Project Type...** dialog.

Now when you run WorldClock, your program will receive null events, even when it is in the background. The `SIZE` resource, which you just modified, is discussed in detail in Chapter 8.

---

## In Review

---

Menus are an intrinsic part of the Macintosh interface. Designing them correctly allows you to take advantage of the familiarity of users with standard Mac menus. The standard pull-down menu does the job for many applications, and hierarchical and pop-up menus bring freshness to the interface.

In Chapter 6, you'll learn about another essential part of the Mac interface: creating and controlling dialog boxes. While you're there, you'll also look at one of the newest managers on the Macintosh: the Process Manager.



---

# Working with Dialogs

---

*Dialogs present a list of alternatives for the user to choose from. Alerts are simplified dialogs, used to report errors and give warnings to the user. Chapter 6 discusses both of these, along with the Notification Manager, Apple's background notification mechanism, and the Process Manager, which can be used to launch other applications.*

---

DIALOGS ARE AN important part of the Macintosh interface; they provide a friendly, standardized way of communicating and receiving feedback from the user. Some dialogs ask questions of the user, or offer the user the opportunity to modify current program parameters (Figure 6.1). Some dialogs are the direct result of a user menu selection. For example, when you select **Print...** from within an application, the **Print Job** dialog appears (Figure 6.2).

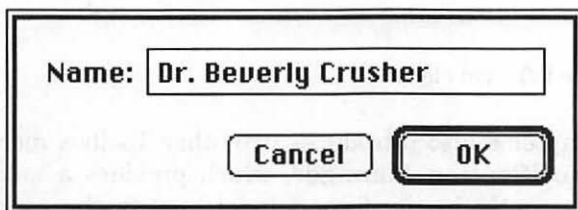


Figure 6.1 "What's your name?" dialog box.

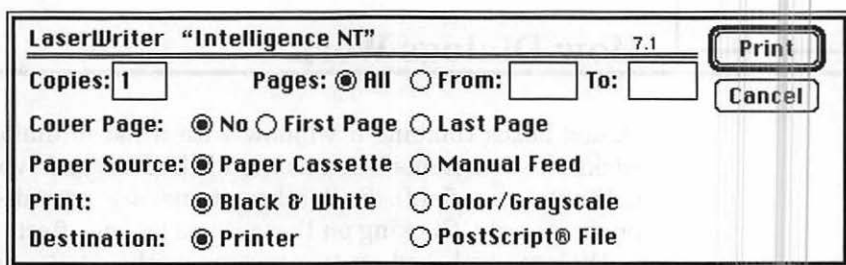
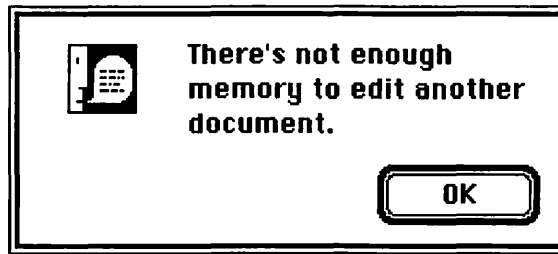


Figure 6.2 Print Job dialog box.



By convention, menu items that spawn dialog boxes always end with an ellipsis (...). For example, the **Print...** item on the **File** menu brings up a **print** dialog box.

Another important part of the Mac interface is the **alert mechanism**. **Alerts** (Figure 6.3) are simplified dialogs, used to report errors and give warnings to the user. From a programmer's point of view, alerts are simple to implement, but not as flexible as dialogs.



**Figure 6.3** An alert.

Chapter 6 also introduces two other Toolbox managers: the first is the **Notification Manager**, which provides a method for programs not currently in the foreground to notify the user of an important event. The second is the **Process Manager**, which allows you to launch other applications from your own application.

---

## How Dialogs Work

---

Dialog boxes combine a window with a list of dialog items. The user clicks on some items (such as the **OK** button) and types text into others (editable text field). Still other items are provided for information purposes only. Clicking on these items has no effect.

Dialogs are based on two resources, the **DLOG** and the **DITL**. Much like a **WIND** resource, the **DLOG** serves as a template for the dialog window. The **DITL** resource contains the **dialog item list**, a list of all items that appear in the dialog window. As you'll see when we get to our program, ResEdit provides everything you'll need to create these resources.

Typical dialog items include check boxes, radio buttons, and push buttons. These items are known as **controls**. In addition, static text fields, editable text fields, **PICTs**, and **ICONS** may also be part of an item list (Figure 6.4). By convention, most dialog boxes offer an **OK** button, which saves your changes, and a **Cancel** button, which gives users a chance to back out without saving their changes.

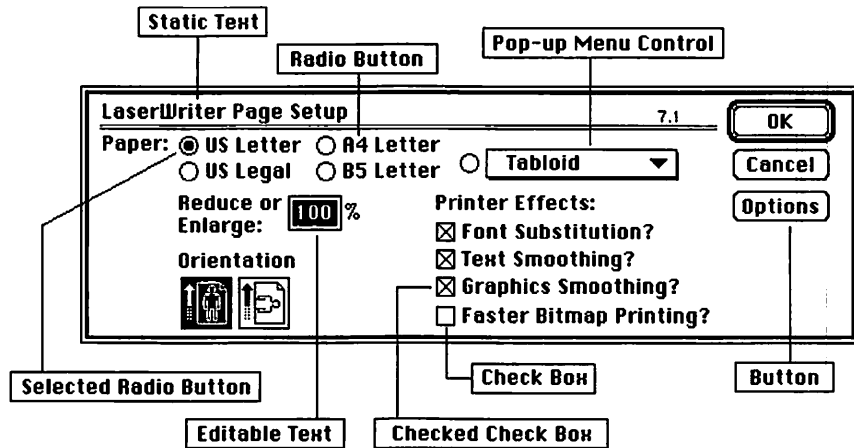


Figure 6.4 Dialog items in a **Page Setup...** dialog box.

## Dialog Items: Controls

Controls are one of the most important types of dialog items. Controls exist in at least two different states. For example, the check box can be checked or unchecked (Figure 6.5). Although controls may be defined by the program designer, four types of controls are already defined in the Toolbox. They are **buttons**, **check boxes**, **radio buttons**, and **dials**.

These controls fall under the jurisdiction of the **Control Manager**, which handles the creation, editing, and use of controls.

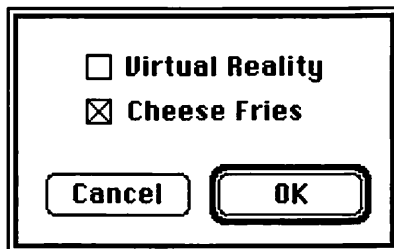


Figure 6.5 Checking a check box.

## Buttons

The classic example of a button is the **OK** button found in most dialog boxes (Figure 6.6). When the mouse button is released with the cursor inside the button, the button's action is performed. For example, clicking an **OK** button might start a print job or save an application's data.

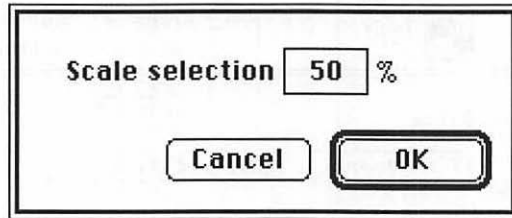


Figure 6.6 The button.



Those of you familiar with HyperCard should note the similarity between HyperCard buttons and Toolbox buttons. Toolbox buttons are rounded-corner rectangles, whereas HyperCard buttons have more variation in shape and appearance.

## Check Boxes

Check boxes are generally used to set options. For example, you might use a check box to determine whether the user wants sound turned on or off in an application (Figure 6.7).

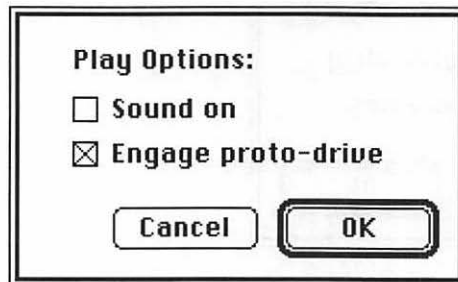


Figure 6.7 Check box example.

## Radio Buttons

Radio buttons are similar to check boxes in function, in that they also are generally used to set options or choices in a dialog box. Figure 6.8 shows some radio buttons. The difference between radio buttons and check boxes is that the choices displayed in radio buttons are mutually exclusive. Radio buttons appear in sets, and only one radio button in a set may be on (or highlighted) at any given time (Figure 6.9).

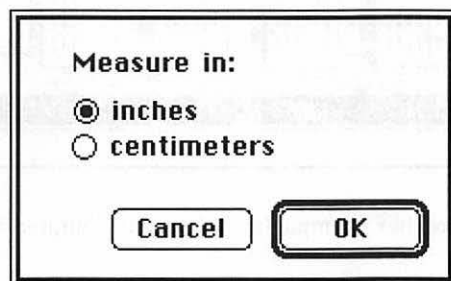


Figure 6.8 Radio button example.

## Dials

Dials are different from other controls: They display and supply qualitative instead of off/on information. Two familiar dial controls predefined in the Toolbox are the pop-up menu control you saw in the previous chapter, and the **scroll bar** (Figure 6.10), which is an integral part of many Mac application windows. In Chapter 7, we'll show you how to set up a scroll bar.

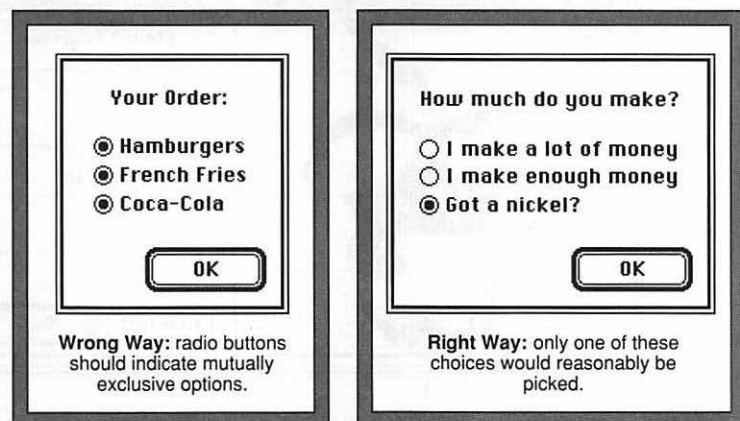


Figure 6.9 Radio button pointers.

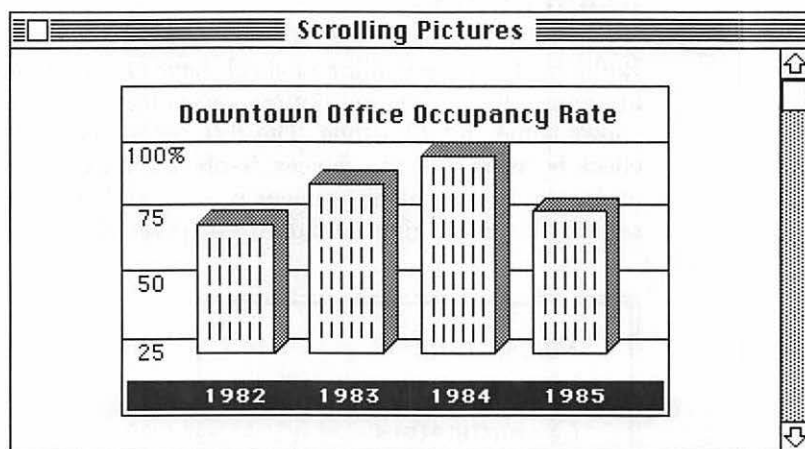


Figure 6.10 Scroll bar example (from Pager in Chapter 7).

## Other Dialog Items

Controls represent one type of dialog item. You can also display pictures (using a PICT resource, as shown in Figure 6.11) and icons (resource type ICON) in a dialog box. You can also add static and editable text fields, as well as **user items**, to your dialogs. A user item designates a rectangle in the dialog window's local coordinates. Typically, you'll use a user item as a placeholder, marking an area in

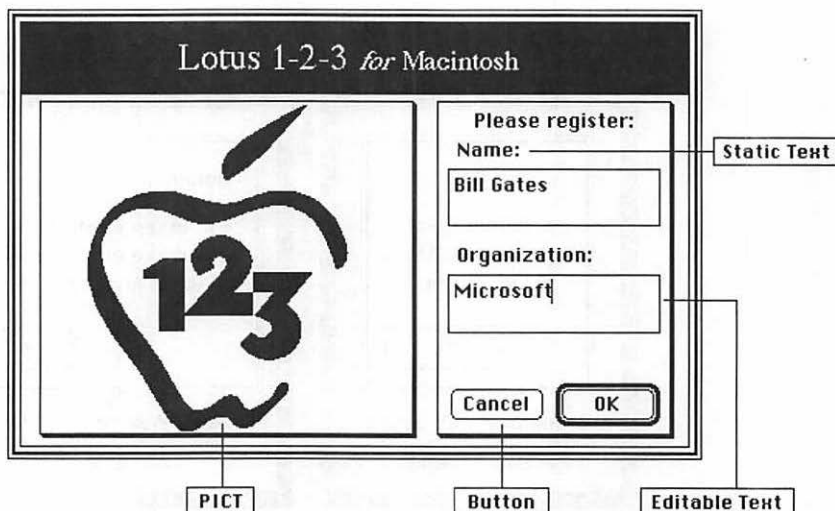


Figure 6.11 Other dialog items.

the dialog for a nonstandard dialog item. For example, your program might use a user item as a guide for placing a scrolling list created on the fly.

Not all dialogs are alike. For example, some dialogs force you to respond to them before you can go on with your program. Others don't. Some let you drag the dialog window around the screen. Some don't. Let's take a look at some of these different dialog types.

---

## Modal Dialogs

---

A **modal dialog** is one to which the user must respond before the program can continue. Modal dialogs are used for decisions that must be made immediately. They represent the vast majority of dialog boxes.



The Macintosh is generally a modeless machine. This means that most of the operations performed by an application are available to the user most of the time. For example, most of the operations performed by THINK C are available through pull-down menus. Modal dialogs come into play when you must focus the user's attention on a specific task or issue. Alerts are always modal. Dialog boxes aren't.

### The Modal Dialog Algorithm

To create a modal dialog, first load the dialog (including the dialog's item list) from the resource file using `GetNewDialog()`. Then, make the dialog window visible (just as you would a new window). Next, enter a loop, first calling `ModalDialog()` to find out which item the user selected, then processing that item. When an exit item (such as **OK** or **Cancel**) is selected, exit the loop.

---

## Modeless Dialogs

---

**Modeless dialogs** act more like regular windows; they appear to the user like any other window and can be brought to the front with a mouse click, or even dragged around the screen. Whereas modal dialogs require an immediate response from the user, modeless dialogs may be set aside until they are needed. The algorithms used to implement modal and modeless dialogs are quite different.

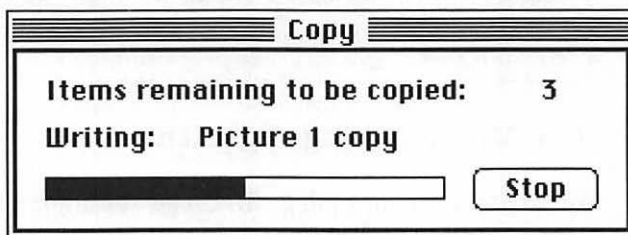


## The Modeless Dialog Algorithm

To implement a modeless dialog, start by loading the dialog and making it visible (as was done with the modal dialog). When an event is returned by `WaitNextEvent()`, pass it on to `IsDialogEvent()`. If `IsDialogEvent()` returns false, the event is not related to the dialog and should be handled normally. Otherwise, the event should be passed to `DialogSelect()`. `DialogSelect()` returns a pointer to the dialog box whose item was selected, as well as the number of the item selected by the user. Process the item as you would with `ModalDialog()`.



There's actually a third kind of dialog: the **movable modal** dialog box. Despite its name, a movable modal dialog is a window, not a dialog box. You create a movable modal by passing `movableDBoxProc` in your call to `NewWindow()`. To make a movable modal behave like a dialog box, you'll need to restrict user actions—for example, by beeping if the user clicks on another window, instead of bringing that window to the front. Figure 6.12 shows a movable modal dialog you probably see every day.



**Figure 6.12** A movable modal dialog box.

To learn more about movable modal dialogs, see Chapter 3-15 in Volume VI of *Inside Macintosh*.

## Adding Dialogs to Your Programs

In this section, we'll show you how to build modal dialog boxes through the use of `DLOG` and `DITL` resources. Although we could have created the dialog structure in `THINK C` instead, we chose to emphasize the resource-based approach.



We'll show you how to create DLOG and DITL resources when we get to the Reminder program later in the chapter.

As was stated in the dialog algorithm, to implement a dialog box in your application, load your dialog box resources, then loop around `ModalDialog()`, responding to clicks in the dialog box window.

Here's an outline of the procedure. First, set up #defines that correspond with the important items in your DITL. You'll want a #define for each DITL item that you want to track or change. As you'll see in a minute, the #define constant matches the item's DITL number. For example, if you designed a DITL with an **OK** button as item 1, a **Cancel** button as item 2, two radio buttons as items 3 and 4, and three check boxes as items 5, 6, and 7, you'd need these #defines:

```
#define iFirstRadio      3
#define iSecondRadio    4

#define iFirstCheckbox  5
#define iSecondCheckbox 6
#define iThirdCheckbox  7

#define kOn              1
#define kOff             0
.
.
.
```

You'll use `kOn` and `kOff` when you turn a radio button or a check box on or off. As you construct your dialog item #defines, you'll find it helpful to turn on ResEdit's **Show Item Numbers** feature. Figure 6.13 shows a sample DITL with item numbering turned on.

Once your #defines are in place, load your DLOG and DITL resources by calling `GetNewDialog()`:

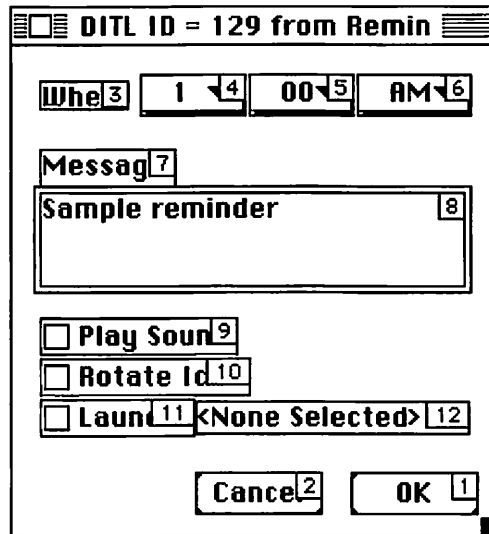
```
dialog = GetNewDialog( kDialogResID, nil, kMoveToFront );
```

Now, initialize each of your controls. Use `GetDItem()` to retrieve a handle to a control item. Then use `SetCtlValue()` to set the buttons, radio buttons, and check boxes to their initial values. For example, the following code will fill the first radio button and clear the second radio button in the dialog box described above:

```

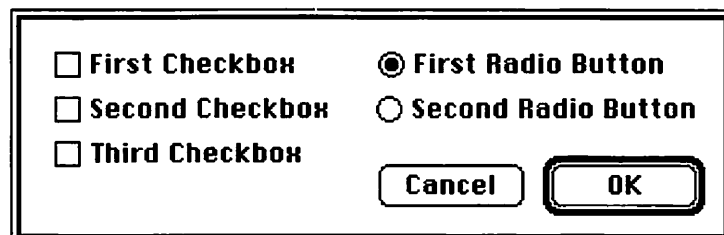
GetDItem( dialog, iFirstRadio, &itemType, &itemHandle,
          &itemRect );
SetCtlValue( itemHandle, kOn );
GetDItem( dialog, iSecondRadio, &itemType, &itemHandle,
          &itemRect );
SetCtlValue( itemHandle, kOff );
.
.
.

```



**Figure 6.13** Sample DITL. To see this in ResEdit, select **Show Item Numbers** in the DITL menu.

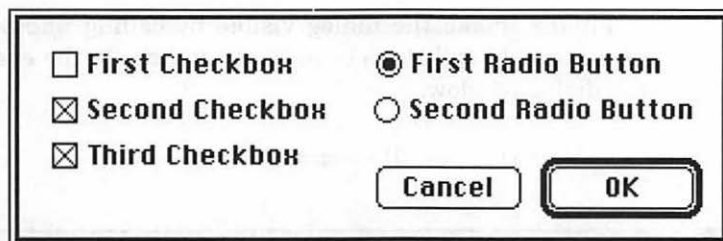
`iFirstRadio` and `iSecondRadio` correspond to the dialog's two radio button items. The first radio button will be set to `kOn` (highlighted), the second to `kOff` (Figure 6.14).



**Figure 6.14** Sample radio buttons initialized.

Here's some code to initialize the three check boxes described earlier. The code fragment clears the first check box and checks the second and third check boxes (Figure 6.15).

```
GetDItem( dialog, iFirstCheckBox, &itemType, &itemHandle,  
          &itemRect );  
SetCtlValue( itemHandle, kOff );  
GetDItem( dialog, iSecondCheckBox, &itemType, &itemHandle,  
          &itemRect );  
SetCtlValue( itemHandle, kOn );  
GetDItem( dialog, iThirdCheckBox, &itemType, &itemHandle,  
          &itemRect );  
SetCtlValue( itemHandle, kOn );
```



**Figure 6.15** Sample check boxes initialized.

Once your controls are initialized, you still have some housekeeping to do. If you plan on drawing in the dialog box with `QuickDraw` (which you might want to do with a **user item procedure**—see the tech block below), make the dialog the current port by passing the `DialogPtr` to `SetPort()`:

```
SetPort( dialog );
```

You can pass a `DialogPtr` to any routine that accepts a `WindowPtr`. In many ways, a dialog acts just like a window.



The Dialog Manager allows you to associate a drawing procedure with a particular user item. The procedure is then called whenever the item needs updating. *Inside Macintosh* is not particularly helpful on the topic of user items, so look at Tech Note #34 for details on implementing them in your programs.

`SetDialogDefaultItem()` allows you to specify a dialog's default item (usually the **OK** button). This function automatically draws the thick, rounded rectangle around the default item. `SetDialogCancelItem()` allows you to specify a dialog's cancel item. After this call is made, typing **⌘**. automatically selects the specified cancel item. Finally, `SetDialogTracksCursor()` enables the Dialog Manager to tie the i-beam cursor to any editable text items in a dialog. The `#defines` `ok` and `cancel` are set to 1 and 2, respectively, by the Dialog Manager, so it's a good idea to make sure that your **OK** button is item number 1 and your **Cancel** button item number 2.

```
SetDialogDefaultItem( dialog, ok );
SetDialogCancelItem( dialog, cancel );
SetDialogTracksCursor( dialog, true );
```

Finally, make the dialog visible by calling `ShowWindow()`. You're now ready to call `ModalDialog()` to handle the events that occur in the dialog window.

```
ShowWindow( dialog );
```



When you create your `DIALOG` in ResEdit, make sure the **Visible** box is unchecked. That way, if you load your dialog at the beginning of your program, it won't appear until you're ready.

In a manner somewhat similar to `WaitNextEvent()`, you continuously loop, calling `ModalDialog()` on each go-around, until `dialogDone` gets set to true.

```
dialogDone = false;
while ( ! dialogDone )
{
    ModalDialog( nil, &itemHit );

    switch( itemHit )
    {
        case ok:
        case cancel:
            dialogDone = true;
            break;
        case kFirstRadio:
```

```
        HandleRadio( kFirstRadio );
        break;
        .
        .
        .
    case iThirdCheckBox:
        HandleRadio( iThirdCheckBox );
        break;
    }
    DisposDialog( dialog );
```

Dialog items are either **enabled** or **disabled**. If an item is disabled, `ModalDialog()` will not report mouse clicks in the item. In general, clicking **ICONS** and **PICTs** in a dialog box has no special significance, so disable both of these types of items.

When the user clicks in an enabled item, call a routine to handle the click. When the user clicks either the **OK** or **Cancel** button, the dialog loop exits and the dialog window is disposed.

To change the state of a control, call `HiliteControl()`:

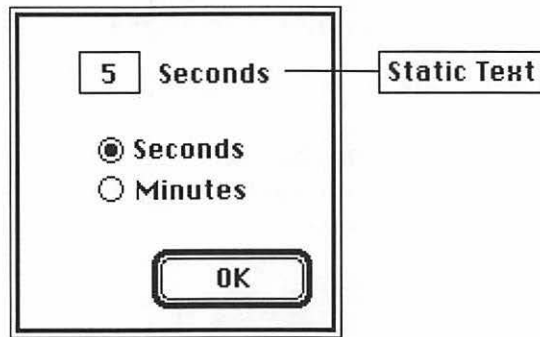
```
HiliteControl( ControlHandle theControl, short hiliteState );
```

To enable a control, call `HiliteControl()` with a `hiliteState` of 0. To disable the control, set `hiliteState` to 254.



Disabling a control gives the control a dimmed appearance. When you get to the program in this chapter, we'll call `HiliteControl()` to enable and disable the **OK** button in a dialog box. It's important to discriminate between the state of a control (whether it is enabled or disabled) and the value of the control.

**Static text** and **editable text** fields are also usually disabled, although you may change them in response to other events. For example, a timer might display the time in minutes or seconds, depending on the value of a set of radio buttons (Figure 6.16). If the **Seconds** radio button is clicked, the static text field could read **Seconds**. If the **Minutes** radio button is clicked, the static text field could be changed to read **Minutes**. Use the routines `GetIText()` and `SetIText()` to read and set the values of static text fields.



**Figure 6.16** Changing static text.

`ParamText()` allows you to create a set of four default strings that can be substituted in your static text fields. To specify them, call `ParamText()` with four `Str255s`:

```
ParamText( "\pPink", "\pBelshazzar",
           "\pFuture hazy, try later", "\pAltarian dog biscuits" );
```

From now on, whenever the strings `^0`, `^1`, `^2`, or `^3` appear in a static text item, they will be replaced by the appropriate `ParamText()` parameter. `ParamText()` is used in Chapter 7's error-handling routines.



You can store `ParamText()` strings in your resource file as resources of type 'STR' or inside a single 'STR#' resource, then read the strings in with `GetResource()` or `GetString()`, and finally, pass them to `ParamText()`. If, during the course of running your program, you decide to change the values of your strings, you can write them back out to the resource file with `WriteResource()`. This is a little tricky, but it gives you a great way to store program defaults. Look at Chapter 7's `ResWriter` project for an example on how that's done.

`GetIText()` and `SetIText()` can also be used to modify the contents of an editable text field. Here's an example:

```
GetDItem( dialog, kTextField, &itemType, &itemHandle,
          &itemRect );
GetIText( itemHandle, myString );
SetIText( itemHandle, "\pI have been replaced!!!" );
```

First, `GetDItem()` is called to retrieve a handle to the item with item number `kTextField`. Next, `GetIText()` uses that handle to retrieve the current string, storing it in the `Str255` variable `myString`. Finally, a new text string is copied to the item. It's important to note that you don't have to call `GetIText()` before you call `SetIText()`.



The last three arguments to `GetDItem()` are placeholders. That is, they won't always be used, but you always need to provide a variable to receive the values returned. In the previous example, `itemHandle` was used, but `itemType` and `itemRect` were not.

Like `ICONS` and `PICTS`, editable and static text items should be disabled so that mouse clicks are not reported. In the case of editable text fields, the dialog manager handles the mouse click for you.

Now that you've got the scoop on dialogs, let's take a look at alerts.

---

## Working with Alerts

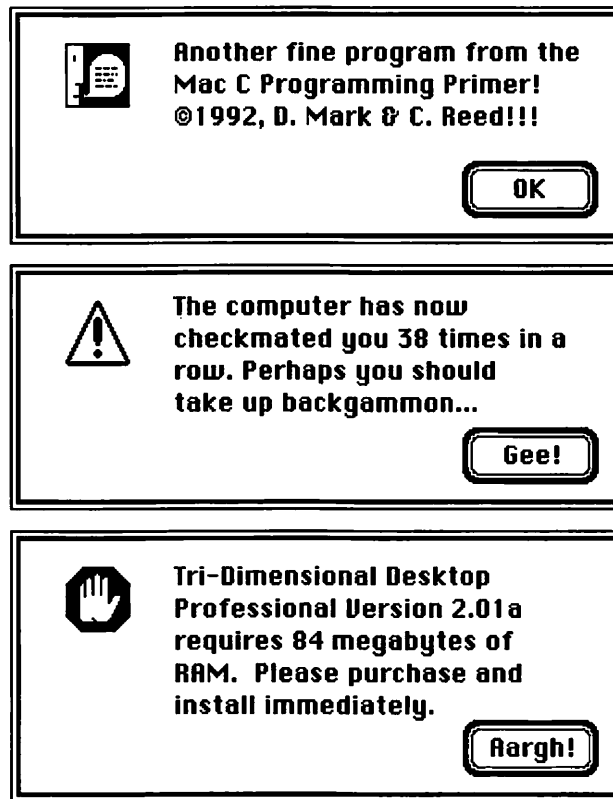
---

Alerts are very much like dialogs: You build them using `ResEdit`, and they consist of a window and a dialog item list. However, alerts are self-contained and can be invoked with a single line of code. Whereas `ModalDialog()` is called repeatedly inside a loop, the alert procedures are called once. Each alert routine takes care of its own housekeeping.

There are three standard types of alerts: note alerts, caution alerts, and stop alerts (Figure 6.17). **Note alerts** have an informative tone and are an easy way to tell the user something. **Caution alerts** tell the user that the next step taken should be considered carefully, as it may lead to unexpected results. **Stop alerts** indicate a critical situation, such as a fatal error, that must be brought to the user's attention.

Each alert exists in stages. The first time an alert is presented, it is a stage 1 alert; the second time, a stage 2 alert; the third time, a stage 3 alert; the fourth and subsequent times, a stage 4 alert. You can design your alerts so that stage 1 alerts are silent but stage 2, 3, and 4 alerts beep when the alert is presented. You can also specify whether or not the alert is presented at different stages.





**Figure 6.17** Note, caution, and stop alerts.

## The Alert Algorithm

Build your alert with ResEdit by creating an ALRT and a DITL resource. Unlike regular dialogs, the only type of control that belongs in an alert DITL is a button. The alert algorithm is as follows:

- Load and present the alert with a call to `StopAlert()`, `NoteAlert()`, or `CautionAlert()`.
- Use the value returned from each of these functions to determine which item was hit (that is, which button was pressed).

## Adding Alerts to Your Program

Unlike a dialog, an alert is implemented with a single call to the appropriate alert function. For example:

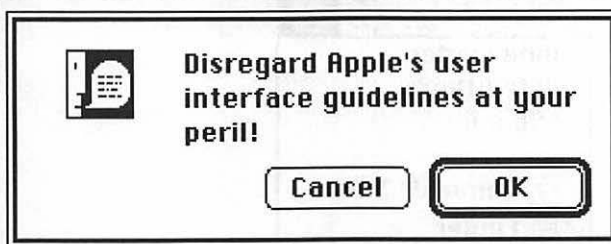
```
itemHit = StopAlert( kAlertID , nil );
```

`StopAlert()`, `NoteAlert()`, and `CautionAlert()` each take the same parameters and return the same thing, a short indicating the number of the item clicked on to dismiss the alert. If the alert offered a choice of three buttons, for example, the return value would indicate which button was clicked.

The first parameter to each of the three alert routines is the `ALRT` resource ID. The second parameter is an optional pointer to a filter procedure. Typically, you'll pass `nil` as the second parameter.



When you go about making your own dialogs or alerts, you should be familiar with Apple's user interface guidelines, which contain specific suggestions for placing and naming (among other things) dialog items. Place dialog and alert elements as shown in Figure 6.18.



**Figure 6.18** Proper dialog item placement.

In a properly designed dialog, the **OK** button is in the lower right corner. If a **Cancel** button exists, place it to the left of the **OK** button. Everything you need to properly design a dialog or alert is in *Inside Macintosh*, Volume VI, Chapter 2. Pay special attention to Figure 2-20 on page 2-29.

Though you'll find lots of dialogs and alerts that don't follow the guidelines, proceed at your own risk—the User Interface Police know where you live... .

That's about it for alerts. Time to look at the next manager in this chapter: the Notification Manager.

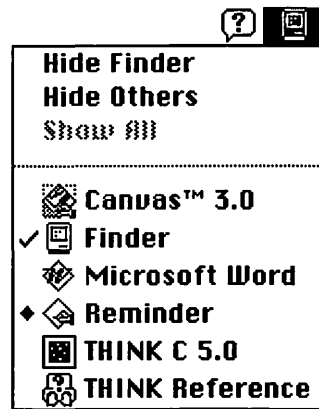
## The Notification Manager

The Notification Manager contains calls that allow applications running in the background to communicate with the user.

### How the Notification Manager Works

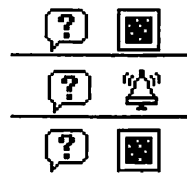
The Notification Manager alerts the user that an application running in the background requires the user's attention. There are five standard ways your application can signal the user:

1. You can place a small diamond-shaped mark (◆) next to the notifying application's item in the current application menu. The current application menu is located on the far right of the menu bar (Figure 6.19).






**Figure 6.19** (◆) mark beside the notifying application.

2. You can rotate the current application menu's icon with another icon. Figure 6.20 shows how a notification using a bell-shaped icon would look if THINK C was the current application.



**Figure 6.20** Rotating icons in the Notification Manager (THINK C is in the foreground).



In System 6, applications are listed under the desk accessories in the  menu. Therefore, the (♦) mark shows up under the  menu. Also, notification icons rotate with the  icon, not the application icon.

3. You can play a system alert sound, or a sound from a 'snd ' resource.
4. You can display an alert box with a short message.
5. You can execute a **response procedure** of your own choosing.

The PrintMonitor application that comes with the Mac's system software offers a good demonstration of these techniques. If the PrintMonitor encounters an error while trying to print a document, it notifies you using the Notification manager.



If you don't use System 7, or System 6 with MultiFinder, the Notification Manager's functionalism will be limited, as you cannot run your application in the background.

## Using the Notification Manager

To create a notification, you'll fill in a notification request data structure (we'll get to the data structure later in this section). Use the request to specify how the user should be notified, using some combination of the five mechanisms described above.

Once the request is filled in, pass it to the Notification Manager by calling `NMInstall()`:

```
OsErr NMInstall( QElemPtr nmReqPtr );
```

`nmReqPtr` is a pointer to your notification request. `NMInstall()` places your request on the **notification queue**. Once your request is in the notification queue, it becomes fair game for the Mac operating system (Mac OS). The Mac OS constantly scans the notification queue. Once it finds a request, it notifies the user in the manner indicated.

Once the user is notified, use `NMRemove()` to remove the notification from the Notification queue:

```
OsErr NMRemove ( QElemPtr nmReqPtr );
```

Before you call `NMRemove()`, you must make a decision. Do you just want to pass a message on to the user, or do you want the user to bring your application to the foreground?

To pass a message on to the user, create a notification with an alert containing your message. When you set up the notification data structure (described below), mark it as auto-removing. This tells the Notification Manager to automatically call `NMRemove()` for you as soon as the notification completes.

To ask the user to bring the notifying application to the foreground, use a rotating icon and a ♦ mark in the application menu, in addition to any other mechanisms you'd like. In your application, you'll call `NMRemove()` as soon as you receive a resume event. At this point, your application is in the foreground and can interact with the user.

Both `NMInstall()` and `NMRemove()` make use of a `QElemPtr`, a pointer to a notification data structure. The next section details that structure.

## The Notification Manager Structure

Each call to the Notification Manager makes use of the `NMRec` data structure:

```
typedef struct NMRec
{
    QElemPtr    qLink        /* the next queue entry */
    short       qType        /* queue type */
    short       nmFlags      /* reserved */
    long        nmPrivate     /* reserved */
    short       nmReserved   /* reserved */
    short       nmMark       /* Application ID to mark in
                             ⌘ menu */
    Handle      nmIcon       /* handle to small icon */
    Handle      nmSound      /* handle to sound record */
    StringPtr   nmStr        /* string to appear in
                             alert */
    ProcPtr     nmResp       /* pointer to response
                             routine */
    long        nmRefCon     /* for application use */
} NMRec;
```

Here's an explanation of the NMRec fields:

- qLink, qType, nmFlags, nmPrivate, and nmReserved are either reserved or contain information about the notification queue; you won't adjust these values.
- nmMark: If nmMark is 0, the (♦) will not be displayed in the application menu when the notification occurs; if nmMark is 1, the application that is making the notifying call receives the mark. If you want a desk accessory to be marked, use the refnum of the desk accessory. Drivers should pass 0.
- nmIcon: If nmIcon is nil, no icon is used; otherwise, a handle to the small icon ('SICN' resource) to be used should be placed here. The handle does not need to be locked, but must be nonpurgeable. Do this by making sure the 'SICN' resource's **Purgeable**: check box is unchecked.
- nmSound: if nmSound is 0, no sound is played; -1 will result in the system sound being played. To play an 'snd ' sound resource, put a handle to the resource here. As with the small icon, the 'snd ' resource handle need not be locked, but it must be nonpurgeable.
- nmStr contains the pointer to the text string to be used in the alert box. Set nmStr to nil for no alert box. Do not dispose of the string until the notification is removed.
- nmResp is a pointer to a response procedure that gets called once the notification is complete (VI:24-9). If you set nmResp to -1L, the request is automatically removed from the notification queue once the notification is complete.

You'll see the Notification Manager in action in the Reminder program later in this chapter. For more information on the Notification Manager, see Chapter 26 in Volume VI of *Inside Macintosh*.

---

## The Process Manager

---

The Process Manager is responsible for keeping track of the applications (or **processes**) that are currently running. The part of the Process Manager we're interested in is the LaunchApplication() routine, which lets you start up another application from within your own program. You'll see an example of this in Reminder, later in this chapter.

Here's the calling sequence for LaunchApplication():

```
OsErr LaunchApplication( LaunchPBPtr LaunchParams );
```

`LaunchApplication()` describes a specific application to the Process Manager, which then launches it. `LaunchParams` is a pointer to a data structure of type `LaunchParamBlockRec`, which specifies the application to be launched:

```
struct LaunchParamBlockRec
{
    unsigned long    reserved1;           /* reserved */
    unsigned short   reserved2;           /* reserved */
    unsigned short   launchBlockID;        /* extended
                                         block */
    unsigned long    launchEPBLength;      /* length of
                                         block */
    unsigned short   launchFileFlags;      /* Finder flags
                                         of
                                         application */
    LaunchFlags      launchControlFlags;   /* launch option */
    FSSpecPtr        launchAppSpec;        /* location of
                                         application */
    ProcessSerialNumber launchProcessSN;    /* returned psn */
    unsigned long    launchPreferredSize;  /* launch preferred
                                         size */
    unsigned long    launchMinimumSize;    /* launch minimum
                                         size */
    unsigned long    launchAvailableSize;  /* launch available
                                         size */
    AppParametersPtr launchAppParameters; /* high-level
                                         event */
};
```

Here's an explanation of the `LaunchParamBlockRec` fields:

- `reserved1` and `nmPrivate` are reserved by Apple; don't adjust these values.
- `launchBlockID` and `launchEPBLength`: are used to describe the `LaunchParamBlockRec` structure to the Mac OS. Pass `extendedBlock` in `launchBlockID`, and `extendedBlockLen` in `launchEPBLength` for all notifications.
- `launchFileFlags`: returns the **Finder flags** from the application you have selected. The Finder flags refer to information the Finder uses to categorize an application; they are the same flags you set in the `EventTracker` program in Chapter 4. You'll learn more about the Finder flags in Chapter 8.
- `launchControlFlags`: you can specify how you want the application launched by passing the appropriate value. The constants you can pass are:

```
enum
{
    launchContinue = 0x4000,
    launchNoFileFlags = 0x0800,
    launchUseMinimum = 0x0400,
    launchDontSwitch = 0x0200,
    launchInhibitDaemon = 0x0080
};
```

`launchContinue` should be set if you want your application to continue executing after launching the specified application. `launchNoFileFlags` should be used if you want to pass your own Finder flag information to the application to be launched, instead of using the launching application's Finder flags. `launchUseMinimum` will try to use the minimum memory size recommended by the application you're trying to launch. `launchDontSwitch` launches the application in the background. Finally, `launchInhibitDaemon` should be set if you don't want to launch a background-only application (like the Backgrounder application in your System Folder).




- `launchAppSpec` tells `LaunchApplication()` the location of the application you are launching. It does that with a pointer to a structure of type `FSSpec`. For now, that's all you need to know about this structure; you'll learn more about the `FSSpec` structure in the `OpenPICT` program in Chapter 7.
- `launchProcessSN` returns a unique serial number for the application you are launching. You can use it to check the status of the application: for example, whether it's still running, or how much memory it's using.
- `launchPreferredSize` and `launchMinimumSize` return the specified application's preferred and minimum memory requirements.
- If there is not enough memory to launch the application, `launchAvailableSize` specifies how much memory *is* available.
- `launchAppParameters` allows you to specify the first high-level event sent to the launched application. If you pass `nil`, the standard `kAEOpenApplication` Apple event will be sent.

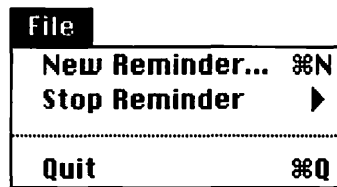
The next section lists and describes `Reminder`, the biggest and most complex program in this book. `Reminder` will show you how to put together all the pieces we've talked about so far: dialogs, alerts, events, menus, the Notification Manager, and the Process Manager.



## Reminder

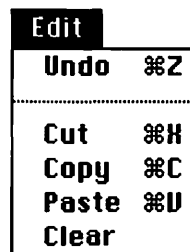
In this chapter, you looked at the Dialog Manager, the Notification Manager, and the Process Manager. Our next program, **Reminder**, combines dialogs, alerts, notifications, and application launching into a surprisingly powerful application. Reminder allows you to send messages to yourself at specific times. Each reminder consists of some combination of a text string, a sound, and a launched application. Reminder can keep track of multiple reminders, so you can send as many messages to yourself as you like.

As you saw in WorldClock, Reminder supports the standard , **File**, and **Edit** menus. The  menu is fully functional, giving you access to all the items you normally expect to see in the  menu. The **File** menu (Figure 6.21) contains three items: **New Reminder**, which allows you to create a new reminder, **Stop Reminder**, which allows you to cancel reminders by selecting them from the attached hierarchical menu, and **Quit**.



**Figure 6.21** Reminder's **File** menu.

Reminder's **Edit** menu contains the minimum standard set of **Edit** commands (Figure 6.22).



**Figure 6.22** Reminder's **Edit** menu.

## The Reminder Algorithm

Reminder uses the same event-loop structure presented in the last few chapters. Here's how Reminder works:

- Initialize the Toolbox.
- Initialize Reminder's menus.
- Enter the event loop, checking once a second to see if any reminders need to be displayed.
- If a reminder's time has come up, put the reminder into the Notification Manager's queue.

---

## Resources

---

Create a folder called `Reminder` inside your `Development` folder. Next, launch `ResEdit` and click the mouse to bring up the **Open file** dialog box. Click on the **New** button. When the **New file** dialog box appears, navigate into the `Reminder` folder and create a resource file named `Reminder.π.rsrc` inside the `Reminder` folder.

## Creating the MBAR Resource

Select **Create New Resource** from `ResEdit`'s **Resource** menu. When prompted for a resource type, type `MBAR`, then click the **OK** button. An `MBAR` editing window will appear. We'll need to add three menus to this menu bar, one for each menu title that appears in the menu bar itself: **⌘**, **File**, and **Edit**. As you saw in `WorldClock`, the hierarchical menu **Reminder** is not part of the `MBAR` resource.

As you did in `WorldClock`, click on the row of asterisks (\*) that appear in the `MBAR` window and select **Insert New Field(s)** from the **Resource** menu to add each field. Add the three menu resource IDs shown in Figure 6.23 to the `MBAR` resource.

Click on the close box of the `MBAR` editing window. Then click on the close box of the `MBAR` window. You should be back to the `Reminder.π.rsrc` window.

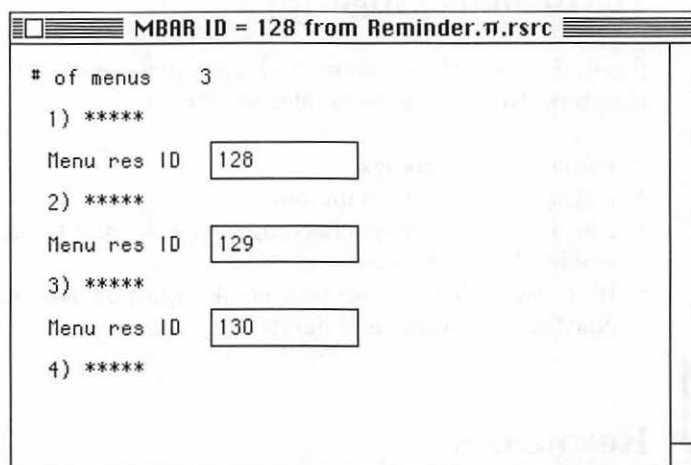



Figure 6.23 Completed MBAR resource.

## Creating the MENU Resources

Next, let's create the four MENU resources that Reminder uses. Select **Create New Resource** from ResEdit's **Resource** menu. When prompted for a resource type, enter MENU and click the **OK** button. A MENU editing window will appear, similar to the one in Figure 6.24. We'll start by editing the  MENU.

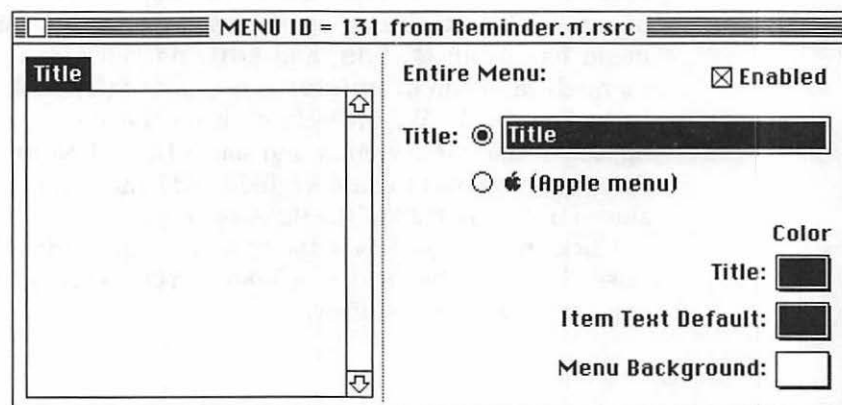




Figure 6.24 A new MENU editing window.

Click on the  (Apple menu) radio button on the right side of the window. Notice that the title of the menu on the left side of the window changed to the  character. Now, hit the return key. The

editor will move on to the first item in this **MENU**. Notice that the field labeled **Title:** has changed to **Text:**. Click in the **Text:** field and type the text **About Reminder...**

Notice that the **Enabled** check box has been checked for you. This makes the item selectable. If the **Enabled** check box was not checked, the item would appear dimmed in the menu and would not be selectable.

Hit the return key to move to the next item. Click on the **(separator line)** radio button to turn this second item into a separator line. Notice that the **Enabled** check box is not checked. This means that the separator will not be selectable. This is normal for separator lines. Leave the **Enabled** check box unchecked for this item. Your **MENU** should look like Figure 6.25.

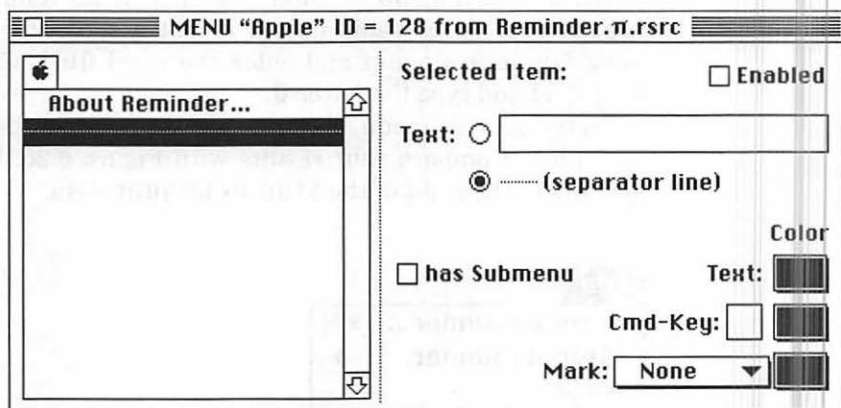


Figure 6.25 The completed **Apple** **MENU**.

Next, select **Get Resource Info** from the **Resource** menu. Make sure the **ID:** field says 128. Close the **Resource Info** window. Now select **Edit Menu & MDEF ID...** from the **MENU** menu. Make sure the **Menu ID:** field says 128 and the **MDEF ID:** field says 0, then click the **OK** button.



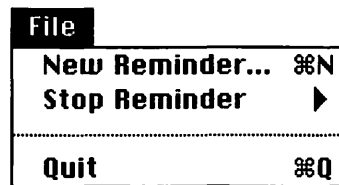
It's critical to make sure that your **MENU**'s resource ID and the value in the **Menu ID:** field in the **Edit Menu & MDEF ID...** dialog box agree. Make sure you check this for each one of Reminder's **MENU** resources.

Close the MENU editing window. You should see the MENU picker window (the window listing each of the MENU resources in this file). Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Without clicking the mouse, type the word **File**. Notice that your text appears in the **Title:** field. Hit a return and type the words **New Reminder...** without hitting a return afterward.

Click the mouse in the **Cmd-Key:** field and type the letter **N**. This tells the Menu Manager to associate the command-key sequence **⌘N** with this item. Hit a return and type the words **Stop Reminder** without hitting a return afterward. Click the mouse in the **has Submenu** check box. Then, click the mouse in the **ID:** field that appears and type in the number **103**. This tells the Menu Manager to associate MENU 103 with this item. When this item is selected, the MENU with resource ID 103 will appear as a hierarchical submenu.

Hit a return again to enter the next menu item: just click in the **(separator line)** radio button to put a separator line in the third item. Hit return again and enter the word **Quit**; Click in the **Cmd-Key** field and type the letter **Q**.

To try out this menu, click on the **File** menu on the *right* side of the menu bar. Compare your results with Figure 6.26. Notice the **▶** that appears to the right of the **Stop Reminder** item.



**Figure 6.26** Testing the **File** menu in ResEdit.

Finally, make sure the resource ID for this MENU is set to 129. Check this using both the **Get Resource Info** and the **Edit Menu & MDEF ID...** menu items.

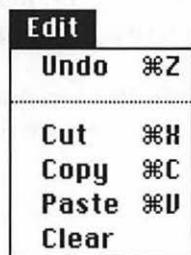
Close the MENU editing window. You should see the MENU picker window (this time it should show two MENUS). The next MENU resource you'll enter is the **Edit** menu resource. If you have already built the **Edit** MENU for Chapter 5's WorldClock project, open `WorldClock.π.rsrc` and copy MENU resource 130 into this project. Otherwise, just follow along for the next four paragraphs.

Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Enter **Edit** in the **Title:** field.

Hit a return and type the word **Undo**. Before you hit a return, type the letter **Z** in the **Cmd-Key** field. Hit a return and click on the **(separator line)** radio button. The **Enabled** check box should *not* be checked for this item.

Hit a return and type the word **Cut**. Type the letter **H** in the **Cmd-Key** field. Hit a return and type the word **Copy**. Type the letter **C** in the **Cmd-Key** field. Hit a return and type the word **Paste**. Type the letter **V** in the **Cmd-Key** field. Hit one last return and type the word **Clear**. **Clear** doesn't have a command-key equivalent.

To try out this menu, click on the **Edit** menu on the *right* side of the menu bar. Compare your results with Figure 6.27.

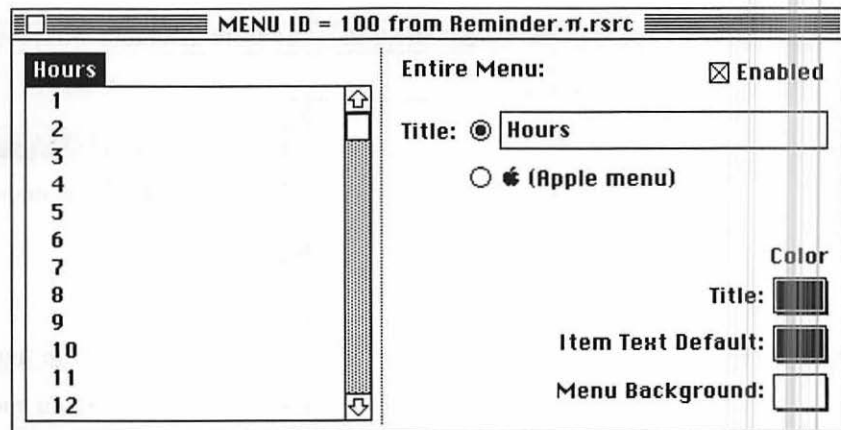


**Figure 6.27** Testing the **Edit** menu in ResEdit.

Make sure the resource ID for this **MENU** is set to 130. Check this with both the **Get Resource Info** and the **Edit Menu & MDEF ID...** menu items.

Close the **MENU** editing window. You should see the **MENU** picker window (this time it should show three **MENUs**). Select **Create New Resource** from the **Resource** menu. A new **MENU** editing window will appear. Enter **Hours** in the **Title:** field.

Hit a return and type the word **Hours**. Then, enter the menu items as shown in Figure 6.28.



**Figure 6.28** The **Hours** menu in ResEdit (elongated for clarity).

To try out this menu, click on the **Hours** menu on the right side of the menu bar. Compare your results with Figure 6.29.

Make sure the resource ID for the **Hours** MENU is set to 100. Check this through both the **Get Resource Info** and the **Edit Menu & MDEF ID...** menu items.

Close the **Hours** MENU editing window. You should see the MENU picker window (this time it should show four MENUS). Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Enter **Minutes** in the **Title:** field.

Hit a return and type the word **Minutes**. Then add the items to the **Minutes** MENU as shown in Figure 6.30.

To try out this menu, click on the **Minutes** menu on the right side of the menu bar. Compare your results with Figure 6.31.

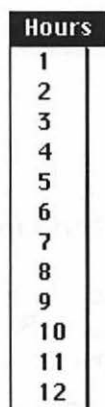


Figure 6.29 Testing the **Hours** menu in ResEdit.

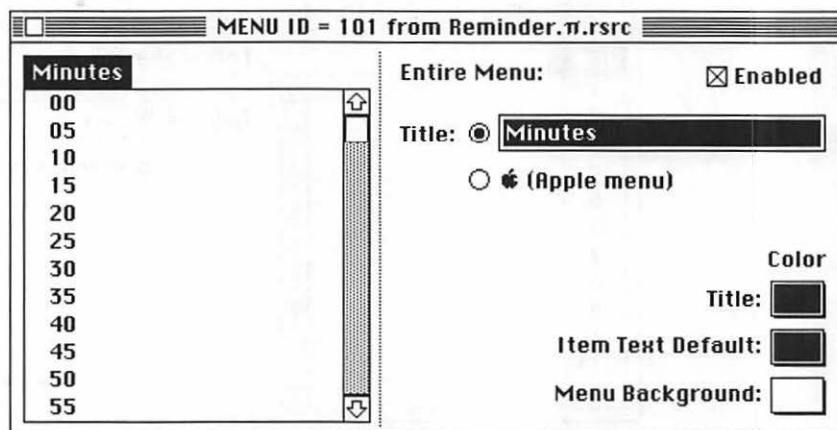
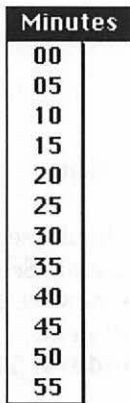


Figure 6.30 The **Minutes** menu in ResEdit (elongated for clarity).

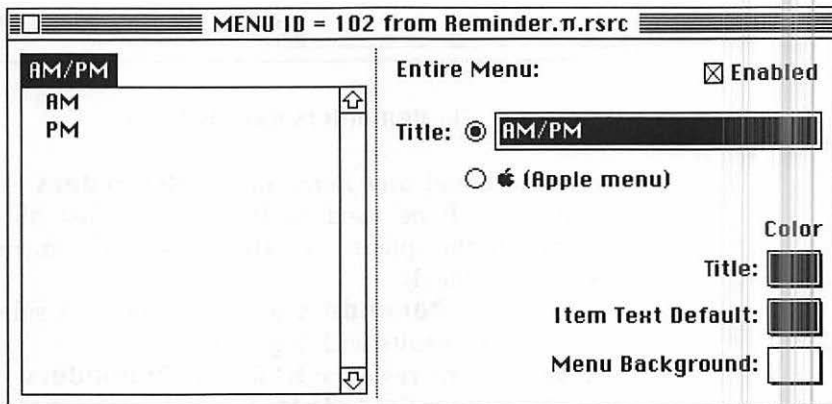


**Figure 6.31** Testing the **Minutes** menu in ResEdit.

Close the **Minutes** MENU editing window. You should see the MENU picker window (this time it should show five MENUS). Select **Create New Resource** from the **Resource** menu. A new MENU editing window will appear. Enter **AM/PM** in the **Title:** field.

Hit a return and type **AM/PM**. Then add the items to the **AM/PM** MENU as shown in Figure 6.32.

To try out this menu, click on the **AM/PM** menu on the right side of the menu bar. Compare your results with Figure 6.33.



**Figure 6.32** The **AM/PM** menu in ResEdit (elongated for clarity).

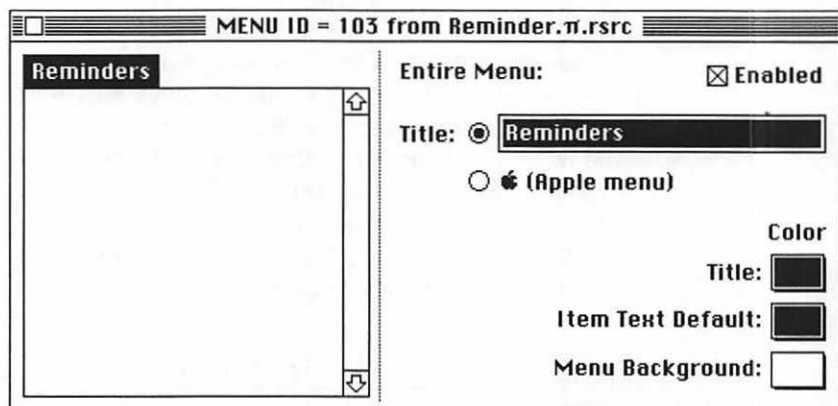




**Figure 6.33** Testing the **AM/PM** menu in ResEdit.

Close the **MENU** editing window. You should see the **MENU** picker window (this time it should show six **MENUS**). Select **Create New Resource** from the **Resource** menu. A new **MENU** editing window will appear. Enter **Reminders** in the **Title:** field.

Hit a return and type the word **Reminders**. The window should look like Figure 6.34.



**Figure 6.34** The **Reminders** menu in ResEdit.

You won't add any items to the **Reminders** **MENU** at this point. This menu will be used in **Reminder** to list all of the reminders currently in the queue. For this reason, the menu will have to be generated on the fly.

Click on the **Reminders** menu on the right side of the menu bar. Compare your results with Figure 6.35.

Make sure the resource ID for the **Reminders** **MENU** is set to 103. Select **Get Resource Info** from the **Resource** menu and **Edit Menu & MDEF ID...** from the **MENU** menu.

Close the **Reminders** **MENU** editing window. You should see the **MENU** picker window (this time it should show seven **MENUS**). Finally! You're done with **MENUS**!

## Reminders

**Figure 6.35** Testing the **Reminders** menu in ResEdit.

### Creating Pop-up Menu CNTL Resources

WorldClock featured a pop-up menu control that allowed you to find out the time in four different cities. This time you'll create three pop-up menus that allow you to specify the exact time of a reminder in hours, minutes, and A.M./P.M.

Back in ResEdit, close the MENU window and the MENU picker window, leaving only the window labeled `Reminder.π.rsrc`. Select **Create New Resource** from the **Resource** menu. Specify the resource type CNTL and click on the **OK** button. A CNTL editing window will appear. Fill in the CNTL fields exactly as specified in Figure 6.36.

The **BoundsRect** field specifies the top, left, bottom, and right of the pop-up menu rectangle. This control does not have a title, so just leave the **Value** field, which specifies how the title should be drawn, at 0. The **Visible** field determines whether the control is initially visible or hidden, just as it does in the WIND resource. The **Max** field specifies the portion of the **BoundsRect** to be allocated to the pop-up title. As there is no title for this popup, **Max** is set to 0 and the **Title** field should be left blank. For **Min**, enter the resource ID for the MENU to be used: 100. For **ProcID**, enter the procID for a pop-up menu: 1008. You won't be using the **RefCon**, so set it to 0.

**Figure 6.36** Specifications for the **Hours** pop-up CNTL.

Finally, select **Get Resource Info** from the **Resource** menu and set the CNTL's resource ID to 128. When you're done, close the resource info window and the CNTL editing window. Two more CNTLs to go!

Select **Create New Resource** from the **Resource** menu again. Specify the resource type CNTL and click on the **OK** button. Another CNTL editing window will appear. Fill in the CNTL fields exactly as specified in Figure 6.37. Next, set the CNTL's resource ID to 129. Close the CNTL windows.

The screenshot shows a Macintosh-style dialog box titled "CNTL ID = 129 from Reminder.π.rsrc". It contains the following fields and controls:

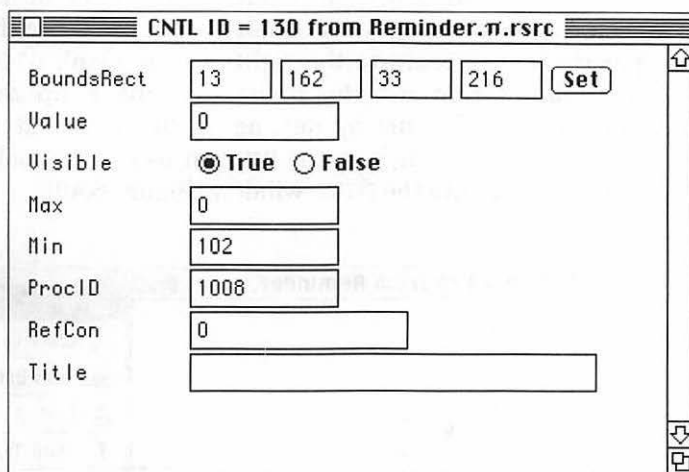
- BoundsRect:** Four text input fields containing the values 13, 111, 33, and 161. A "Set" button is located to the right of these fields.
- Value:** A text input field containing the value 0.
- Visible:** Two radio buttons, "True" (which is selected) and "False".
- Max:** A text input field containing the value 0.
- Min:** A text input field containing the value 101.
- ProcID:** A text input field containing the value 1008.
- RefCon:** A text input field containing the value 0.
- Title:** A large empty text input field.

Standard Macintosh window controls (close, zoom, scroll) are visible on the right side of the dialog.

**Figure 6.37** Specifications for the **Minutes** pop-up CNTL.

Select **Create New Resource** from the **Resource** menu one more time. Specify the resource type CNTL and click on the **OK** button. The final CNTL editing window will appear. Fill in the CNTL fields as shown in Figure 6.38. Next, set the CNTL's resource ID to 130.

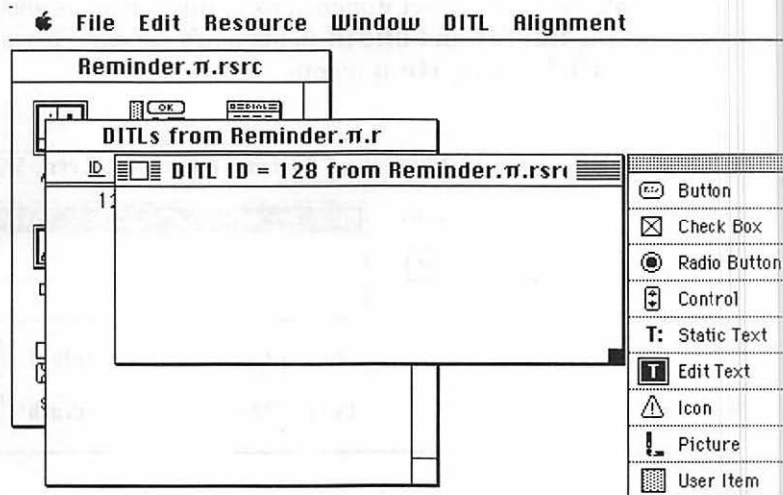
Close the CNTL window and the CNTL picker window, leaving only the window labeled `Reminder.π.rsrc`.



**Figure 6.38** Specifications for the AM/PM pop-up CNTL.

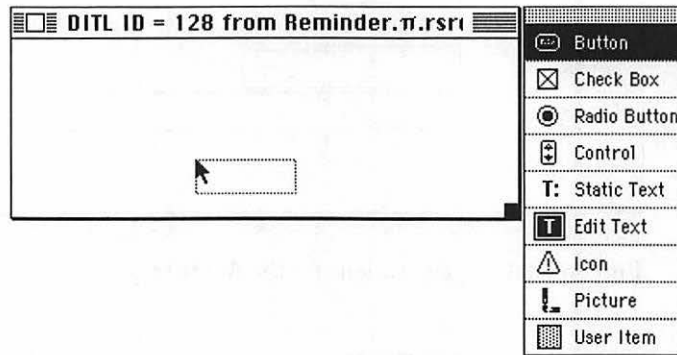
## Creating a DITL Resource

The next resources you'll create are the DITL resources that define dialog items for dialogs. Select **Create New Resource** from the **Resource** menu. When prompted for a resource type, enter DITL and click the **OK** button. The DITL editing window that appears should look something like Figure 6.39.



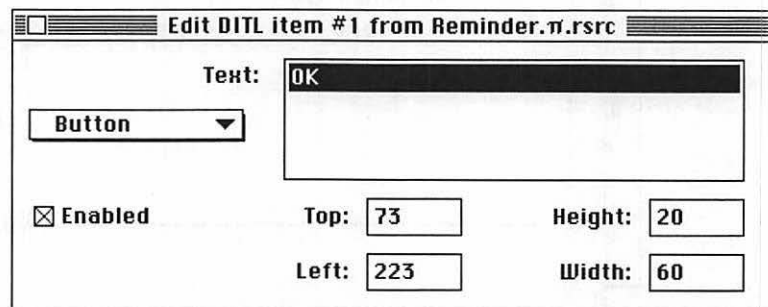
**Figure 6.39** DITL editing window.

The DITL editor is a bit different from other ResEdit editors you've seen before. To create a new dialog item, you'll use the dialog item tool palette that appears to the right of your blank DITL. The first DITL you'll build contains the items that make up the About box for Reminder. Let's start by putting an **OK** button at the bottom of the DITL. To do this, click on the **Button** tool in the tool palette, dragging a new button into the DITL window (Figure 6.40).



**Figure 6.40** Putting a button into the DITL.

When you release the mouse button, the button dimensions are displayed as a moving marquee (sometimes known as the “marching ants”). To name and accurately locate the button, double-click on it. The Edit DITL item #1 window will appear. Enter **OK** in the **Text:** field and the correct dimensions of the button as shown in Figure 6.41. If the **Height** and **Width** fields don't appear, select **Show Height & Width** from the **Item** menu.



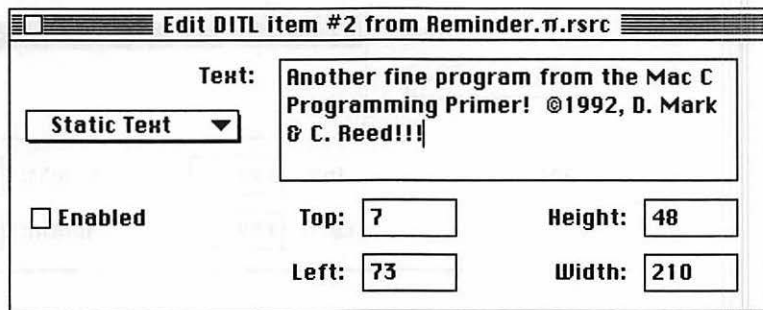
**Figure 6.41** Putting a button into the DITL.

Click in the close box to close the Edit DITL item #1 window. Next, add a static text item to your DITL by dragging it from the **Static Text** tool in the palette window to the middle of the DITL window, as shown in Figure 6.42.



**Figure 6.42** Adding a static text item to the DITL.

As you did with the **OK** button, double-click in the **Static Text** item to set up its size and text, as shown in Figure 6.43.



**Figure 6.43** Specifications for the static text item.

Close the Edit DITL item #2 window by clicking in the close box. If you entered the information correctly, your DITL should look like Figure 6.44. If the DITL window isn't quite the same shape as the one shown in Figure 6.44, don't worry. If you like, grab the small grow box in the bottom right corner, stretching the DITL window to a more workable size. Remember, the dialog and alert window information is provided by the DLOG and ALRT resources, not by the DITL.

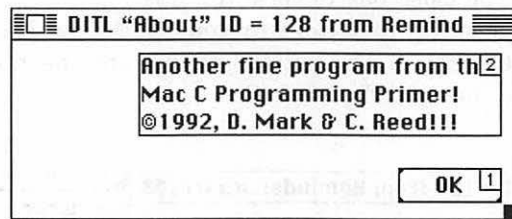


Figure 6.44 Your first DITL.

Finally, select **Get Resource Info** from the **Resource** menu and set the DITL's resource ID to 128 and its name to **About**. When you're done, close the resource info and DITL editing windows, leaving only the DITL picker window.

Now that you've done the About box DITL, let's do the DITL that is shown when you want to create a new reminder. Select **Create New Resource** from the **Resource** menu. The next DITL editing window will appear. This DITL will consist of 12 items. The first one is the **OK** button: Drag a button from the button tool, then double-click on it to set it to the specifications in Figure 6.45.

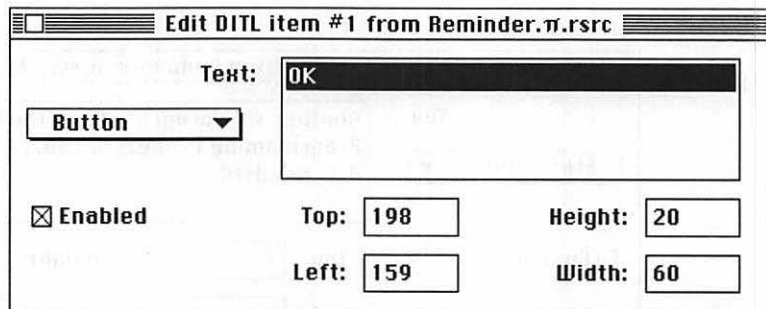


Figure 6.45 OK button specifications.

The next item is the **Cancel** button: Drag a button from the button tool, then double-click on it to set it to the specifications in Figure 6.46.

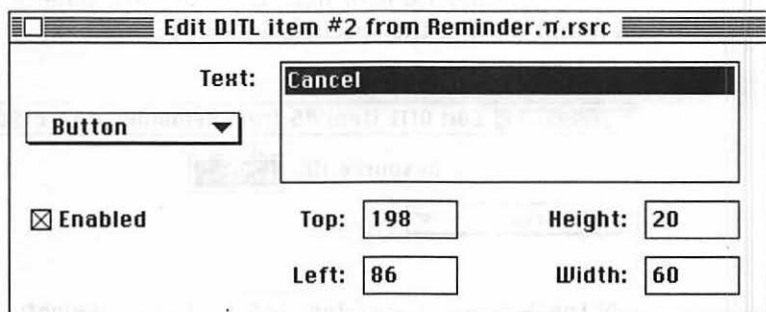


Figure 6.46 Cancel button specifications.

Next, some static text: Drag a field from the static text tool, then double-click on it to set it to the specifications in Figure 6.47.

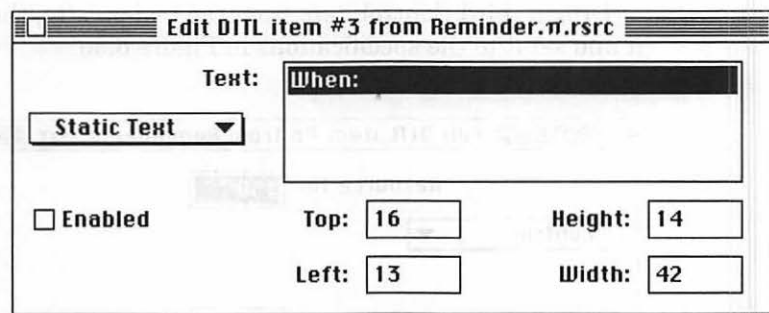


Figure 6.47 DITL item #3.

Now to place the three pop-up menu controls that we created earlier. Drag a Control item from the tool palette, then double-click on it and set to the specifications in Figure 6.48.

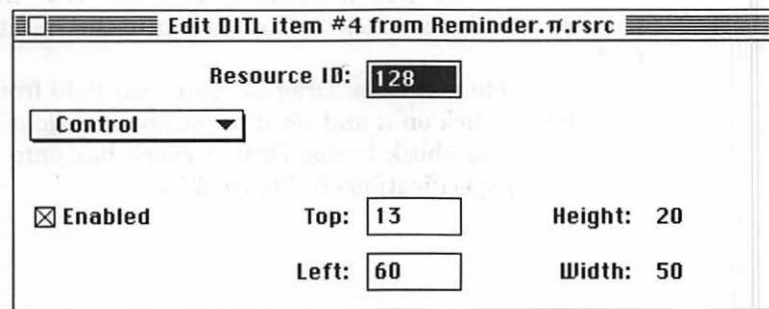
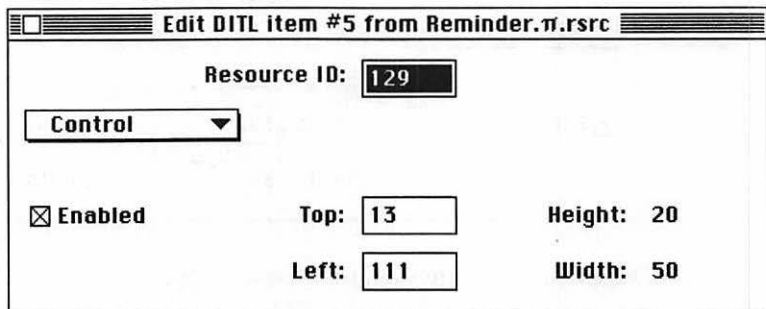


Figure 6.48 DITL item #4, the first popup.

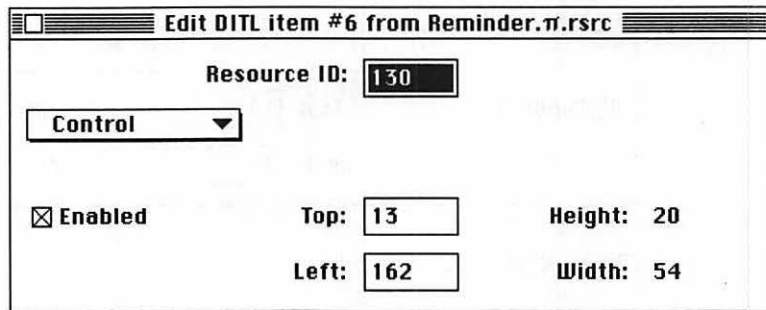


Drag another Control item from the tool palette, then double-click on it and set it to the specifications in Figure 6.49.



**Figure 6.49** DITL item #5, the second popup.

Drag a third Control item from the tool palette, then double-click on it and set it to the specifications in Figure 6.50.



**Figure 6.50** DITL item #6, the third popup.

Another static text item: Drag a Static Text field from the tool palette, then double-click on it and set it to the specifications in Figure 6.51.

An editable text item: Drag an Edit Text field from the tool palette, then double-click on it and set it to the specifications in Figure 6.52.

Now for the check boxes: Drag a check box onto your window and set it to the specifications in Figure 6.53.

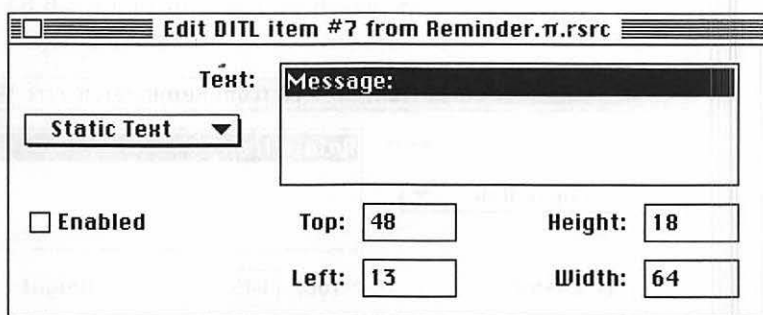


Figure 6.51 DITL item #7.

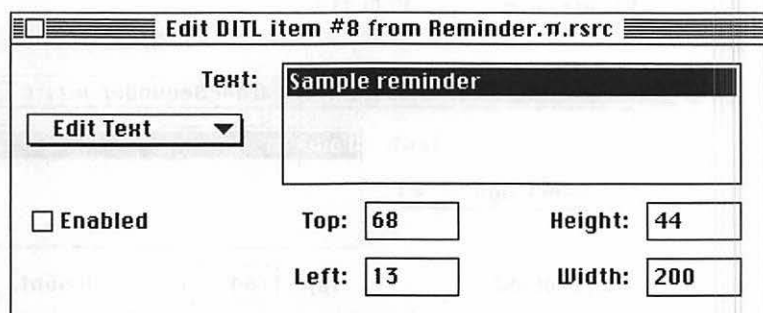


Figure 6.52 DITL item #8.

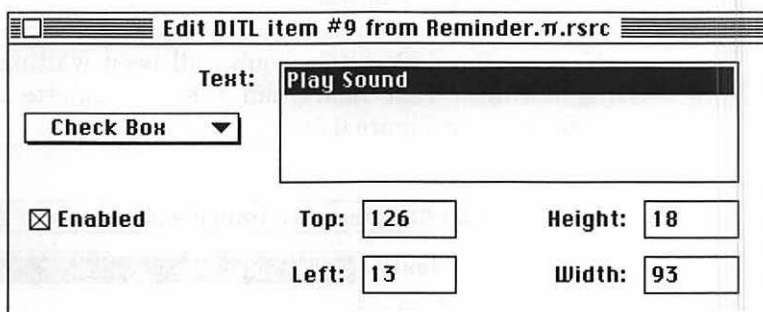


Figure 6.53 DITL item #9, the first check box.

Ditto for the next two check boxes in Figures 6.54 and 6.55.

Edit DITL item #10 from Reminder.rsrc

Text: Rotate Icon

Check Box

☒ Enabled

Top: 145 Height: 18

Left: 13 Width: 99

Figure 6.54 DITL item #10.

Edit DITL item #11 from Reminder.rsrc

Text: Launch:

Check Box

☒ Enabled

Top: 164 Height: 18

Left: 13 Width: 73

Figure 6.55 DITL item #11.

Now for the dialog item you've all been waiting for: the last one! Drag a Static Text field from the tool palette and set it to the specifications in Figure 6.56.

Edit DITL item #12 from Reminder.rsrc

Text: <None Selected>

Static Text

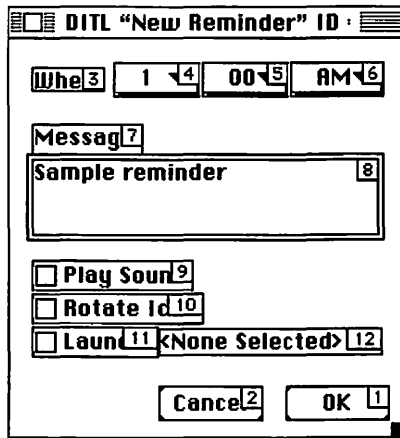
☒ Enabled

Top: 165 Height: 16

Left: 86 Width: 129

Figure 6.56 DITL item #12 (The last DITL item—whew!).

Your DITL and Figure 6.57 should look pretty much the same. If the pop-up menus didn't show up, check your CNTL resources and make sure they were input correctly. If any element seems out of place, you probably just dropped a digit; double-click on the recalcitrant element and check your figures.



**Figure 6.57** The Reminder DITL.

Finally, select **Get Resource Info** from the **Resource** menu and set the DITL's resource ID to 129 and its name to **New Reminder**. When you're done, close the resource info and DITL editing windows, leaving only the window labeled `Reminder.p.rsrc`.

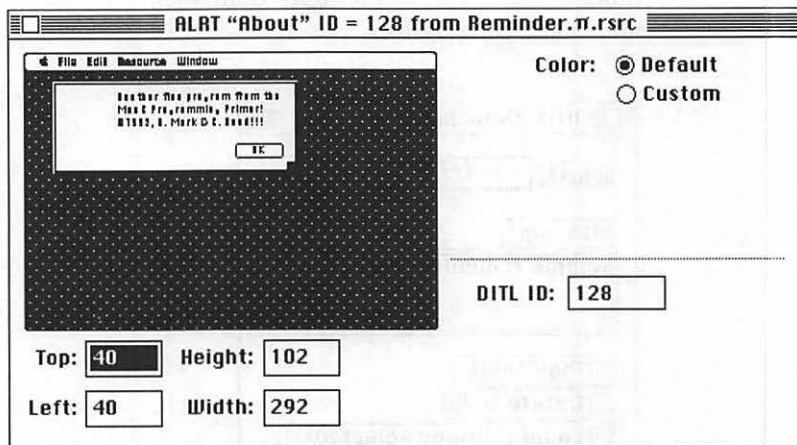
That's it for DITLs, but you still have a few more resources to set up. If you're feeling that ResEdit spelled backward looks like Satan, perhaps you should take a short break before continuing. Whether you stop at this point or not, select **Save** from the **File** menu in ResEdit to save all the work you just did.

At this point, you've created two DITLs, seven MENUs, one MBAR and three CNTLs: doughty work from doughty coders. Let's forge ahead and add the resources that the DITLs are created for: the ALRT and DLOG resources.

## Creating an ALRT Resource

The next resource you'll create is the ALRT resource, which acts as a template for alerts, much like WIND resources do for windows. Select **Create New Resource** from the **Resource** menu. When prompted for a resource type, enter ALRT and click the **OK** button. An ALRT editing window will appear. Use the ALRT specifications in Figure 6.58 to customize your ALRT. Make sure you enter 128 in the **DITL ID:** field.

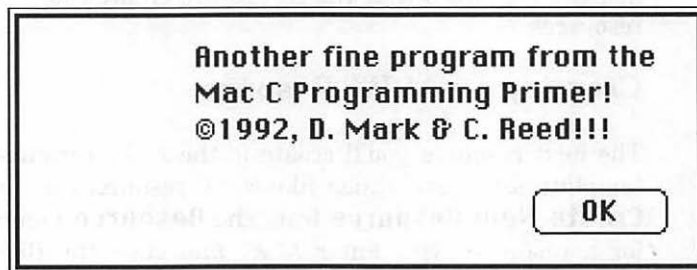
This links this ALRT with the items in DITL 128 which you created earlier.



**Figure 6.58** A new ALRT resource.

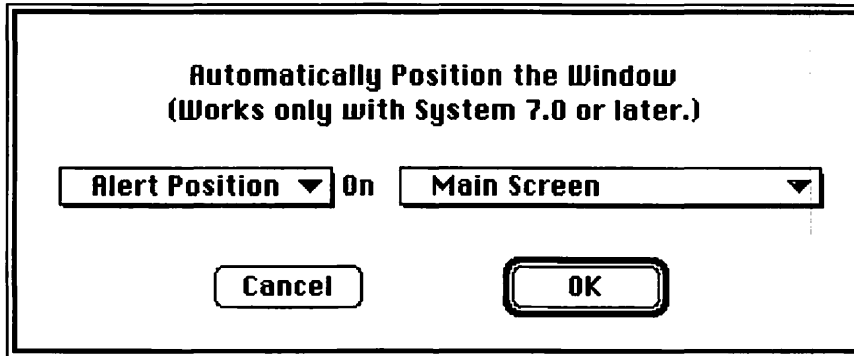
Select **Get Resource Info** from the **Resource** menu and set the ALRT's resource ID to 128 and its name to **About**. Close the **Resource Info** window.

To preview the alert, select the **Preview at Full Size** menu item in the **ALRT** menu. You should see the alert window as shown in Figure 6.59. As you've probably noticed, the alert icon didn't appear and the bold rounded rectangle never appeared around the **OK** button. The alert icon is filled in automatically by whichever routine you call to bring up the alert. A different icon will be drawn, depending on whether the alert is a note, caution, or stop alert. The ring around the **OK** button is drawn automatically by the Dialog Manager as soon as the alert is drawn on the screen. Click anywhere on the screen to dismiss the preview window.



**Figure 6.59** Preview of the **About Reminder...** alert.

One more thing to do before you finish the alert. System 7 allows you to automate the placement of all windows, including those created for dialogs and alerts. To use this feature, select **Auto Position...** from the **ALRT** menu. When the **Auto Position...** dialog box appears, use the pop-up menus to select **Alert Position** on the **Main Screen**, as shown in Figure 6.60. When Reminder brings up this alert, it will automatically be placed in the appropriate position on the screen.



**Figure 6.60** Auto-positioning your alert.

Click **OK** to dismiss the **Auto Position...** window. Then close the **ALRT** edit window and the **ALRT** picker window. Time to build the **DLOG** resource.

## Creating a DLOG Resource

The next resource you'll create is the **DLOG** resource, which acts as a template for the dialog window. All **ALRT** windows should be closed at this point, leaving only the window labeled `Reminder.π.rsrc`. Select **Create New Resource** from the **Resource** menu. When prompted for a resource type, enter **DLOG** and click the **OK** button. A **DLOG** editing window will appear. Before you do anything else, enter 129 in the **DITL ID:** field. This links the **DLOG** to the proper **DITL**. If you didn't do this, the **DLOG** would display the About alert's **DITL**. Use the **DLOG** specifications in Figure 6.61 to complete your **DLOG**. Make sure you select the eighth icon from the left in the **DLOG** editor's top row. This selects the standard dialog window type.

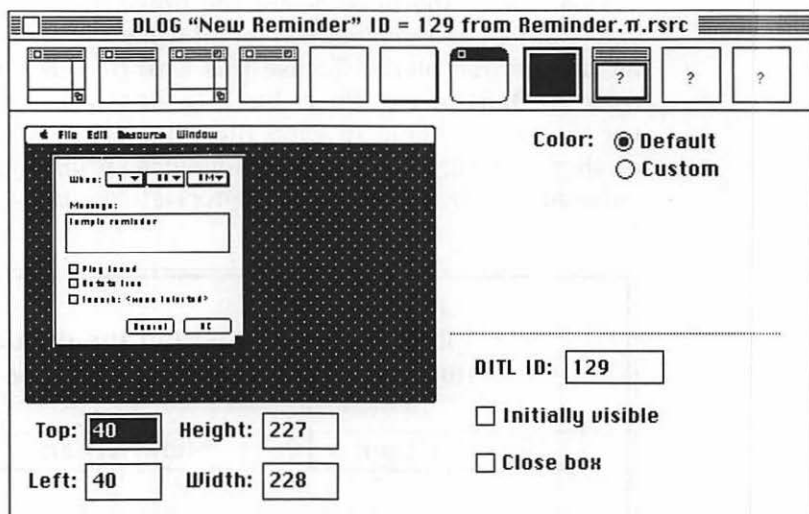


Figure 6.61 A new DLOG resource.

Select **Get Resource Info** from the **Resource** menu and set the DLOG's resource ID to 129 and its name to **New Reminder**. Close the **Resource Info** window.

To preview the dialog, select the **Preview at Full Size** menu item in the **DLOG** menu. You should see the alert window as shown in Figure 6.62. This time, the ring around the **OK** button gets drawn at run-time, once we call `SetDialogDefaultItem()`. Click anywhere on the screen to dismiss the preview window.

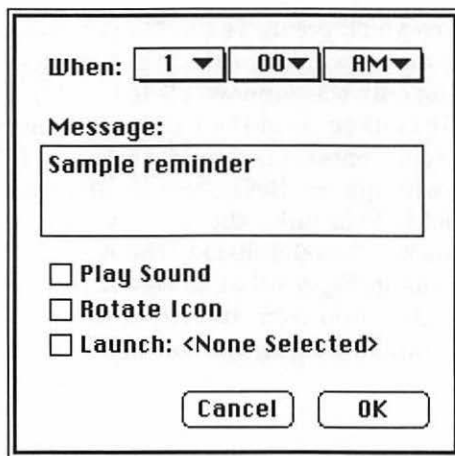
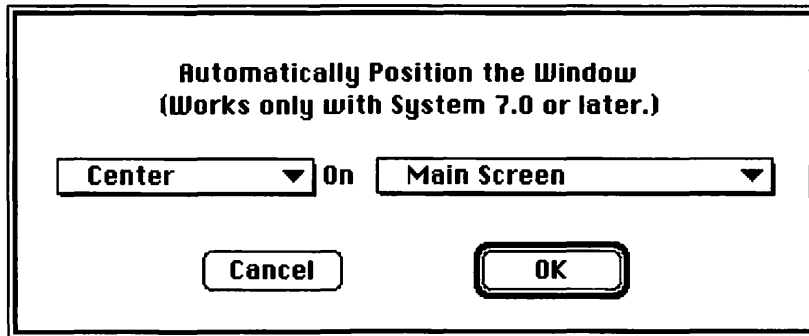


Figure 6.62 Preview of the **New Reminder** DLOG.

One more thing to do before you finish the DLOG. Just as you did with the alert, select **Auto Position...** from the **DLOG** menu. When the **Auto Position...** dialog box appears, use the pop-up menus to select **Center** on the **Main Screen**, as shown in Figure 6.63. When Reminder brings up this alert, it will automatically be placed in the appropriate position on the screen.



**Figure 6.63** Auto-positioning your dialog.

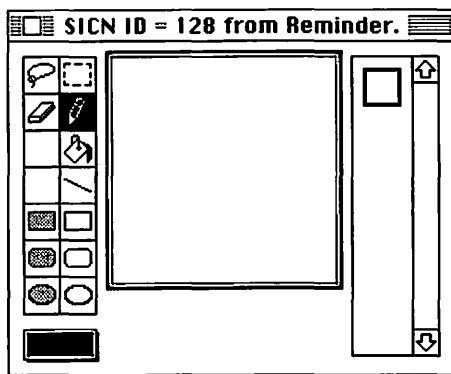
Click **OK** to dismiss the **Auto Position...** window. Then close the DLOG edit window and the DLOG picker window. One more resource to go!

## Creating a SICN Resource

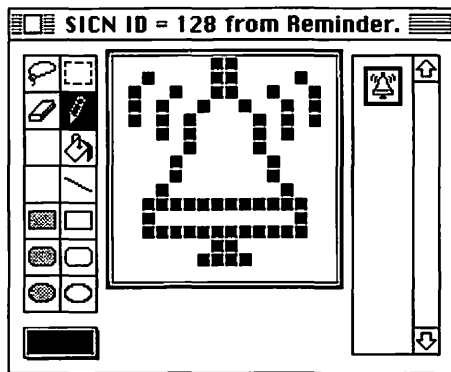
The final (honest) resource you need for the Reminder project is a SICN, or small icon resource. This is the icon that will be rotated with the **Application** menu when your notification occurs. Starting with the window labeled `Reminder.π.rsrc`, select **Create New Resource** from the **Resource** menu. When prompted for a resource type, enter SICN and click the **OK** button. The SICN editing window that appears should look like Figure 6.64.

As you can see, the SICN editor is rather like a simple black and white paint program. On the left, you have some standard paint tools. In the upper right, your SICN is displayed as it will appear on the desktop. The icon we built for Reminder is shown in Figure 6.65. Use it, or create another masterpiece.





**Figure 6.64** SICN editing window.



**Figure 6.65** The Reminder SICN.

When you're done with the SICN, select **Get Resource Info...** in the **Resource** menu and make sure the resource ID is set to 128. Close the SICN editing window and the SICN picker window.

Your resource file should consist of three CNTLs, one MBAR, seven MENUS, one ALRT, one DLOG, and one SICN. Congratulations on surviving the Reminder resource death march! Select **Quit** from the **File** menu and save your changes: It's time to code!

## Setting Up the Project

Launch **THINK C** and create a new project named `Reminder.π` in the `Reminder` folder, where you created your resource file. Add `MacTraps` to the project. Next, create a new source code file, save it as `Reminder.c`, and add it to the project.

Here's the source code for `Reminder.c`:

```
#include <Notification.h>
#include <Processes.h>
#include <Aliases.h>

#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L
#define kSleep              3600L
#define kLeaveWhereItIs      false
#define kUseDefaultProc      (void *) -1L

#define kNotANormalMenu     -1

#define mApple               kBaseResID
#define iAbout               1

#define mFile                kBaseResID+1
#define iSetReminder         1
#define iCancelReminder      2
#define iQuit                4

#define mHours               100
#define mMinutes             101
#define mAMorPM              102
#define mReminders           103

#define kDialogResID         kBaseResID+1

#define iHoursPopup          4
#define iMinutesPopup        5
#define iAMorPMPopup         6

#define iMessageText         8

#define iSoundCheckBox        9
#define iRotateCheckBox       10
#define iLaunchCheckBox       11

#define iAppNameText         12

#define kOn                   1
#define kOff                   0

#define kMarkApp              1
```

```

#define kAM          1
#define kPM          2

#define kMinTextPosition  0
#define kMaxTextPosition 32767

#define kDisableButton  255
#define kEnableButton   0

typedef struct
{
    QElem      queue;
    NMRec      notify;
    FSSpec     file;
    short      hour;
    short      minute;
    Boolean     launch;
    Str255     alert;
    Str255     menuString;
    short      menuItem;
    Boolean     dispose;
    Boolean     wasPosted;
}  ReminderRec, *ReminderPtr;

/*****/
/*  Functions  */
/*****/

void      ToolBoxInit( void );
void      MenuBarInit( void );
void      EventLoop( void );
void      DoEvent( EventRecord *eventPtr );
void      HandleNull( void );
void      HandleMouseDown( EventRecord *eventPtr );
void      HandleMenuChoice( long menuChoice );
void      HandleAppleChoice( short item );
void      HandleFileChoice( short item );

ReminderPtr  HandleDialog( void );

void      GetFileName( StandardFileReply *replyPtr );

```

```

pascal void LaunchResponse( NMRecPtr notifyPtr );
pascal void NormalResponse( NMRecPtr notifyPtr );

void CopyDialogToReminder( DialogPtr dialog,
                           ReminderPtr
                           reminder );

ReminderPtr GetFirstReminder( void );
ReminderPtr GetNextReminder( ReminderPtr reminder );
ReminderPtr GetReminderFromNotification( NMRecPtr
notifyPtr );

ReminderPtr FindReminderOnMenu( short menuItem );
ReminderPtr FindReminderToPost( short hour, short
minute );
ReminderPtr FindReminderToDispose( void );

void SetupReminderMenu( void );
short CountRemindersOnMenu( void );
void RenumberTrailingReminders( ReminderPtr
                               reminder );
void InsertReminderIntoMenu( ReminderPtr
                             reminder );
void ScheduleReminder( ReminderPtr reminder );
void PostReminder( ReminderPtr reminder );
void DeleteReminderFromMenu( ReminderPtr
                             reminder );
void DeleteReminder( ReminderPtr reminder );
ReminderPtr DisposeReminder( ReminderPtr reminder );

void ConcatString( Str255 str1, Str255 str2);

/* see tech note 304 */
pascal OSErr SetDialogDefaultItem(DialogPtr theDialog,
short newItem)
    = { 0x303C, 0x0304, 0xAA68 };
pascal OSErr SetDialogCancelItem(DialogPtr theDialog,
short newItem)
    = { 0x303C, 0x0305, 0xAA68 };
pascal OSErr SetDialogTracksCursor(DialogPtr theDialog,
Boolean tracks)
    = { 0x303C, 0x0306, 0xAA68 };

```

```

/*****/
/*  Globals  */
/*****/

Boolean      gDone;
QHdr         gReminderQueue;

/*****/ main *****/

void      main( void )
{
    ToolBoxInit();
    MenuBarInit();

    EventLoop();
}

/*****/ ToolBoxInit */

void      ToolBoxInit( void )
{
    InitGraf( &qd.thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/*****/ MenuBarInit */

void      MenuBarInit( void )
{
    Handle      menuBar;
    MenuHandle   menu;

    menuBar = GetNewMBar( kBaseResID );
    if ( menuBar == nil )
    {

```

```
        SysBeep( 20 );
        ExitToShell();
    }

    SetMenuBar( menuBar );

    menu = GetMenu( mReminders );
    InsertMenu( menu, kNotANormalMenu );

    menu = GetMHandle( mApple );
    AddResMenu( menu, 'DRVR' );

    DrawMenuBar();
}

/***** EventLoop */
void      EventLoop( void )
{
    EventRecord      event;

    gDone = false;

    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event,
                           GetCaretTime(), nil ) )
            DoEvent( &event );
        else
            HandleNull();
    }
}

/***** DoEvent */
void      DoEvent( EventRecord *eventPtr )
{
    char      theChar;

    switch ( eventPtr->what )
    {
```

```

        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
        case keyDown:
        case autoKey:
            theChar = eventPtr->message & charCodeMask;
            if ( (eventPtr->modifiers & cmdKey) != 0 )
                HandleMenuChoice( MenuKey( theChar ) );
            break;
    }
}

/***** HandleNull *****/

void    HandleNull( void )
{
    unsigned long    time;
    DateTimeRec      dateTime;
    ReminderPtr      theReminder;

    GetDateTime( &time );
    Secs2Date( time, &dateTime );

    theReminder = FindReminderToPost( dateTime.hour,
                                     dateTime.minute );
    while ( theReminder )
    {
        PostReminder( theReminder );
        DeleteReminderFromMenu( theReminder );
        theReminder = FindReminderToPost ( dateTime.hour,
                                           dateTime.minute );
    }

    theReminder = FindReminderToDispose();
    while ( theReminder )
    {
        DisposeReminder( theReminder );
        theReminder = FindReminderToDispose ();
    }
}

```

```

/***** HandleMouseDown *****/

void      HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr      window;
    short          thePart;
    long           menuChoice;

    thePart = FindWindow( eventPtr->where, &window );
    switch ( thePart )
    {
        case inMenuBar:
            SetupReminderMenu();
            menuChoice = MenuSelect( eventPtr->where );
            HandleMenuChoice( menuChoice );
            break;
        case inSysWindow:
            SystemClick( eventPtr, window );
            break;
    }
}

/***** SetupReminderMenu *****/

void      SetupReminderMenu( void )
{
    MenuHandle    fileMenu;
    short         items;

    fileMenu = GetMenu( mFile );
    items = CountRemindersOnMenu();
    if ( items ) EnableItem( fileMenu, iCancelReminder);
    else DisableItem( fileMenu, iCancelReminder);
}

/***** HandleMenuChoice *****/

void      HandleMenuChoice( long menuChoice )
{

```



```

short      menu;
short      item;
ReminderPtr reminder;

if ( menuChoice != 0 )
{
    menu = HiWord( menuChoice );
    item = LoWord( menuChoice );

    switch ( menu )
    {
        case mApple:
            HandleAppleChoice( item );
            break;
        case mFile:
            HandleFileChoice( item );
            break;
        case mReminders:
            reminder = FindReminderOnMenu( item );
            if ( reminder )
                DeleteReminder( reminder );
            break;
    }
    HiliteMenu( 0 );
}

}

/***** HandleAppleChoice *****/

void      HandleAppleChoice( short item )
{
    MenuHandle  appleMenu;
    Str255      accName;
    short       accNumber;

    switch ( item )
    {
        case iAbout:
            NoteAlert( kBaseResID , nil );
            break;
        default:
            appleMenu = GetMHandle( mApple );
            GetItem( appleMenu, item, accName );
    }
}

```

```
        accNumber = OpenDeskAcc( accName );
        break;
    }
}

/***** HandleFileChoice *****/

void    HandleFileChoice( short item )
{
    ReminderPtr  reminder;

    switch ( item )
    {
        case iSetReminder:
            reminder = HandleDialog();
            if ( reminder )
                ScheduleReminder( reminder );
            break;
        case iQuit :
            gDone = true;
            break;
    }
}

/***** GetFirstReminder *****/

ReminderPtr  GetFirstReminder( void )
{
    return( (ReminderPtr)gReminderQueue.qHead );
}

/***** GetNextReminder *****/

ReminderPtr  GetNextReminder( ReminderPtr reminder )
{
    return( (ReminderPtr)reminder->queue.qLink );
}
```

```

/***** FindReminderOnMenu *****/

ReminderPtr FindReminderOnMenu( short menuItem )
{
    ReminderPtr theReminder;

    theReminder = GetFirstReminder();
    while ( theReminder )
    {
        if ( theReminder->menuItem == menuItem )
            break;
        theReminder = GetNextReminder( theReminder );
    }
    return( theReminder );
}

/***** FindReminderToPost *****/

ReminderPtr FindReminderToPost( short hour, short minute )
{
    ReminderPtr theReminder;

    theReminder = GetFirstReminder();
    while ( theReminder )
    {
        if ( (!theReminder->wasPosted)
            && (theReminder->hour <= hour)
            && (theReminder->minute <= minute) )

            break;
        theReminder = GetNextReminder (theReminder);
    }
    return( theReminder );
}

/***** FindReminderToDispose *****/

ReminderPtr FindReminderToDispose( void )
{

```

```
ReminderPtr theReminder;

theReminder = GetFirstReminder ();
while ( theReminder )
{
    if ( theReminder->dispose )
        break;
    theReminder = GetNextReminder (theReminder);
}
return( theReminder );
}

/***** InsertReminderIntoMenu *****/

void      InsertReminderIntoMenu( ReminderPtr reminder )
{
    short          itemBefore;
    MenuHandle      reminderMenu;

    reminderMenu = GetMenu( mReminders );

    itemBefore = CountRemindersOnMenu();

    InsMenuItem( reminderMenu, reminder->menuString,
itemBefore );

    reminder->menuItem = itemBefore + 1;
}

/***** CountRemindersOnMenu *****/

short      CountRemindersOnMenu( void )
{
    MenuHandle reminderMenu;

    reminderMenu = GetMenu( mReminders );

    return( CountMItems( reminderMenu ) );
}
```

```

/***** DeleteReminderFromMenu *****/

void DeleteReminderFromMenu( ReminderPtr reminder )
{
    MenuHandle reminderMenu;

    reminderMenu = GetMenu( mReminders );
    RenumberTrailingReminders( reminder );
    DelMenuItem( reminderMenu, reminder->menuItem );
    reminder->menuItem = 0;
}

/***** RenumberTrailingReminders *****/

void    RenumberTrailingReminders( ReminderPtr reminder )
{
    short    count;

    count = reminder->menuItem;
    reminder = GetNextReminder( reminder );
    while ( reminder )
    {
        if (reminder->menuItem != 0)
            reminder->menuItem = count++;
        reminder = GetNextReminder( reminder );
    }
}

/***** ScheduleReminder *****/

void    ScheduleReminder( ReminderPtr reminder )
{
    Enqueue( &reminder->queue, &gReminderQueue );
    InsertReminderIntoMenu( reminder );
}

```

```

/***** PostReminder *****/

void      PostReminder( ReminderPtr reminder )
{
    reminder->notify.nmRefCon = (long)reminder;
    reminder->wasPosted = true;
    NMInstall( &reminder->notify );
}

/***** DeleteReminder *****/

void      DeleteReminder( ReminderPtr reminder )
{
    if ( reminder->menuItem )
        DeleteReminderFromMenu( reminder );

    reminder->dispose = true;
}

/***** DisposeReminder *****/

ReminderPtr      DisposeReminder( ReminderPtr reminder )
{
    ReminderPtr  next;

    if (reminder->menuItem)
        DeleteReminderFromMenu( reminder );
    next = (ReminderPtr)reminder->queue.qLink;
    Dequeue( &reminder->queue, &gReminderQueue );
    DisposePtr( (Ptr)reminder );
    return( next );
}

/***** GetFileName *****/

void      GetFileName( StandardFileReply *replyPtr )
{
    SFTypelist  typeList;
    short       numTypes;

```

```

    typeList[ 0 ] = 'APPL';
    numTypes = 1;

    StandardGetFile( nil, numTypes, typeList, replyPtr );
}

/***** HandleDialog *****/

ReminderPtr HandleDialog( void )
{
    DialogPtr          dialog;
    Boolean             dialogDone = false;
    short              itemHit, itemType;
    Handle              textItemHandle;
    Handle              itemHandle;
    Handle              okItemHandle;
    Handle              launchItemHandle;
    Rect               itemRect;
    Str255              itemText;
    StandardFileReply   reply;
    ReminderPtr         reminder;

    dialog = GetNewDialog( kDialogResID, nil,
                          kMoveToFront );

    ShowWindow( dialog );
    SetPort( dialog );

    reminder = (ReminderPtr)NewPtr( sizeof
                                    ( ReminderRec ) );
    reminder->menuItem = 0;
    reminder->dispose = false;
    reminder->wasPosted = false;

    SetDialogDefaultItem( dialog, ok );
    SetDialogCancelItem( dialog, cancel );
    SetDialogTracksCursor( dialog, true );

    GetDItem( dialog, iMessageText, &itemType,
              &textItemHandle, &itemRect );
    GetDItem( dialog, ok, &itemType, &okItemHandle,
              &itemRect );
    GetDItem( dialog, iLaunchCheckBox, &itemType,
              &launchItemHandle, &itemRect );

```

```

SetText( dialog, iMessageText, kMinTextPosition,
          kMaxTextPosition );

while ( ! dialogDone )
{
    GetIText( textItemHandle, itemText );

    if ( itemText[ 0 ] == 0 &&
        !GetCtlValue( (ControlHandle)
                      launchItemHandle ) )
        HiliteControl( (ControlHandle)okItemHandle,
                        kDisableButton );
    else
        HiliteControl( (ControlHandle)okItemHandle,
                        kEnableButton );

    ModalDialog( nil, &itemHit );

    switch ( itemHit )
    {
        case ok:
        case cancel:
            dialogDone = true;
            break;
        case iSoundCheckBox:
        case iRotateCheckBox:
            GetDItem( dialog, itemHit, &itemType,
                      &itemHandle, &itemRect );
            SetCtlValue( (ControlHandle)itemHandle,
                          ! GetCtlValue( (ControlHandle)
                                           itemHandle ) );

            break;
        case iLaunchCheckBox:
        case iAppNameText:
            if ( ! GetCtlValue( (ControlHandle)
                              launchItemHandle ) )
            {
                GetFileName( &reply );
                if ( reply.sfGood )
                {
                    SetCtlValue( (ControlHandle)
                                launchItemHandle, kOn );
                    reminder->file = reply.sfFile;
                }
            }
        }
    }
}

```



```

        GetDItem( dialog, iAppNameText,
                  &itemType,&itemHandle,
                  &itemRect );
        SetIText( itemHandle,
                  reminder->file.name );
    }
}
else
{
    SetCtlValue( (ControlHandle)
                  launchItemHandle, kOff );
    GetDItem( dialog, iAppNameText,
              &itemType,&itemHandle,
              &itemRect );
    SetIText( itemHandle,
              "\p<Not Selected>" );
}
break;
}
}

if ( itemHit == cancel )
{
    DisposePtr( (Ptr)reminder );
    reminder = nil;
} else
    CopyDialogToReminder( dialog, reminder );

DisposDialog( dialog );

return( reminder );
}

/***** CopyDialogToReminder */

void CopyDialogToReminder( DialogPtr dialog,
                           ReminderPtr reminder)
{
    short      itemType;
    Rect       itemRect;
    Handle     itemHandle;
    Str255     string;
    MenuHandle menu;
    short      val;
    long       tmp;

```

```
GetDItem( dialog, iMessageText, &itemType,
          &itemHandle, &itemRect );
GetIText( itemHandle, reminder->alert );
reminder->notify.nmStr = (StringPtr)&reminder->alert;

GetDItem( dialog, iSoundCheckBox, &itemType,
          &itemHandle, &itemRect );
if ( GetCtlValue( (ControlHandle)itemHandle ) )
    reminder->notify.nmSound = (Handle)-1L;
else
    reminder->notify.nmSound = nil;

GetDItem( dialog, iRotateCheckBox, &itemType,
          &itemHandle, &itemRect );
if( GetCtlValue( (ControlHandle)itemHandle ) )
    reminder->notify.nmIcon = GetResource( 'SICN',
                                          kBaseResID );
else
    reminder->notify.nmIcon = nil;

GetDItem( dialog, iLaunchCheckBox, &itemType,
          &itemHandle, &itemRect );
if( reminder->launch = GetCtlValue(
    (ControlHandle)itemHandle ) )
    reminder->notify.nmResp = &LaunchResponse;
else
    reminder->notify.nmResp = &NormalResponse;

GetDItem( dialog, iHoursPopup, &itemType, &itemHandle,
          &itemRect );
val = GetCtlValue( (ControlHandle)itemHandle );
NumToString( (long) val, string );
StringToNum ( string, &tmp );
reminder->hour = tmp;

reminder->menuString[0] = 0;
ConcatString( reminder->menuString, string );
ConcatString( reminder->menuString, "\\p:" );

GetDItem( dialog, iMinutesPopup, &itemType,
          &itemHandle, &itemRect );
val = GetCtlValue( (ControlHandle)itemHandle );
menu = GetMHandle( mMinutes );
GetItem( menu, val, string );
```

```

StringToNum ( string, &tmp );
reminder->minute = tmp;

ConcatString( reminder->menuString, string );
ConcatString( reminder->menuString, "\p ");

GetDItem( dialog, iAMorPMPopup, &itemType,
          &itemHandle, &itemRect );
val = GetCtlValue( (ControlHandle)itemHandle );

if( val == kPM )
    reminder->hour += 12;

menu = GetMenu ( mAMorPM );
GetItem( menu, val, string );
ConcatString( reminder->menuString, string );

reminder->notify.qType = nmType;
reminder->notify.nmMark = kMarkApp;
}

/***** ConcatString *****/

void      ConcatString( Str255 str1, Str255 str2)
{
    short i;

    for (i=str1[0];i<str2[0]+str1[0];i++)
    {
        str1[i+1]=str2[i-str1[0]+1];
    }
    str1[0]=i;
}

/***** NormalResponse *****/

pascal void      NormalResponse( NMRecPtr notifyPtr )
{
    ReminderPtr  reminder;
    OSErr        err;

    reminder = GetReminderFromNotification( notifyPtr );
    err = NMRemove( notifyPtr );
    reminder->dispose = true;
}

```

```

/***** LaunchResponse *****/

pascal void LaunchResponse( NMRecPtr notifyPtr )
{
    LaunchParamBlockRec launchParams;
    OSErr err;
    FSSpec fileSpec;
    ReminderPtr reminder;
    Boolean isFolder;
    Boolean wasAlias;

    reminder = GetReminderFromNotification( notifyPtr );

    fileSpec = reminder->file;

    err = ResolveAliasFile( &fileSpec, true, &isFolder,
                          &wasAlias );

    launchParams.launchBlockID = extendedBlock;
    launchParams.launchEPBLength = extendedBlockLen;
    launchParams.launchFileFlags = 0;
    launchParams.launchControlFlags = launchContinue +
        launchNoFileFlags;
    launchParams.launchAppSpec = &fileSpec;

    launchParams.launchAppParameters = nil;

    if ( LaunchApplication( &launchParams ) ) SysBeep( 20 );

    err = NMRemove( notifyPtr );

    reminder->dispose = true;
}

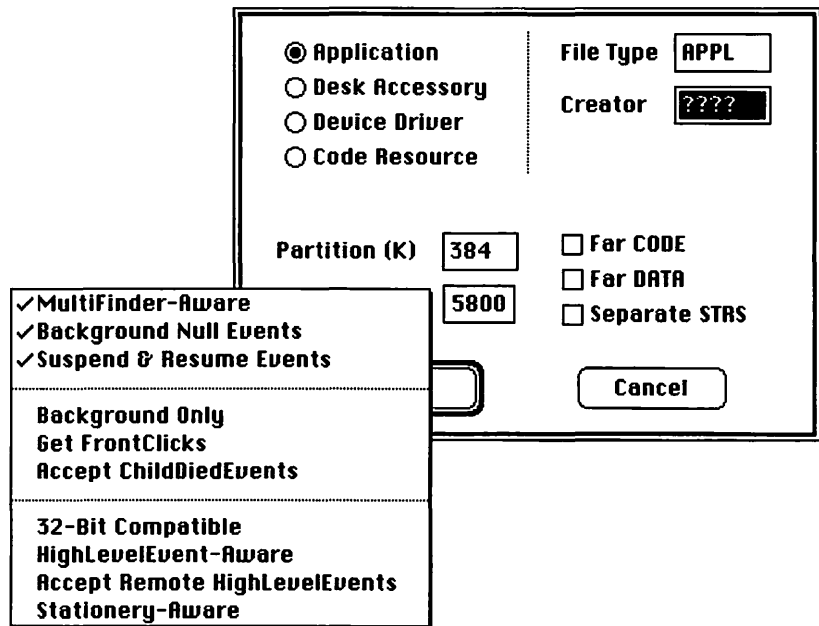
/***** GetReminderFromNotification *****/

ReminderPtr GetReminderFromNotification( NMRecPtr
                                         notifyPtr )
{
    return (ReminderPtr) notifyPtr->nmRefCon;
}

```

## Running Reminder

Now that your source code is in, you're about ready to run Reminder. Like EventTrigger in Chapter 4 and WorldClock in Chapter 5, you should set up the SIZE flags first. To do this, select **Set Project Type...** from the **Project** menu. Set the flags as shown in Figure 6.66.



**Figure 6.66** Setting the SIZE flags.

Save your changes, then select **Run** from the **Project** menu. When asked to **Bring the project up to date?** click **Yes**. If everything went well, the Reminder menus should appear in the menu bar. Let's look at each one in turn. First, examine the **Reminder** menu. It should display the **About Reminder...** menu item at the top, and the remainder of the **Reminder** menu items below (Figure 6.67).

As with WorldClock, all **Reminder** menu items are fully available in Reminder. Select the **About Reminder...** menu item in the **Reminder** menu. You should see the alert you created earlier (Figure 6.68).

Now, examine the **File** menu. It should look like Figure 6.69, with only the **New Reminder** and **Quit** menu items enabled.

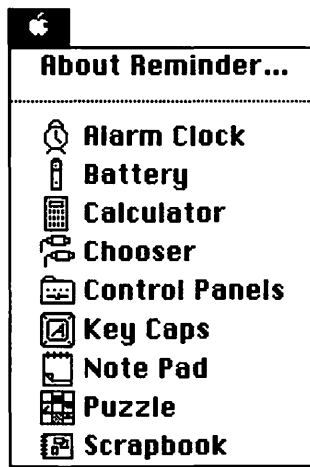


Figure 6.67 Reminder's Apple menu.

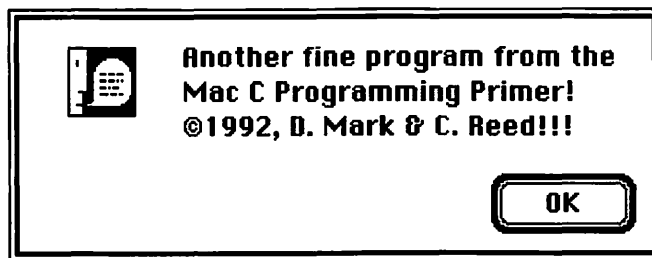


Figure 6.68 The About Reminder... alert.

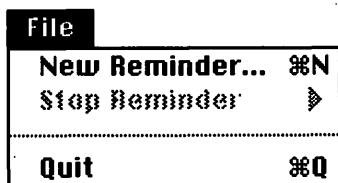


Figure 6.69 Reminder's File Menu.

The **Edit** Menu should be completely disabled (Figure 6.70).

The **Edit** menu will be enabled if a desk accessory is used or if the Reminder dialog is open, as you'll see in a moment.

If everything seems to be working OK, select **New Reminder...** from the **File** menu. You should see the dialog box shown in Figure 6.71, centered on the monitor.

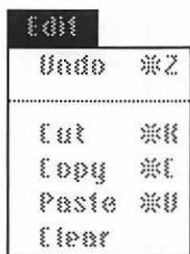


Figure 6.70 Reminder's **Edit** Menu (disabled).



Figure 6.71 Reminder in action.

Reminder is ready to do some work for you. Let's start by looking at the three pop-up menus at the top of the dialog box. If you click on the **Hour**, the **Minute** and the **AM/PM** pop-up menus in turn, you should see the menu items shown in Figure 6.72.

✓1	✓00	✓AM
2	05	PM
3	10	
4	15	
5	20	
6	25	
7	30	
8	35	
9	40	
10	45	
11	50	
12	55	

Figure 6.72 Reminder's pop-up menus.



If things didn't go as planned, check the Reminder resources, as well as the source code in `Reminder.c`. Make sure your project and resource files are named correctly. Since this is the longest of all the Primer projects, it's possible you introduced a bug somewhere along the way. Check everything carefully.

Use the popups to select a time for this reminder. Suppose it is 9:40 P.M. (prime time for programmers). Enter a test reminder for 11 P.M. as shown in Figure 6.73.

Click **OK**. Now go to the **File** menu and select **Stop Notification**. Keep the mouse down to examine the hierarchical menu. It should display one reminder for 11 P.M. (Figure 6.74). Select the 11 P.M. menu item. This will cancel the notification.

Enter a test reminder and set the time for just after whatever the current time is. In this example, the current time is 9:47 P.M. Set the first pop-up menu to **9**, the second pop-up menu to **50**, and the third pop-up menu to **PM**. The dialog box should look like Figure 6.75. Notice that the **Play Sound** and **Rotate Icon** check boxes are checked. Also notice the important reminder in the **Message:** field.



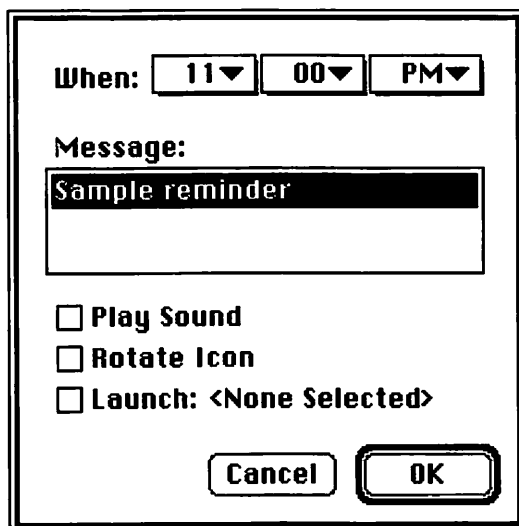


Figure 6.73 The first reminder.

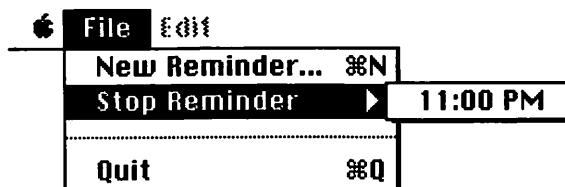
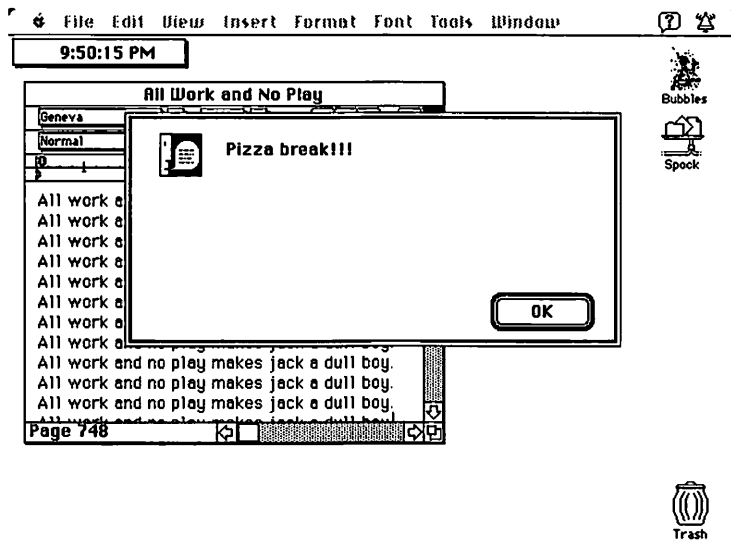


Figure 6.74 A reminder is set for 11 P.M.



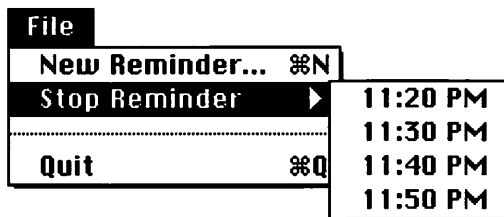
Figure 6.75 Making a test reminder.

Once you've set the text and pop-up menus, click on the **OK** button to set the reminder. Now, try starting another application (we'll run Microsoft Word). At the appointed time, your Mac should beep, and the alert shown in Figure 6.76 should appear. If you look carefully, you'll see the SICN you designed rotating back and forth with your application's icon. Figure 6.76 shows our bell SICN.



**Figure 6.76** Making a test reminder.

Now, let's see what happens when you set multiple reminders. Use **New Reminder...** to set four reminders for some time after the current time. In our example, we'll set up reminders for 11:20 P.M., 11:30 P.M., 11:40 P.M., and 11:50 P.M. Once you've completed entering the reminders, select the **Stop Notification** menu item in the **File** menu again. It should look something like Figure 6.77.



**Figure 6.77** Four reminders set after 11 P.M.

Next, create a new reminder. This time, click on the **Launch Application** check box. The standard **Open File** dialog will appear (Figure 6.78). Use this dialog to select any application you like.

Go ahead and select an application (we selected EventTracker). When you click on the **Open** button, the name of the application is placed in the **Reminder** dialog box (Figure 6.79).

If you accept the reminder, your application will be launched at the appropriate time.

Finally, type the Command key equivalent **⌘Q** to quit Reminder. Typing **⌘Q** is equivalent to selecting **Quit** from Reminder's **File** menu.

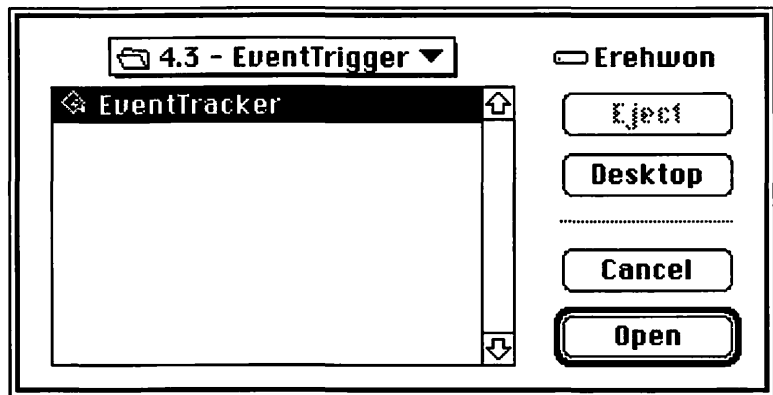


Figure 6.78 Looking for an application to launch.

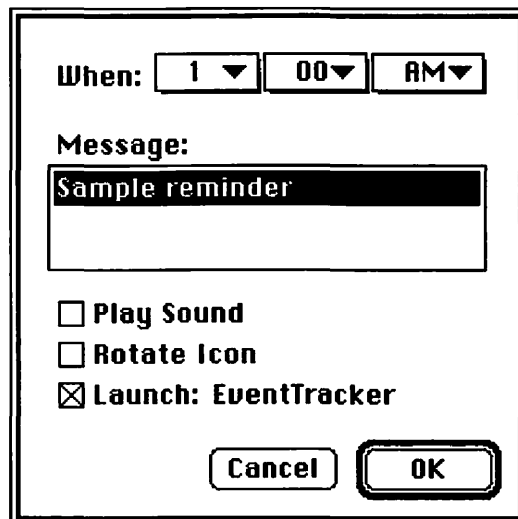


Figure 6.79 Using Reminder to launch an application.

## Walking Through the Reminder Code

---

Let's take a look at the Reminder code. We'll focus on the Toolbox issues and not so much on the Reminder algorithm.

Reminder.c includes three special files you'll need in order to use the Notification and Process managers, and to resolve file aliases when Reminder launches an application.

```
#include <Notification.h>
#include <Processes.h>
#include <Aliases.h>
```

We'll address the #defines as they occur in the code.

```
#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L
#define kSleep              3600L
#define kLeaveWhereItIs      false
#define kUseDefaultProc      (void *) -1L

#define kNotANormalMenu     -1

#define mApple              kBaseResID
#define iAbout              1

#define mFile               kBaseResID+1
#define iSetReminder        1
#define iCancelReminder     2
#define iQuit               4

#define mHours              100
#define mMinutes            101
#define mAMorPM             102
#define mReminders          103

#define kDialogResID        kBaseResID+1

#define iHoursPopup         4
#define iMinutesPopup       5
#define iAMorPMPopup        6

#define iMessageText        8
```

```

#define iSoundCheckBox          9
#define iRotateCheckBox        10
#define iLaunchCheckBox        11

#define iAppNameText           12

#define kOn                     1
#define kOff                    0

#define kMarkApp                1

#define kAM                     1
#define kPM                     2

#define kMinTextPosition        0
#define kMaxTextPosition        32767

#define kDisableButton          255
#define kEnableButton           0

```

We'll use the `ReminderRec` structure to hold all the data associated with the **New Reminder...** dialog. Each field will be explained as it is used. Remember that `ReminderPtr` is declared as a pointer to a `ReminderRec`. Quite a few of `Reminder's` routines either take a `ReminderPtr` as a parameter or return a `ReminderPtr`.

```

typedef struct
{
    QElem          queue;
    NMRec          notify;
    FSSpec         file;
    short          hour;
    short          minute;
    Boolean        launch;
    Str255         alert;
    Str255         menuString;
    short          menuItem;
    Boolean        dispose;
    Boolean        wasPosted;
} ReminderRec,    *ReminderPtr;

```

**Lots of function prototypes...**

```

/*****
/*  Functions  */
*****/

void          ToolBoxInit( void );
void          MenuBarInit( void );
void          EventLoop( void );
void          DoEvent( EventRecord *eventPtr );
void          HandleNull( void );
void          HandleMouseDown( EventRecord *eventPtr );
void          HandleMenuChoice( long menuChoice );
void          HandleAppleChoice( short item );
void          HandleFileChoice( short item );

ReminderPtr   HandleDialog( void );

void          GetFileName( StandardFileReply
                        *replyPtr );

pascal void   LaunchResponse( NMRecPtr notifyPtr );
pascal void   NormalResponse( NMRecPtr notifyPtr );

void          CopyDialogToReminder( DialogPtr dialog,
                        ReminderPtr reminder );

ReminderPtr   GetFirstReminder( void );
ReminderPtr   GetNextReminder( ReminderPtr reminder );
ReminderPtr   GetReminderFromNotification( NMRecPtr
                        notifyPtr );

ReminderPtr   FindReminderOnMenu( short menuItem );
ReminderPtr   FindReminderToPost( short hour,
                        short minute );
ReminderPtr   FindReminderToDispose( void );

void          SetupReminderMenu( void );
short         CountRemindersOnMenu( void );
void          RenumberTrailingReminders( ReminderPtr
                        reminder );
void          InsertReminderIntoMenu( ReminderPtr
                        reminder );
void          ScheduleReminder( ReminderPtr reminder );
void          PostReminder( ReminderPtr reminder );
void          DeleteReminderFromMenu( ReminderPtr
                        reminder );

```

```

void          DeleteReminder( ReminderPtr reminder );
ReminderPtr   DisposeReminder( ReminderPtr reminder );

void          ConcatString( Str255 str1, Str255 str2);

```

These three routines were explained earlier in the chapter. THINK C 5 did not include these declarations in the Dialog Manager #include file, so we've included them here. If your version of THINK C comes with these declarations built in, just delete the duplicate declarations below.

```

/* see tech note 304 */
pascal OSErr SetDialogDefaultItem(DialogPtr theDialog,
                                   short newItem)
    = { 0x303C, 0x0304, 0xAA68 };
pascal OSErr SetDialogCancelItem(DialogPtr theDialog,
                                   short newItem)
    = { 0x303C, 0x0305, 0xAA68 };
pascal OSErr SetDialogTracksCursor(DialogPtr theDialog,
                                    Boolean tracks)
    = { 0x303C, 0x0306, 0xAA68 };

```

gDone plays its normal role. gReminderQueue is a pointer to the **reminder queue**. We'll use some of the operating system utilities to create and maintain a queue of reminders. The routine Enqueue() places a queue element at the end of the queue, and the routine Dequeue() removes an element from the queue. As you'll see, Reminder uses these routines to keep track of individual ReminderRecs.

```

/*****
/* Globals */
*****/

Boolean      gDone;
QHdr         gReminderQueue;

```

main() initializes the Toolbox and the menu bar, then enters the main event loop.

```
/****** main *****/
```

```
void    main( void )
{
    ToolBoxInit();
    MenuBarInit();

    EventLoop();
}
```

`ToolBoxInit()` looks the same as ever.

```
/****** ToolBoxInit */
```

```
void    ToolBoxInit( void )
{
    InitGraf( &qd.thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}
```

`MenuBarInit()` loads the MBar resource. If the resource couldn't be loaded, beep once, then exit.

```
/****** MenuBarInit */
```

```
void    MenuBarInit( void )
{
    Handle        menuBar;
    MenuHandle     menu;

    menuBar = GetNewMBar( kBaseResID );
    if ( menuBar == nil )
    {
        SysBeep( 20 );
        ExitToShell();
    }
}
```



Once the MBar is loaded, it is made current. Next, the mReminders popup is installed in the menu list and marked as kNotANormalMenu. Next, the Apple menu is built and the menu bar is drawn.

```

SetMenuBar( menuBar );

menu = GetMenu( mReminders );
InsertMenu( menu, kNotANormalMenu );

menu = GetMHandle( mApple );
AddResMenu( menu, 'DRVr' );

DrawMenuBar();
}

```

EventLoop() calls WaitNextEvent() to retrieve an event from the event queue. Interestingly, the same routines we'll use to manage the reminder queue are used by the Event Manager to manage the event queue.

```

/***** EventLoop */

void      EventLoop( void )
{
    EventRecord      event;

    gDone = false;

    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event,
                           GetCaretTime(), nil ) )
            DoEvent( &event );
        else
            HandleNull();
    }
}

```

Since we aren't supporting any windows, the list of handled events is fairly small.



In that case, `PostReminder()` is called, passing the reminder on to the Notification Manager. Next, `DeleteReminderFromMenu()` is called to delete the reminder from the hierarchical menu. Finally, `FindReminderToPost()` is called again, to see if any other reminders are ready for posting.

```
while ( theReminder )
{
    PostReminder( theReminder );
    DeleteReminderFromMenu( theReminder );
    theReminder = FindReminderToPost ( dateTime.hour,
                                     dateTime.minute );
}
```

Next, the same process is repeated, using the routine `FindReminderToDispose()`. `FindReminderToDispose()` steps through the queue, looking for reminders that either have been deleted (by way of the **Stop Notification** menu) or have been to the Notification Manager and have completed their notification.

```
theReminder = FindReminderToDispose();
while ( theReminder )
{
```

`DisposeReminder()` calls `Dequeue()` to remove the specified reminder from the reminder queue, then disposes of the memory allocated for the reminder's data structure.

```
    DisposeReminder( theReminder );
    theReminder = FindReminderToDispose ();
}
}
```

`HandleMouseDown()` calls `FindWindow()` to find which window the mouseDown was in.

```
/***** HandleMouseDown *****/
```

```
void    HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr    window;
    short         thePart;
    long          menuChoice;
```

```

thePart = FindWindow( eventPtr->where, &window );
switch ( thePart )
{

```

If the mouseDown was in the menu bar, call SetupReminderMenu() to enable the **Stop Reminder** menu if there is at least one current reminder. Then call MenuSelect() and pass the selected menu on to HandleMenuChoice().

```

case inMenuBar:
    SetupReminderMenu();
    menuChoice = MenuSelect( eventPtr->where );
    HandleMenuChoice( menuChoice );
    break;

```

If the mouseDown was inSysWindow, pass the event on to SystemClick().

```

case inSysWindow:
    SystemClick( eventPtr, window );
    break;
}
}

```

SetupReminderMenu() gets a handle to the **File** menu from GetMenu(), then calls CountRemindersOnMenu() to find out how many items are currently on the hierarchical menu.

```

/***** SetupReminderMenu *****/

```

```

void    SetupReminderMenu( void )
{
    MenuHandle    fileMenu;
    short         items;

    fileMenu = GetMenu( mFile );
    items = CountRemindersOnMenu();

```

If the hierarchical menu contains at least one item, enable the **File** menu's **Stop Reminder** item, else disable the menu item.

```

    if ( items ) EnableItem( fileMenu, iCancelReminder);
    else DisableItem( fileMenu, iCancelReminder);
}

```

HandleMenuChoice() passes the selected item on to HandleAppleChoice() or HandleFileChoice() if the selection was from one of those two menus.

```

/***** HandleMenuChoice *****/

```

```

void      HandleMenuChoice( long menuChoice )
{
    short          menu;
    short          item;
    ReminderPtr     reminder;

    if ( menuChoice != 0 )
    {
        menu = HiWord( menuChoice );
        item = LoWord( menuChoice );

        switch ( menu )
        {
            case mApple:
                HandleAppleChoice( item );
                break;
            case mFile:
                HandleFileChoice( item );
                break;

```

If the selection was from the hierarchical menu, FindReminderOnMenu() is called to turn the menu item into a ReminderPtr. If a reminder was found, DeleteReminder() is called to delete it from the queue.

```

            case mReminders:
                reminder = FindReminderOnMenu( item );
                if ( reminder )
                    DeleteReminder( reminder );
                break;
        }
        HiliteMenu( 0 );
    }
}

```

HandleAppleChoice() handles all selections in the Apple menu.

```

/***** HandleAppleChoice *****/

```

```

void    HandleAppleChoice( short item )
{
    MenuHandle    appleMenu;
    Str255        accName;
    short         accNumber;

    switch ( item )
    {

```

If the **About Reminder...** item is selected, `NoteAlert()` is called to display the alert we built earlier. Otherwise, `OpenDeskAcc()` is called to open the appropriate item in the Apple menu.

```

        case iAbout:
            NoteAlert( kBaseResID , nil );
            break;
        default:
            appleMenu = GetMHandle( mApple );
            GetItem( appleMenu, item, accName );
            accNumber = OpenDeskAcc( accName );
            break;
    }
}

```

If the **New Reminder...** item is selected from the **File** menu, `HandleDialog()` is called to bring up the dialog box designed earlier.

```

/***** HandleFileChoice *****/

```

```

void    HandleFileChoice( short item )
{
    ReminderPtr  reminder;

    switch ( item )
    {
        case iSetReminder:
            reminder = HandleDialog();

```

`HandleDialog()` returns a `ReminderPtr` if the **OK** button was clicked. In that case, the reminder is passed on to `ScheduleReminder()` to be placed on the reminder queue.

```

    if ( reminder )
        ScheduleReminder( reminder );
    break;

```

If **Quit** was selected, `gDone` is set to `true`, allowing `Reminder` to exit.

```

        case iQuit :
            gDone = true;
            break;
    }
}

```

`GetFirstReminder()` returns the first element on the reminder queue.

```

/***** GetFirstReminder *****/

```

```

ReminderPtr GetFirstReminder( void )
{
    return( (ReminderPtr)gReminderQueue.qHead );
}

```

`GetNextReminder()` takes a pointer to a queue element and returns the next element on the queue.

```

/***** GetNextReminder *****/

```

```

ReminderPtr GetNextReminder( ReminderPtr reminder )
{
    return( (ReminderPtr)reminder->queue.qLink );
}

```

`FindReminderOnMenu()` steps through the reminder queue, comparing the `menuItem` field of each `ReminderRec` with the specified `menuItem`. If a match is found, a pointer to that reminder is returned.

```

/***** FindReminderOnMenu *****/

```

```

ReminderPtr FindReminderOnMenu( short menuItem )
{
    ReminderPtr theReminder;

```

```

    theReminder = GetFirstReminder();
    while ( theReminder )
    {
        if ( theReminder->menuItem == menuItem )
            break;
        theReminder = GetNextReminder( theReminder );
    }
    return( theReminder );
}

```

**FindReminderToPost()** steps through the queue, looking for a reminder which has not yet been posted (passed on to the Notification Manager) and whose time has passed or equalled the hour and minute specified in the parameters.

```

/***** FindReminderToPost *****/

```

```

ReminderPtr FindReminderToPost( short hour, short minute )
{
    ReminderPtr theReminder;

    theReminder = GetFirstReminder();
    while ( theReminder )
    {
        if ( (!theReminder->wasPosted)
            && (theReminder->hour <= hour)
            && (theReminder->minute <= minute) )

            break;
        theReminder = GetNextReminder (theReminder);
    }
}

```

If the entire queue was searched with no success, the last elements next item pointer (which is conveniently initialized to nil) will be returned.

```

    return( theReminder );
}

```

**FindReminderToDispose()** walks through the reminder queue, returning the first reminder whose dispose flag is set to true.



```

/***** FindReminderToDispose *****/

```

```

ReminderPtr FindReminderToDispose( void )
{
    ReminderPtr theReminder;

    theReminder = GetFirstReminder ();
    while ( theReminder )
    {
        if ( theReminder->dispose )
            break;
        theReminder = GetNextReminder (theReminder);
    }
    return( theReminder );
}

```

InsertReminderIntoMenu() uses InsMenuItem() to insert the reminder's menuString in the hierarchical menu. The reminder's menuItem is set to indicate its position in the menu.

```

/***** InsertReminderIntoMenu *****/

```

```

void      InsertReminderIntoMenu( ReminderPtr reminder )
{
    short      itemBefore;
    MenuHandle reminderMenu;

    reminderMenu = GetMenu( mReminders );

    itemBefore = CountRemindersOnMenu();

    InsMenuItem( reminderMenu, reminder->menuString,
                itemBefore );

    reminder->menuItem = itemBefore + 1;
}

```

CountRemindersOnMenu() returns the number of reminders on the hierarchical menu.

```

/***** CountRemindersOnMenu *****/

```

```

short    CountRemindersOnMenu( void )
{
    MenuHandle reminderMenu;

    reminderMenu = GetMenu( mReminders );

    return( CountMItems( reminderMenu ) );
}

```

`DeleteReminderFromMenu()` deletes a reminder from the hierarchical menu, calling `RenumberTrailingReminders()` to update the remaining reminder's item numbers. Finally, the `menuItem` field is set to 0, indicating that this reminder is no longer in the hierarchical menu.

```

/***** DeleteReminderFromMenu *****/

```

```

void      DeleteReminderFromMenu( ReminderPtr reminder )
{
    MenuHandle reminderMenu;

    reminderMenu = GetMenu( mReminders );
    RenumberTrailingReminders( reminder );
    DelMenuItem( reminderMenu, reminder->menuItem );
    reminder->menuItem = 0;
}

```

`RenumberTrailingReminders()` steps through the reminder queue from the specified reminder through to the end, updating each reminder's `menuItem` field. Reminders whose `menuItem` field is set to 0 are ignored.

```

/***** RenumberTrailingReminders *****/

```

```

void      RenumberTrailingReminders( ReminderPtr reminder )
{
    short      count;

    count = reminder->menuItem;
    reminder = GetNextReminder( reminder );
    while ( reminder )
    {

```

```

        if (reminder->menuItem != 0)
            reminder->menuItem = count++;
        reminder = GetNextReminder( reminder );
    }
}

```

**ScheduleReminder()** places the specified reminder on the reminder queue and into the hierarchical menu.

```

/***** ScheduleReminder *****/

void    ScheduleReminder( ReminderPtr reminder )
{
    Enqueue( &reminder->queue, &gReminderQueue );
    InsertReminderIntoMenu( reminder );
}

```

**PostReminder()** posts the reminder on the Notification Manager's queue using **NMInstall()**. Don't confuse the reminder queue with the Notification Manager's queue. As we explained earlier in the chapter, the Notification Manager routines work with a **NMRec** data structure, not a **ReminderRec**.

```

/***** PostReminder *****/

void    PostReminder( ReminderPtr reminder )
{

```

The **ReminderRec** struct has a **NMRec** embedded in it. A pointer to the reminder is placed in the **NMRec**'s **nmRefCon** field for later retrieval. The reminder's **wasPosted** flag is set to true and the **NMRec** is passed to **NMInstall()**.

```

    reminder->notify.nmRefCon = (long)reminder;
    reminder->wasPosted = true;
    NMInstall( &reminder->notify );
}

```

**DeleteReminder()** first checks the reminder's **menuItem** flag to see if the reminder is in the hierarchical menu. If so, **DeleteReminderFromMenu()** is called. Either way, the reminder's **dispose** flag is set to true, marking the reminder for later disposal.

```

/***** DeleteReminder *****/

```

```

void      DeleteReminder( ReminderPtr reminder )
{
    if ( reminder->menuItem )
        DeleteReminderFromMenu( reminder );

    reminder->dispose = true;
}

```

DisposeReminder() deletes the reminder from the menu, if appropriate. The next reminder in the list is stored in the variable next for later returning. Next, the reminder is removed from the reminder queue using Dequeue(). Finally, the memory allocated for the reminder is freed up with a call to DisposePtr().

```

/***** DisposeReminder *****/

```

```

ReminderPtr      DisposeReminder( ReminderPtr reminder )
{
    ReminderPtr  next;

    if (reminder->menuItem)
        DeleteReminderFromMenu( reminder );
    next = (ReminderPtr)reminder->queue.qLink;
    Dequeue( &reminder->queue, &gReminderQueue );
    DisposePtr( (Ptr)reminder );
    return( next );
}

```

GetFileName() calls StandardGetFile() to ask the user to select an application file. A complete description of this routine is found in Chapter 7's OpenPICT program. Reminder uses the selected application when the **Launch:** check box is checked.

```

/***** GetFileName *****/

```

```

void      GetFileName( StandardFileReply *replyPtr )
{
    SFTypelist  typeList;
    short       numTypes;

    typeList[ 0 ] = 'APPL';
    numTypes = 1;
}

```

```

        StandardGetFile( nil, numTypes, typeList, replyPtr );
    }

```

Perhaps one of the most important routines in this program and one of the longest, `HandleDialog()` implements the **New Reminder...** dialog box.

```

/***** HandleDialog *****/

```

```

ReminderPtr HandleDialog( void )
{
    DialogPtr      dialog;
    Boolean        dialogDone = false;
    short          itemHit, itemType;
    Handle         textItemHandle;
    Handle         itemHandle;
    Handle         okItemHandle;
    Handle         launchItemHandle;
    Rect           itemRect;
    Str255         itemText;
    StandardFileReply reply;
    ReminderPtr    reminder;

```

First, the `DLOG` and `DITL` are loaded by the call to `GetNewDialog()`. Next, the dialog is made visible and the current port.

```

    dialog = GetNewDialog( kDialogResID, nil,
                          kMoveToFront );

    ShowWindow( dialog );
    SetPort( dialog );

```

Next, a `ReminderRec` is allocated and initialized.

```

    reminder = (ReminderPtr)NewPtr ( sizeof
                                    ( ReminderRec ) );

    reminder->menuItem = 0;
    reminder->dispose = false;
    reminder->wasPosted = false;

```

These three routines were discussed earlier in the chapter.

```
SetDialogDefaultItem( dialog, ok );
SetDialogCancelItem( dialog, cancel );
SetDialogTracksCursor( dialog, true );
```

The first call to `GetDItem()` places a handle to the editable text field in `textItemHandle`. The second call loads a handle to the **OK** button in `okItemHandle`. The third call places a handle to the **Launch: check box** in `launchItemHandle`. `SelIText()` makes sure the entire text string in the editable text field is selected when the dialog box first appears.

```
GetDItem( dialog, iMessageText, &itemType,
          &textItemHandle, &itemRect );
GetDItem( dialog, ok, &itemType, &okItemHandle,
          &itemRect );
GetDItem( dialog, iLaunchCheckBox, &itemType,
          &launchItemHandle, &itemRect );
SelIText( dialog, iMessageText, kMinTextPosition,
          kMaxTextPosition );
```

When the dialog loop is first entered, `GetIText()` is called, placing the text from the editable text field into `itemText`.

```
while ( ! dialogDone )
{
    GetIText( textItemHandle, itemText );
```

If the editable text field was empty and the **Launch: check box** is unchecked, call `HiliteControl()` to dim the **OK** button, else enable the **OK** button.

```
if ( itemText[ 0 ] == 0 &&
    !GetCtlValue( (ControlHandle)
                  launchItemHandle ) )
    HiliteControl( (ControlHandle)okItemHandle,
                  kDisableButton );
else
    HiliteControl( (ControlHandle)okItemHandle,
                  kEnableButton );
```

Call `ModalDialog()`, retrieving the item acted upon by the user in the variable `itemHit`.

```

ModalDialog( nil, &itemHit );

switch ( itemHit )
{

```

If itemHit is the ok or cancel, drop out of the dialog loop.

```

    case ok:
    case cancel:
        dialogDone = true;
        break;

```

If the **Play Sound** or **Rotate Icon** check boxes were hit, reverse their values, checking an unchecked box or unchecking a checked box.

```

    case iSoundCheckBox:
    case iRotateCheckBox:
        GetDItem( dialog, itemHit, &itemType,
                  &itemHandle, &itemRect );
        SetCtlValue( (ControlHandle)itemHandle,
                     ! GetCtlValue
                       ( (ControlHandle)itemHandle ) );
        break;

```

If either the **Launch:** check box or the static text string bearing the launched application's name was hit, check the value of the **Launch:** check box.

```

    case iLaunchCheckBox:
    case iAppNameText:
        if ( ! GetCtlValue
              ( (ControlHandle)launchItemHandle ) )
        {

```

If the check box is unchecked, call `GetFileName()` to get an application file name. If the user didn't hit the **Cancel** button when prompted for an application name, copy the application name into the `iAppNameText` static text string. As we stated earlier, the `File Manager` and `StandardGetFile()` are covered in Chapter 7's `OpenPICT` program.

```

        GetFileName( &reply );
        if ( reply.sfGood )
        {

```

```

        SetCtlValue( (ControlHandle)
            launchItemHandle, kOn );
        reminder->file = reply.sfFile;
        GetDItem( dialog, iAppNameText,
            &itemType, &itemHandle,
            &itemRect );
        SetIText( itemHandle,
            reminder->file.name );
    }
}

```

If the **Launch:** check box was already checked, call `SetCtlValue()` to uncheck it. Next, call `SetIText()` to set the `iAppNameText` static text item to "\p<Not Selected>".

```

        else
        {
            SetCtlValue( (ControlHandle)
                launchItemHandle, kOff );
            GetDItem( dialog, iAppNameText,
                &itemType, &itemHandle,
                &itemRect );
            SetIText( itemHandle,
                "\p<Not Selected>" );
        }
        break;
    }
}

```

If we dropped out of the dialog loop because of a click in the **Cancel** button, de-allocate the reminder and set it to nil. Otherwise, call `CopyDialogToReminder()` to copy the dialog fields into the `ReminderRec`.

```

if ( itemHit == cancel )
{
    DisposePtr( (Ptr)reminder );
    reminder = nil;
} else
    CopyDialogToReminder( dialog, reminder );

```



Once that's done, call `DisposDialog()` to de-allocate any memory allocated for the dialog itself, then return.

```
DisposDialog( dialog );

return( reminder );
}
```

`CopyDialogToReminder()` copies the specified dialog's fields into the `ReminderRec` pointed to by `reminder`.

```
/****** CopyDialogToReminder */

void CopyDialogToReminder( DialogPtr dialog,
ReminderPtr reminder)
{
    short itemType;
    Rect itemRect;
    Handle itemHandle;
    Str255 string;
    MenuHandle menu;
    short val;
    long tmp;
```

First, the message text is copied into the `NMRec`'s `nmStr` field.

```
GetDItem( dialog, iMessageText, &itemType,
          &itemHandle, &itemRect );
GetIText( itemHandle, reminder->alert );
reminder->notify.nmStr = (StringPtr)&reminder->alert;
```

Next, the **Play Sound** check box determines whether a `nil` or a `-1L` is stored in the `NMRec`'s `nmSound` field.

```
GetDItem( dialog, iSoundCheckBox, &itemType,
          &itemHandle, &itemRect );
if ( GetCtlValue( (ControlHandle)itemHandle ) )
    reminder->notify.nmSound = (Handle)-1L;
else
    reminder->notify.nmSound = nil;
```

The **Rotate Icon** check box determines whether the `SICN` resource is loaded into the `nmIcon` field.

```

GetDItem( dialog, iRotateCheckBox, &itemType,
          &itemHandle, &itemRect );
if( GetCtlValue( (ControlHandle)itemHandle ) )
    reminder->notify.nmIcon = GetResource( 'SICN',
                                           kBaseResID );
else
    reminder->notify.nmIcon = nil;

```

Next, the value of the **Launch**: check box is stored in the reminder's **launch** field. If the check box was checked, the function `LaunchResponse()` is set as the response procedure. Otherwise, the function `NormalResponse()` is set as the response procedure. Both are explained below.

```

GetDItem( dialog, iLaunchCheckBox, &itemType,
          &itemHandle, &itemRect );
if( reminder->launch = GetCtlValue( (ControlHandle)
                                   itemHandle ) )
    reminder->notify.nmResp = &LaunchResponse;
else
    reminder->notify.nmResp = &NormalResponse;

```

Next, the current value of the **iHoursPopup** is converted to a short and stored in the reminder's **hour** field.

```

GetDItem( dialog, iHoursPopup, &itemType, &itemHandle,
          &itemRect );
val = GetCtlValue( (ControlHandle)itemHandle );
NumToString( (long) val, string );
StringToNum ( string, &tmp );
reminder->hour = tmp;

```

Next, the hour string and a single **:** character are concatenated to form the **menuString** field.

```

reminder->menuString[0] = 0;
ConcatString( reminder->menuString, string );
ConcatString( reminder->menuString, "\p:" );

```

Next, the same is done for the minute field. First, the short is constructed.

```

GetDItem( dialog, iMinutesPopup, &itemType,
          &itemHandle, &itemRect );
val = GetCtlValue( (ControlHandle)itemHandle );
menu = GetMHandle( mMinutes );
GetItem( menu, val, string );
StringToNum ( string, &tmp );
reminder->minute = tmp;

```

Then, the string representation of the minute is concatenated onto the menuString.

```

ConcatString( reminder->menuString, string );
ConcatString( reminder->menuString, "\p " );

```

Next, the **AM/PM** menu is dealt with. If the popup is set to **PM**, the hour is bumped up by 12. After that, the appropriate value is concatenated onto the menuString.

```

GetDItem( dialog, iAMorPMPopup, &itemType,
&itemHandle, &itemRect );
val = GetCtlValue( (ControlHandle)itemHandle );

if( val == kPM )
    reminder->hour += 12;

menu = GetMenu ( mAMorPM );
GetItem( menu, val, string );
ConcatString( reminder->menuString, string );

```

Finally, the qType field is set to the type appropriate for the Notification Manager, and the nmMark field is set so the application menu will be marked when the Notification occurs.

```

reminder->notify.qType = nmType;
reminder->notify.nmMark = kMarkApp;
}

```

ConcatString() copies str2 onto the end of str1.

```

/***** ConcatString *****/

void ConcatString( Str255 str1, Str255 str2)
{
    short i;

```

```

    for (i=str1[0];i<str2[0]+str1[0];i++)
    {
        str1[i+1]=str2[i-str1[0]+1];
    }
    str1[0]=i;
}

```

The `NormalResponse()` procedure is called by the Notification Manager when the notification completes. As is the case whenever declaring a routine that will be called by the Toolbox, both `NormalResponse()` and `LaunchResponse()` are declared to be of type `pascal`. This ensures that all parameters are passed according to the Toolbox parameter-passing conventions, which differ from C's standard parameter-passing conventions.

```

/***** NormalResponse *****/
pascal void NormalResponse( NMRecPtr notifyPtr )
{
    ReminderPtr reminder;
    OSErr err;

```

`NormalResponse()` retrieves the original reminder pointer from the `nmRefCon` field, calls `NMRemove()` to remove the Notification from the notification queue, then marks the reminder for later disposal.

```

    reminder = GetReminderFromNotification( notifyPtr );
    err = NMRemove( notifyPtr );
    reminder->dispose = true;
}

```

`LaunchResponse()` also retrieves the original reminder pointer from the `nmRefCon` field.

```

/***** LaunchResponse *****/
pascal void LaunchResponse( NMRecPtr notifyPtr )
{
    LaunchParamBlockRec launchParams;
    OSErr err;
    FSSpec fileSpec;
    ReminderPtr reminder;
    Boolean isFolder;
    Boolean wasAlias;

    reminder = GetReminderFromNotification( notifyPtr );

```

The `fileSpec` specifies the application file to be launched. `ResolveAliasFile()` converts an alias (if that's what the user selected) into the real, honest-to-gosh file that was aliased.

```
fileSpec = reminder->file;

err = ResolveAliasFile( &fileSpec, true, &isFolder,
                       &wasAlias );
```

Next, the `launchParams` fields are initialized (as described in the earlier Process Manager discussion) and the application is launched. If a problem was encountered launching the application, `SysBeep()` is called.

```
launchParams.launchBlockID = extendedBlock;
launchParams.launchEPBLength = extendedBlockLen;
launchParams.launchFileFlags = 0;
launchParams.launchControlFlags = launchContinue +
    launchNoFileFlags;
launchParams.launchAppSpec = &fileSpec;

launchParams.launchAppParameters = nil;

if ( LaunchApplication( &launchParams ) ) SysBeep
    ( 20 );
```

Finally, `NMRemove()` is called to remove the Notification from the notification queue, and the reminder is marked for later disposal.

```
err = NMRemove( notifyPtr );

reminder->dispose = true;
}
```

`GetReminderFromNotification()` returns the value stored in the `nmRefCon` field.

```
/****** GetReminderFromNotification *****/

ReminderPtr      GetReminderFromNotification( NMRecPtr
notifyPtr )
{
    return (ReminderPtr) notifyPtr->nmRefCon;
}
```

## In Review

---

Like menus and windows, dialogs and alerts are an intrinsic part of the Macintosh interface. It is very important that you read through Apple's interface guidelines, found in *Inside Macintosh*, Volume VI, Chapter 2. Get off to a good start by learning the proper way to design your interface elements.

In Chapter 7, we'll address some of the programming issues we've not touched on yet, such as error handling, using the clipboard, file management, printing, and scrolling. We'll even take a brief sojourn into the Macintosh Sound Manager.

Congratulations! The toughest part of the book is behind you.

# Toolbox Potpourri

*Congratulations! Now that you have  
the Macintosh interface under your  
belt, you're ready to add some more  
advanced Toolbox features to your  
programs.*

CHAPTER 7 PRESENTS six programs, each of which explores a new part of the Toolbox. The first program, **ResWriter**, shows you the proper way to load, modify, and then write a resource back out to your application's resource fork.

The second program, **Pager**, uses a scroll bar to scroll through a list of pictures. Next, the desk scrap, more commonly known as the Clipboard, is introduced. The Scrap Manager utilities that support cut, copy, and paste operations are discussed. The third program, **ShowClip**, uses these routines to display the current scrap in a window.

The fourth application, **SoundMaker**, takes advantage of System 7's built-in sound recording and playback features. The fifth application, **OpenPICT**, demonstrates the System-7-savvy method for opening and reading from a standard Macintosh file. **OpenPICT** uses the standard file mechanism to select a PICT file, reads in the picture, then draws the picture in a window.

Chapter 7's final program, **PrintPICT**, loads a PICT from the resource fork, then works with the Printing Manager to print the file on the currently selected printer.

---

## Writing Out Resources

---

So far, the relationship between programs and resources has been one way only. You've learned how to load a resource from the resource fork. Our next program, **ResWriter**, shows you how to change the resource and write it back out.

The ability to change a resource and write it back out to the resource file is extremely important. Suppose you wrote an application that used a WIND resource to determine the initial size and position of a window. Imagine that just before your program quit, you could record the current window size and position, writing over whatever values were currently stored in the WIND resource. This means that the next time the user opened your application, their last window size and position will have been preserved.

---

## ResWriter

---

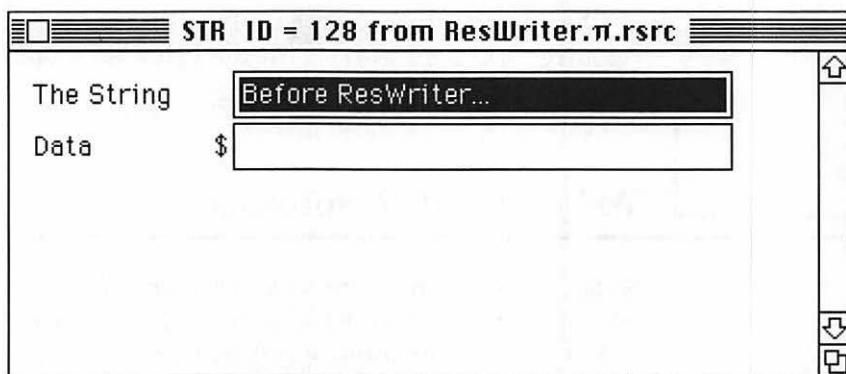
**ResWriter** demonstrates the technique of loading, changing, and writing a resource back out. Though **ResWriter** works with a resource of type 'STR ', the same approach will work for any resource, as long as you know the format of the resource. A 'STR ' resource is stored as a Pascal string—a length byte followed by that many bytes of text.



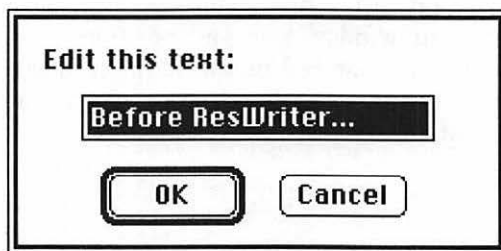


You can find the format for most resource types in the pages of *Inside Macintosh*. For example, the format of the `WIND` resource is detailed on I:302.

ResWriter starts by loading a 'STR' resource, like the one shown in Figure 7.1. ResWriter then copies the text string into an editable text field in a dialog box (Figure 7.2), then lets the dialog fly. When the dialog's **OK** button is pressed, any changes made to the text are written back out to the original 'STR' resource.



**Figure 7.1** ResWriter's 'STR' resource.



**Figure 7.2** ResWriter's dialog box.

## ResWriter Resources

Create a new folder called `ResWriter` in the `Development` folder. Next, use `ResEdit` to create a new resource file called `ResWriter.π.rsrc` inside the `ResWriter` folder. `ResWriter` makes use of three resources: a DITL, a DLOG, and a 'STR'.

Select **Create New Resource** from the **Resource** menu and create a DITL resource. Use the specifications in Figure 7.3 to create the four items that make up your DITL. Finally, select **Get Resource Info** from the **Resource** menu, change the DITL's resource ID to 128 and make sure the **Purgeable** check box is checked.

The figure displays three sequential screenshots of the 'Edit DITL item' dialog boxes for items #1, #2, and #3 in the resource file `ResWriter.π.rsrc`.

**Item #1:** The dialog is titled 'Edit DITL item #1 from ResWriter.π.rsrc'. It shows a 'Text' field with the value 'OK'. The 'Button' type is selected from the dropdown menu. The 'Enabled' checkbox is checked. The coordinates are Top: 70, Left: 40, Height: 20, and Width: 60.

**Item #2:** The dialog is titled 'Edit DITL item #2 from ResWriter.π.rsrc'. It shows a 'Text' field with the value 'Cancel'. The 'Button' type is selected from the dropdown menu. The 'Enabled' checkbox is checked. The coordinates are Top: 70, Left: 120, Height: 20, and Width: 60.

**Item #3:** The dialog is titled 'Edit DITL item #3 from ResWriter.π.rsrc'. It shows a 'Text' field with the value 'Edit this text:'. The 'Static Text' type is selected from the dropdown menu. The 'Enabled' checkbox is unchecked. The coordinates are Top: 8, Left: 8, Height: 20, and Width: 108.

**Figure 7.3** Specifications for the four items in DITL 128.

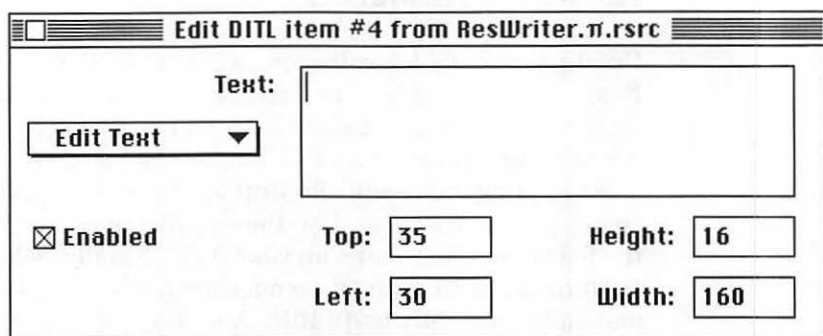


Figure 7.3 Specifications for the four items in DITL 128 (continued).

After closing all the windows associated with your DITL, select **Create New Resource** from the **Resource** menu and create a DLOG resource. Use the specifications in Figure 7.4 to create the DLOG. If the **Bottom:** and **Right:** fields don't appear in your DLOG editor, select **Show Bottom & Right** from the **DLOG** menu. Make sure you enter 128 in the **DITL ID:** field. Next, change the DLOG's resource ID to 128 and make sure the **Purgeable** check box is checked. Finally, select **Auto Position...** from the **DLOG** menu and make sure the settings match those in Figure 7.5.

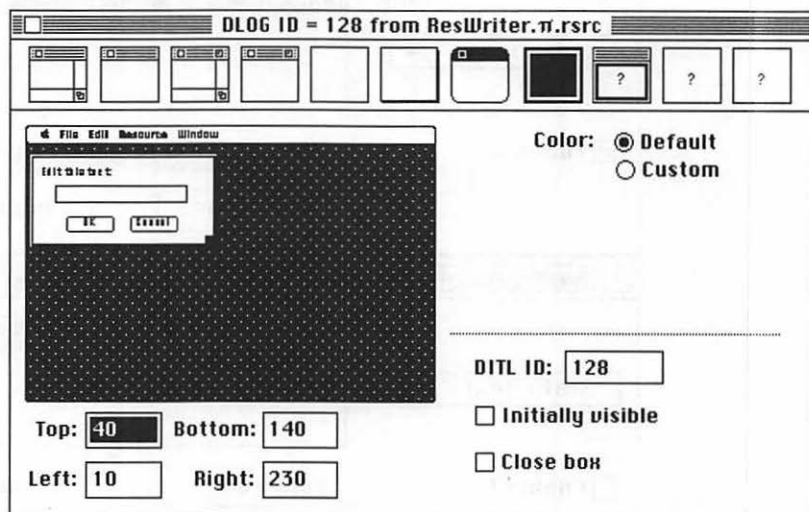
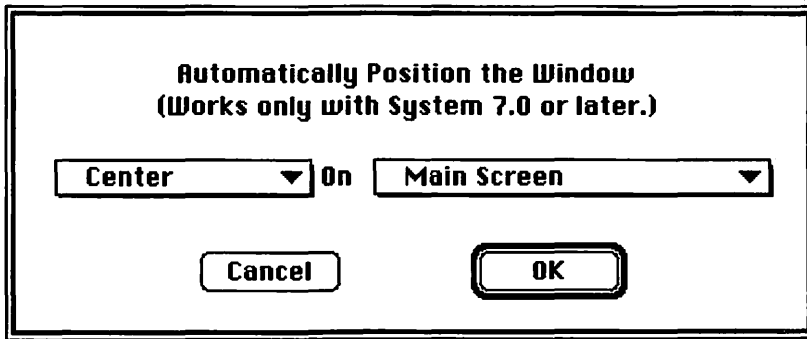


Figure 7.4 Specifications for the DLOG resource.



**Figure 7.5** The Auto Position... dialog box.

After closing all the windows associated with your DLOG, select **Create New Resource** from the **Resource** menu and create a 'STR ' resource. Be sure to include the space following the STR when specifying the resource type. Use the specification in Figure 7.1 to create the 'STR '. Finally, change the resource ID of the 'STR ' to 128 and make sure the **Purgeable** check box is checked.

Quit ResEdit, saving your changes.

## The ResWriter Project

Go into THINK C and create a new project called ResWriter.π inside the ResWriter folder. Use **Add...** from the **Source** menu to add MacTraps to the project.

Once MacTraps is added, open a new source code window and enter the program:

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

#define iText           4

#define kDisableButton  255
#define kEnableButton   0

#define kWriteTextOut   true
#define kDontWriteTextOut false

#define kMinTextPosition 0
#define kMaxTextPosition 32767
```

```

/*****/
/*  Functions  */
/*****/

void      ToolBoxInit( void );
Boolean   DoTextDialog( StringHandle oldTextHandle );

pascal OSErr SetDialogDefaultItem( DialogPtr theDialog,
                                   short newItem ) = { 0x303C, 0x0304, 0xAA68 };
pascal OSErr SetDialogCancelItem( DialogPtr theDialog,
                                   short newItem ) = { 0x303C, 0x0305, 0xAA68 };
pascal OSErr SetDialogTracksCursor( DialogPtr theDialog,
                                   Boolean tracks ) = { 0x303C, 0x0306, 0xAA68 };

/***** main *****/

void      main( void )
{
    StringHandle textHandle;

    ToolBoxInit();

    textHandle = GetString( kBaseResID );

    if ( textHandle == nil )
    {
        SysBeep( 20 );
        ExitToShell();
    }

    if ( DoTextDialog( textHandle ) == kWriteTextOut )
    {
        ChangedResource( (Handle)textHandle );
        WriteResource( (Handle)textHandle );
    }
}

/***** ToolBoxInit *****/

void      ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();

```

```
InitWindows();
InitMenus();
TEInit();
InitDialogs( nil );
InitCursor();
}

/***** DoTextDialog *****/

Boolean DoTextDialog( StringHandle textHandle )
{
    DialogPtr    dialog;
    Boolean      done;
    short        itemHit, itemType;
    Handle       OKItemHandle, textItemHandle;
    Rect         itemRect;
    Str255       itemText;

    dialog = GetNewDialog( kBaseResID, nil, kMoveToFront );

    GetDItem( dialog, ok, &itemType, &OKItemHandle,
              &itemRect );
    GetDItem( dialog, iTText, &itemType, &textItemHandle,
              &itemRect );

    HLock( (Handle)textHandle );
    SetIText( textItemHandle, *textHandle );
    HUnlock( (Handle)textHandle );

    SelIText( dialog, iTText, kMinTextPosition,
              kMaxTextPosition );

    ShowWindow( dialog );
    SetPort( dialog );

    SetDialogDefaultItem( dialog, ok );
    SetDialogCancelItem( dialog, cancel );
    SetDialogTracksCursor( dialog, true );

    done = false;
    while ( ! done )
    {
        GetIText( textItemHandle, itemText );
```

```

        if ( itemText[ 0 ] == 0 )
            HiliteControl( (ControlHandle)OKItemHandle,
                           kDisableButton );
        else
            HiliteControl( (ControlHandle)OKItemHandle,
                           kEnableButton );

        ModalDialog( nil, &itemHit );

        done = ( (itemHit == ok) || (itemHit == cancel) );
    }

    if ( itemHit == ok )
    {
        GetIText( textItemHandle, itemText );
        SetHandleSize( (Handle)textHandle,
                       (Size)(itemText[ 0 ] + 1) );

        HLock( (Handle)textHandle );
        GetIText( textItemHandle, *textHandle );
        HUnlock( (Handle)textHandle );

        DisposDialog( dialog );

        return( kWriteTextOut );
    }
    else
    {
        DisposDialog( dialog );

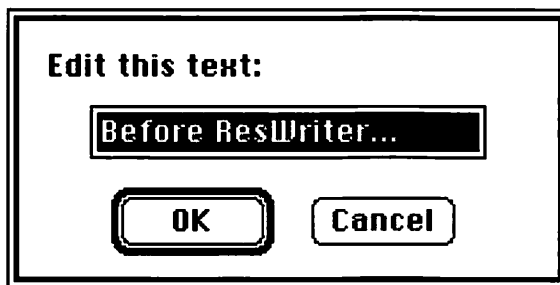
        return( kDontWriteTextOut );
    }
}

```

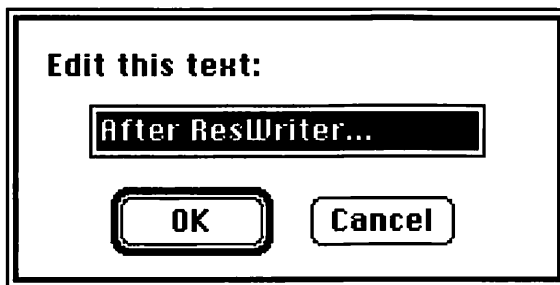
## Running ResWriter

Once you've finished typing in the code, save it as `ResWriter.c` and add it to the project using **Add** from the **Source** menu. Next, select **Run** from the **Project** menu, clicking **Yes** to the question **Bring the project up to date?** If the source code compiles correctly, a dialog box similar to that shown in Figure 7.6 should appear.

Notice that the dialog's entire text string is highlighted. Hit the delete key, leaving the text field empty. Notice that the **OK** button gets dimmed. Type the string **After ResWriter...** and press the **OK**



**Figure 7.6** ResWriter in action.



**Figure 7.7** ResWriter after the edited 'STR' resource is retrieved.

button. ResWriter will write your edited text string out to the 'STR' resource and then exit. Run ResWriter again. The text you typed should appear in the dialog box (Figure 7.7).

Click **OK** to exit ResWriter. Let's take a look at the code.

---

## Walking Through the ResWriter Code

---

As usual, ResWriter starts off with some `#defines`. The first two should look familiar.

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L
```

`iText` represents the item number of the dialog's textedit field. `kDisableButton` and `kEnableButton` represent the two values we'll pass to `HiliteControl()` to either enable or disable the **OK** button. `kWriteTextOut` and `kDontWriteTextOut` represent the two possible



return values of `DoTextDialog()`. They'll tell us whether we need to write the 'STR' resource back out. Finally, `kMinTextPosition` and `kMaxTextPosition` are used to force highlighting of all text in the `textedit` field in the call to `SetIText()`.

```
#define iText                4

#define kDisableButton       255
#define kEnableButton        0

#define kWriteTextOut        true
#define kDontWriteTextOut    false

#define kMinTextPosition     0
#define kMaxTextPosition     32767
```

Next come `ResWriter`'s function prototypes.

```
/* ***** */
/*  Functions  */
/* ***** */

void      ToolBoxInit( void );
Boolean   DoTextDialog( StringHandle oldTextHandle );
```

The three functions declared next are part of the System 7 Toolbox. Unfortunately, these declarations have not yet made their way into THINK C. If you are using a newer version of THINK C that contains these declarations, feel free to leave these out.

`SetDialogDefaultItem()` allows you to specify a dialog's default item (usually the **OK** button). This function automatically draws the thick, rounded rectangle around the default item. `SetDialogCancelItem()` allows you to specify a dialog's cancel item. After this call is made, typing **⌘**. (Command-period) automatically selects the specified cancel item. Finally, `SetDialogTracksCursor()` enables the Dialog Manager to tie the I-beam cursor to any editable text items in a dialog.

```
pascal OSErr SetDialogDefaultItem( DialogPtr theDialog,
                                   short newItem ) = { 0x303C, 0x0304, 0xAA68 };
pascal OSErr SetDialogCancelItem( DialogPtr theDialog,
                                   short newItem ) = { 0x303C, 0x0305, 0xAA68 };
```

```
pascal OSErr SetDialogTracksCursor( DialogPtr theDialog,
                                   Boolean tracks ) = { 0x303C, 0x0306, 0xAA68 };
```

`main()` starts with a call to `ToolBoxInit()`.

```
/****** main *****/
```

```
void    main( void )
{
    StringHandle textHandle;
    ToolBoxInit();
```

Next, `GetString()` is called to retrieve the 'STR' resource with resource ID `kBaseResID`. `GetString()` returns a handle to a 'STR' resource, just as `GetPicture()` returns a handle to a PICT resource.

```
textHandle = GetString( kBaseResID );
```

If the 'STR' resource wasn't found, `ResWriter` beeps once, then exits. If this happens, chances are either your resource file wasn't found or your 'STR' resource had the wrong ID.

```
if ( textHandle == nil )
{
    SysBeep( 20 );
    ExitToShell();
}
```

Next, `DoTextDialog()` gets called, bringing up the `ResWriter` dialog. If `DoTextDialog()` returns a value of `kWriteTextOut`, the resource will be written out. The rules for writing out a resource are simple. First, call `ChangedResource()` to mark the resource as changed. `WriteResource()` will write the resource back out only if it was marked as changed.

```
if ( DoTextDialog( textHandle ) == kWriteTextOut )
{
    ChangedResource( (Handle)textHandle );
    WriteResource( (Handle)textHandle );
}
}
```



Whenever you call one of the Resource Manager Toolbox routines, it's a good idea to follow it up with a call to **ResError()**. **ResError()** checks the status of the last Resource Manager function called and returns an appropriate status message. The meaning of the status message depends on the function called.

As you develop an error-handling strategy for your program, you'll want to incorporate the possible errors returned by **ResError()**. To find out more, read *Inside Macintosh*, Volume I, Chapter 5.

**ToolBoxInit()** remains the same as in earlier programs.

```

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

**DoTextDialog()** handles the ResWriter dialog.

```

/***** DoTextDialog *****/

Boolean DoTextDialog( StringHandle textHandle )
{
    DialogPtr    dialog;
    Boolean      done;
    short        itemHit, itemType;
    Handle       OKItemHandle, textItemHandle;
    Rect         itemRect;
    Str255       itemText;
}

```

First, **GetNewDialog()** is called to load the DLOG resource.

```

dialog = GetNewDialog( kBaseResID, nil, kMoveToFront );

```

Next, `GetDItem()` is used to retrieve a handle to the **OK** button and textedit field items, placing the handles in the variables `OKItemHandle` and `textItemHandle`.

```
GetDItem( dialog, ok, &itemType, &OKItemHandle,  
          &itemRect );  
GetDItem( dialog, iText, &itemType, &textItemHandle,  
          &itemRect );
```

Next, `textHandle`, the handle to the 'STR ' resource, is locked. This is done so the handle can be singly dereferenced. Remember, handles are pointers to pointers. If you ever want to singly dereference a handle, make sure you lock it first!

```
HLock( (Handle)textHandle );
```

Once locked, `SetIText()` is called to copy the text handled by `textHandle` to the item handled by `textItemHandle`. Basically, this means that the text in the 'STR ' resource is copied into the dialog's textedit item.

```
SetIText( textItemHandle, *textHandle );  
HUnlock( (Handle)textHandle );
```

Next, `SelIText()` is called to highlight the entire text string.

```
SelIText( dialog, iText, kMinTextPosition,  
          kMaxTextPosition );
```

Now that the dialog is properly set up, make it visible and make it the current port.

```
ShowWindow( dialog );  
SetPort( dialog );
```

Call the three functions described at the top of the file.

```
SetDialogDefaultItem( dialog, ok );  
SetDialogCancelItem( dialog, cancel );  
SetDialogTracksCursor( dialog, true );
```

Next, enter the main dialog loop. At the top of the loop, retrieve the text in the textedit item, placing the text into the variable `itemText`.

```
done = false;
while ( ! done )
{
    GetIText( textItemHandle, itemText );
```

If the textedit field is empty, disable the **OK** button, otherwise enable the **OK** button.

```
if ( itemText[ 0 ] == 0 )
    HiliteControl( (ControlHandle)OKItemHandle,
                  kDisableButton );
else
    HiliteControl( (ControlHandle)OKItemHandle,
                  kEnableButton );
```

Next, call `ModalDialog()`, using `itemHit` to determine if the **OK** or **Cancel** buttons were hit. If so, drop out of the loop.

```
ModalDialog( nil, &itemHit );

done = ( (itemHit == ok) || (itemHit == cancel) );
}
```

If the **OK** button was hit, `GetIText()` is called to retrieve the dialog's text, placing the string into the variable `itemText`. Note that this string may be a different length than the text in the original 'STR' resource. For example, the word "Before" requires 7 bytes, while the word "After" requires only 6 bytes. When the resource was first loaded, exactly the right amount of memory was allocated to hold the resource. Since the size of the text string may have changed, we'll use `SetHandleSize()` to force the handle to refer to a block of the proper size. The Memory Manager takes care of all the details.

```
if ( itemHit == ok )
{
    GetIText( textItemHandle, itemText );
    SetHandleSize( (Handle)textHandle,
                  (Size)(itemText[ 0 ] + 1) );
```



If you're working with a resource of a fixed size (such as an **ICON**), you won't need to call `SetHandleSize()`, since the resource will always be of the right size. On the other hand, if you're not sure whether a resource is of a fixed size, call `SetHandleSize()` just to be safe.

Now that we have a block of memory of the correct size, lock the resource handle, call `GetIText()` to load the text into the newly sized resource, and unlock the handle again.

```
HLock( (Handle)textHandle );
GetIText( textItemHandle, *textHandle );
HUnlock( (Handle)textHandle );
```

Finally, dispose of the dialog and return the proper value.

```
DisposDialog( dialog );

return( kWriteTextOut );
}
```

If the **Cancel** button was hit, dispose of the dialog and return the appropriate value.

```
else
{
    DisposDialog( dialog );

    return( kDontWriteTextOut );
}
}
```

The subject of handles and Macintosh memory management in general is a complex one. To learn more, check out the Toolbox Techniques chapter in Volume II of the *Mac Primer*.

---

## Scroll Bars! We're Gonna Do Scroll Bars!

---

**Scroll bars** are a common control used in Macintosh applications. This section shows you how to set one up to control paging between a series of pictures in a window.

### Making Use of Scroll Bars

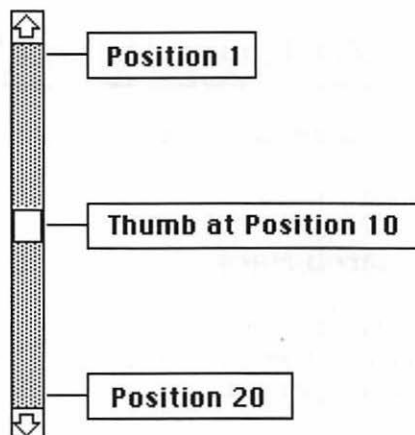
Scroll bars are a common control used in Macintosh applications (Figure 7.8). The routines that create and control scroll bars are part of the Control Manager. The function `NewControl()` is used to create a new control:



**Figure 7.8** A Macintosh window with a scroll bar.

```
pascal ControlHandle NewControl( WindowPtr theWindow,  
                                const Rect *boundsRect, ConstStr255Param title,  
                                Boolean visible, short value, short min, short max,  
                                short procID, long refCon );
```

The parameter `procID` specifies the type of control to be created. To create a new scroll bar, pass the constant `scrollBarProc` to `NewControl()`. Every scroll bar has a minimum, maximum, and current value. For example, a scroll bar may go from 1 to 20, and currently be at 10 (Figure 7.9).



**Figure 7.9** Scroll bar positioning.

Once the scroll bar is created, call `DrawControls()` to draw it in your window:

```
pascal void DrawControls( WindowPtr theWindow );
```



Since Window Manager routines, such as `ShowWindow()` and `MoveWindow()`, don't automatically redraw controls in a window, you'll want to call `DrawControls()` whenever the window receives an update event.

Typically, when a `mouseDown` event occurs, `FindWindow()` gets called. `FindWindow()` returns a part code describing the part of the window in which the `mouseDown` occurred. If the `mouseDown` was inContent, call `FindControl()`:

```
pascal short FindControl( Point thePoint,  
                          WindowPtr theWindow, ControlHandle *theControl );
```

Like `FindWindow()`, `FindControl()` returns a part code. This time, the part code specifies which part of the scroll bar was clicked in (Figure 7.10). Pass the part code returned by `FindControl()` to `TrackControl()`:

```
pascal short TrackControl( ControlHandle theControl,  
                           Point thePoint, ProcPtr actionProc);
```

`TrackControl()` will perform the action appropriate to that part of the scroll bar. For example, if the `mouseDown` was in the thumb of the scroll bar, an outline of the thumb is moved up and down (or across) the scroll bar until the mouse button is released. Once `TrackControl()` returns, take the appropriate action, depending on the new value of the scroll bar.

Next, let's look at **Pager**, a program that uses a scroll bar to page through a series of PICT resources.



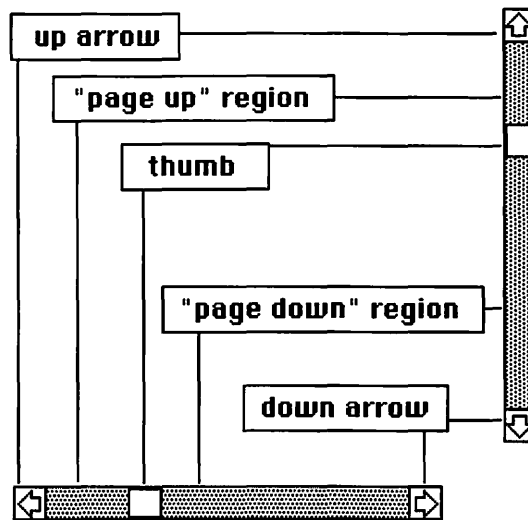


Figure 7.10 Scroll bar anatomy.

## Pager

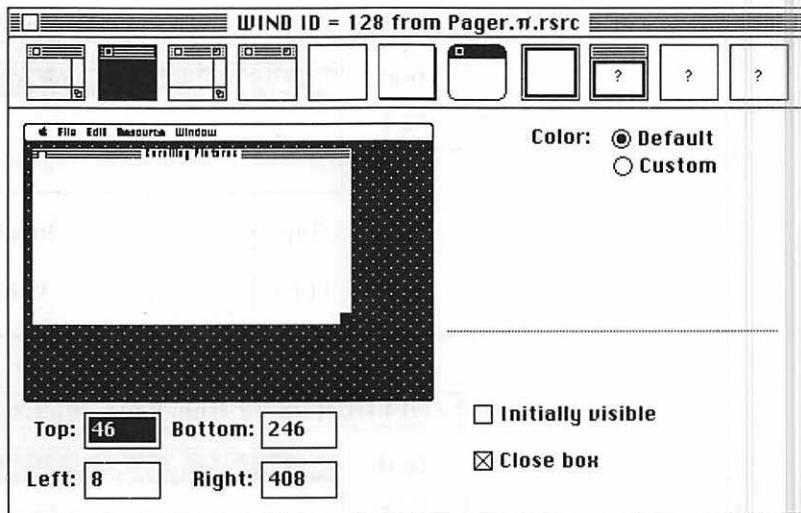
Pager demonstrates the use of scroll bars in a Macintosh application. Here's the Pager algorithm:

- The Toolbox is initialized.
- A scroll bar is created, using the number of available PICT resources to determine the maximum value of the scroll bar.
- When a mouseDown occurs in the scroll bar, Pager loads the appropriate PICT and displays the PICT in the window.
- Pager quits when the close box is clicked.

Pager also introduces a basic error-handling mechanism, the `DoError()` function.

## Pager Resources

Create a folder called Pager inside your Development folder. Use ResEdit to create a new file called `Pager.π.rsrc` inside the Pager folder. Create a WIND resource according to the specifications in Figure 7.11. Select **Set 'WIND' Characteristics...** from the **WIND**



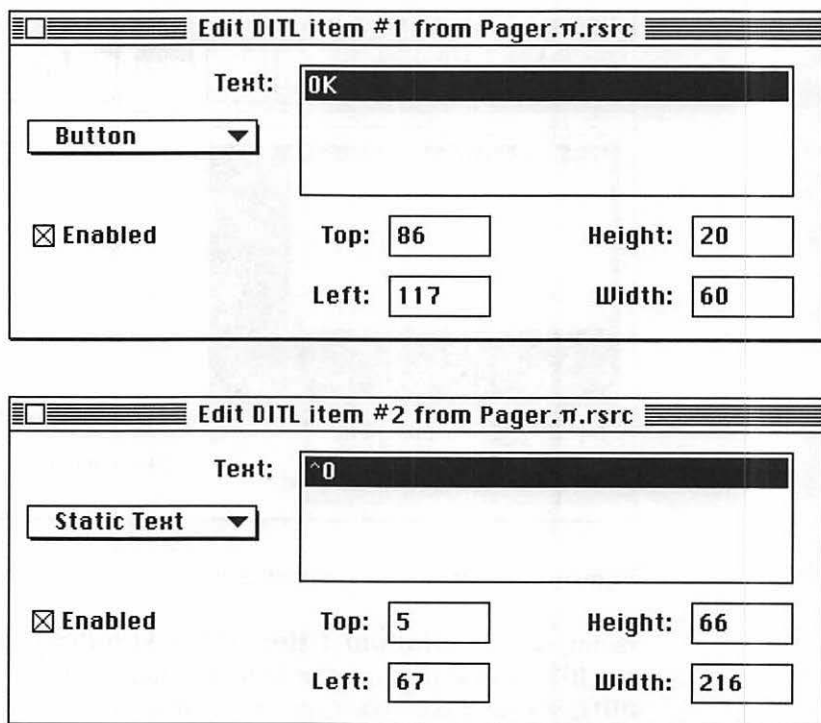
**Figure 7.11** Pager's WIND specifications.

menu, set the **Window title:** field to **Scrolling Pictures**, and click the **OK** button. Change the WIND's resource ID to 128 and check the **purgeable** check box. Close the windows associated with the WIND resource.

Next, create a two-item DITL resource using the specifications in Figure 7.12. Be sure to enter the text **^0** (carat-zero) into the **Static Text** item's **Text:** field. Our error-handling function will call `ParamText()` to substitute the current error message for the **^0**.

Change the DITL's resource ID to 128 and check the **purgeable** check box. Close any windows associated with the DITL.

Next, use the specifications in Figure 7.13 to create an ALRT resource. Make sure to set the **DITL ID:** field to 128. Change the ALRT's resource ID to 128 and check the **purgeable** check box. Then, select **Auto Position...** from the **ALRT** menu, and set the left pop-up to **Alert Position** and the right pop-up to **Main Screen**. Click the **OK** button, then close any windows associated with the ALRT.



**Edit DITL item #1 from Pager.π.rsrc**

Text:

Button

☒ Enabled

Top:  Height:

Left:  Width:

**Edit DITL item #2 from Pager.π.rsrc**

Text:

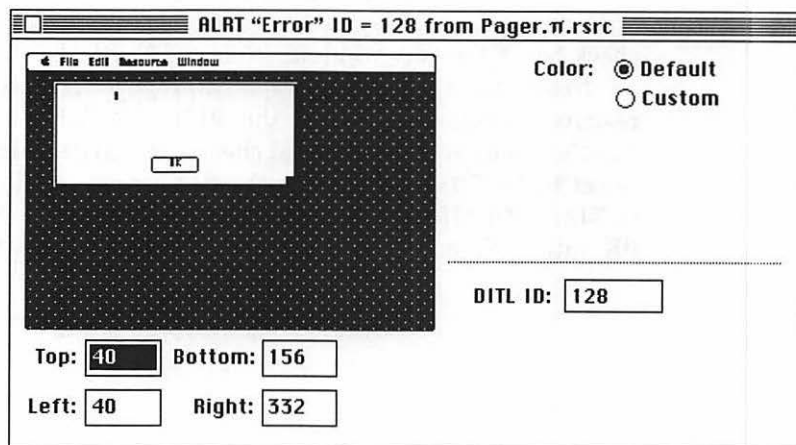
Static Text

☒ Enabled

Top:  Height:

Left:  Width:

**Figure 7.12** Specifications for Pager's DITL. Be sure to enter the text ^0 for the Static Text item.



**ALRT "Error" ID = 128 from Pager.π.rsrc**

File Edit Resource Window

Color: ☒ Default ☐ Custom

DITL ID:

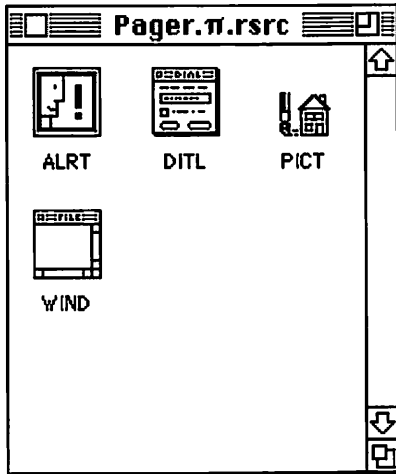
Top:  Bottom:

Left:  Right:

**Figure 7.13** Specifications for Pager's ALRT resource.

Finally, create some PICT resources from your favorite clip art and paste them into the `Pager.π.rsrc`. Paste in as many as you like. Don't worry about changing resource IDs for the PICT resources. Pager will display every available PICT, regardless of race, creed, or resource ID. When you're done, the resource window of `Pager.π.rsrc` should look like Figure 7.14.

Quit ResEdit, saving your changes.



**Figure 7.14** `Pager.π.rsrc`, once all the resources have been created.

## The Pager Project

Now you're ready to launch THINK C. Create a new project in the Pager folder. Call it `Pager.π`. Use **Add...** from the **Source** menu to add MacTraps to the project.

Once MacTraps is added, open a new source code window and enter the program:

```
#include <Values.h>

#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L
#define kScrollBarWidth 16
#define kNilActionProc  nil
#define kSleep           MAXLONG
```

```

#define kVisible            true
#define kStartValue        1
#define kMinValue          1
#define kNilRefCon         0L
#define kEmptyTitle        "\p"

#define kEmptyString        "\p"
#define kNilFilterProc     nil

#define kErrorAlertID      kBaseResID

/*****/
/*  Globals  */
/*****/

Boolean            gDone;

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
void    WindowInit( void );
void    SetUpScrollBar( WindowPtr window );
pascal void ScrollProc( ControlHandle theControl, short
                        partCode );
void    EventLoop( void );
void    DoEvent( EventRecord *eventPtr );
void    HandleMouseDown( EventRecord *eventPtr );
void    UpdateWindow( WindowPtr window );
void    CenterPict( PicHandle picture, Rect *destRectPtr );
void    DoError( Str255 errorString, Boolean fatal );

/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();

    EventLoop();
}

```

```

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    if ( ( window = GetNewWindow( kBaseResID, nil,
                                kMoveToFront ) ) == nil )
        DoError( "\pCan't Load WIND resource!", true );

    SetUpScrollBar( window );

    ShowWindow( window );
    SetPort( window );
}

/***** SetUpScrollBar *****/

void    SetUpScrollBar( WindowPtr window )
{
    Rect            vScrollRect;
    short           numPictures;
    ControlHandle    scrollBarH;

    if ( ( numPictures = CountResources( 'PICT' ) ) <= 0 )
        DoError( "\pNo PICT resources were found!", true );
}

```

```

vScrollRect = window->portRect;
vScrollRect.top -= 1;
vScrollRect.bottom += 1;
vScrollRect.left = vScrollRect.right -
    kScrollBarWidth + 1;
vScrollRect.right += 1;

scrollBarH = NewControl( window, &vScrollRect,
    kEmptyTitle, kVisible, kStartValue, kMinValue,
    numPictures, scrollBarProc, kNilRefCon );
}

/***** ScrollProc *****/

pascal void ScrollProc( ControlHandle theControl, short
    partCode )
{
    short          curCtlValue, maxCtlValue, minCtlValue;
    WindowPtr      window;

    maxCtlValue = GetCtlMax( theControl );
    curCtlValue = GetCtlValue( theControl );
    minCtlValue = GetCtlMin( theControl );

    window = (**theControl).ctrlOwner;

    switch ( partCode )
    {
        case inPageDown:
        case inDownButton:
            if ( curCtlValue < maxCtlValue )
            {
                curCtlValue += 1;
                SetCtlValue( theControl, curCtlValue );
                UpdateWindow( window );
            }
            break;
        case inPageUp:
        case inUpButton:
            if ( curCtlValue > minCtlValue )
            {
                curCtlValue -= 1;
                SetCtlValue( theControl, curCtlValue );
            }
    }
}

```

```
        UpdateWindow( window );
    }
}

/***** EventLoop *****/

void    EventLoop( void )
{
    EventRecord    event;

    gDone = false;

    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, kSleep,
                           nil ) )
            DoEvent( &event );
    }
}

/***** DoEvent *****/

void    DoEvent( EventRecord *eventPtr )
{
    WindowPtr    window;

    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
        case updateEvt:
            window = (WindowPtr)eventPtr->message;

            BeginUpdate( window );
            DrawControls( window );
            UpdateWindow( window );
            EndUpdate( window );
            break;
    }
}
```



```

/***** HandleMouseDown *****/

void      HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr      window;
    short           thePart;
    Point           thePoint;
    ControlHandle    theControl;

    thePart = FindWindow( eventPtr->where, &window );
    switch ( thePart )
    {
        case inSysWindow :
            SystemClick( eventPtr, window );
            break;
        case inDrag :
            DragWindow( window, eventPtr->where,
                        &screenBits.bounds );
            break;
        case inContent:
            thePoint = eventPtr->where;
            GlobalToLocal( &thePoint );

            thePart = FindControl( thePoint, window,
                                   &theControl );

            if ( theControl ==
                ((WindowPeek)window)->controlList )
            {
                if ( thePart == inThumb )
                {
                    thePart = TrackControl( theControl,
                                             thePoint, kNilActionProc );
                    InvalRect( &(window->portRect) );
                }
                else
                    thePart = TrackControl( theControl,
                                             thePoint, &ScrollProc );
            }
            break;
        case inGoAway :
            gDone = true;
            break;
    }
}

```

```

/***** UpdateWindow *****/

void      UpdateWindow( WindowPtr window )
{
    PicHandle      currentPicture;
    Rect           windowRect;
    RgnHandle      tempRgn;

    tempRgn = NewRgn();
    GetClip( tempRgn );

    windowRect = window->portRect;
    windowRect.right -= kScrollBarWidth;
    EraseRect( &windowRect );

    ClipRect( &windowRect );

    currentPicture = (PicHandle)GetIndResource( 'PICT',
        GetCtlValue( ((WindowPeek)window)->
            controlList ) );

    if ( currentPicture == nil )
        DoError( "\pCan't Load PICT resource!", true );

    CenterPict( currentPicture, &windowRect );
    DrawPicture( currentPicture, &windowRect );

    SetClip( tempRgn );
    DisposeRgn( tempRgn );
}

/***** CenterPict *****/

void      CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
        windRect.top - pictRect.top );
    OffsetRect( &pictRect, (windRect.right -
        pictRect.right)/2, (windRect.bottom -
        pictRect.bottom)/2 );
    *destRectPtr = pictRect;
}

```

```
/****** DoError *****/  
  
void DoError( Str255 errorString, Boolean fatal )  
{  
    ParamText( errorString, kEmptyString, kEmptyString,  
               kEmptyString );  
  
    StopAlert( kErrorAlertID, kNilFilterProc );  
  
    if ( fatal )  
        ExitToShell();  
}
```

## Running Pager

Once you've finished typing in the code, save it as `Pager.c` and add it to the project using **Add** from the **Source** menu. Next, select **Run** from the **Project** menu, clicking **Yes** to the question **Bring the project up to date?** If the source code compiles correctly, a scrolling window similar to the one shown in Figure 7.15 should appear.

Drag the window around on the screen. Click on the arrows at each end of the scroll bar. Pager will scroll through your pictures one at a time. Click in the gray paging regions. This will cause Pager to scroll as well. Click and drag the scroll bar's thumb. Releasing the thumb at the top of the scroll bar will move Pager to the first picture. Dragging the thumb to the bottom of the scroll bar will move Pager to the last picture. Any position in between will scroll to the appropriate picture.

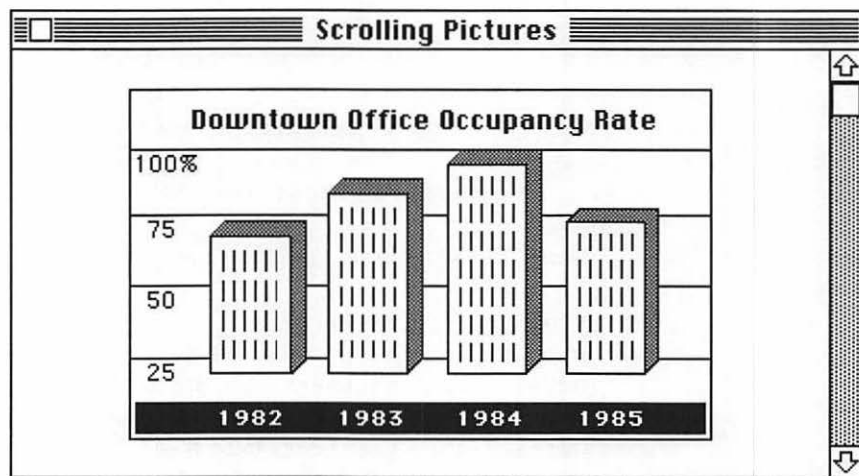


Figure 7.15 Pager in action.



You may have noticed a few unexpected pictures in your scrolling window. Here's why.

Every application has access to resources from two different places: the resource fork of the application itself, and the resource fork of the system file. In addition, an application may use the Resource Manager to open additional resource files. When looking for a resource, the Resource Manager searches the most recently opened resource file first.

When you're done admiring your handiwork, click on the close box to exit Pager. Let's take a look at the code.

## Walking Through the Pager Code

Pager starts with a single `#include`. `Values.h` contains the `#define` `MAXLONG`, defined as the largest possible long. We use `MAXLONG` to `#define` `kSleep`, maximizing our `WaitNextEvent()` sleep time. `kBaseResID` and `kMoveToFront` should look familiar. `kScrollBarWidth` defines the width of a scroll bar in pixels. `kNilActionProc` is passed to `TrackControl()` a bit later in the program.

```
#include <Values.h>

#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L
#define kScrollBarWidth 16
#define kNilActionProc  nil
#define kSleep          MAXLONG
```

The next five `#defines` are used in the call to `NewControl()` in the function `SetUpScrollBar()`.

```
#define kVisible        true
#define kStartValue     1
#define kMinValue       1
#define kNilRefCon      0L
#define kEmptyTitle     "\p"
```

kEmptyString is used as a parameter to ParamText() and kNilFilterProc and kErrorAlertID are used as parameters to StopAlert(). Both of these calls are made in the function DoError().

```
#define kEmptyString      "\p"
#define kNilFilterProc    nil

#define kErrorAlertID     kBaseResID
```

gDone plays its usual role, triggering the end of the main event loop.

```
/* *****
 * Globals
 * *****
 */
```

```
Boolean      gDone;
```

Next come Pager's function prototypes.

```
/* *****
 * Functions
 * *****
 */
```

```
void      ToolBoxInit( void );
void      WindowInit( void );
void      SetUpScrollBar( WindowPtr window );
pascal void ScrollProc( ControlHandle theControl, short
                        partCode );
void      EventLoop( void );
void      DoEvent( EventRecord *eventPtr );
void      HandleMouseDown( EventRecord *eventPtr );
void      UpdateWindow( WindowPtr window );
void      CenterPict( PicHandle picture, Rect *destRectPtr );
void      DoError( Str255 errorString, Boolean fatal );
```

main() calls ToolBoxInit() and WindowInit(), then enters the main event loop.

```

/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();

    EventLoop();
}

```

Nope. Still looks the same.

```

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

WindowInit() loads the WIND resource. If the resource wasn't found, an appropriate error message is passed to DoError(). Next, SetUpScrollBar() creates the Pager scroll bar. ShowWindow() makes the window visible, and SetPort() makes it the current port.

```

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    if ( ( window = GetNewWindow( kBaseResID, nil,
                                   kMoveToFront ) ) == nil )
        DoError( "\pCan't Load WIND resource!", true );

    SetUpScrollBar( window );

    ShowWindow( window );
    SetPort( window );
}

```

`SetUpScrollBar()` calls `CountResources()` to find out how many PICT resources are available.



To work with PICTs in the current resource file only, use the functions `Count1Resources()` and `Get1IndResource()`, described in (IV:15). `Count1Resources()` and `Get1IndResource()` are identical to the functions `CountResources()` and `GetIndResource()` in every other aspect.

```

/*****SetUpScrollBar *****/

void      SetUpScrollBar( WindowPtr window )
{
    Rect          vScrollRect;
    short          numPictures;
    ControlHandle  scrollBarH;

    if ( ( numPictures = CountResources( 'PICT' ) ) <= 0 )
        DoError( "\pNo PICT resources were found!", true );
}

```

If no PICT resources are available, `DoError()` is called. Otherwise, `SetUpScrollBar()` creates a `Rect` the proper size for the scroll bar, then creates the scroll bar with a call to `NewControl()`. The scroll bar is attached to `window`, bounded by `vScrollRect`. Since a scroll bar's title is never used, `kEmptyTitle` does the job adequately.

The scroll bar will be visible and will range in value from `kMinValue` to `numPictures`, the number of available PICT resources. `kStartValue` is the initial value of the scroll bar and determines the initial position of the scroll bar thumb. `scrollBarProc` tells `NewControl()` to create a scroll bar, as opposed to some other type of control. The final parameter is a reference value available for your application's convenience. You can use these 4 bytes as scratch pad space.

```

vScrollRect = window->portRect;
vScrollRect.top -= 1;
vScrollRect.bottom += 1;
vScrollRect.left = vScrollRect.right -
                    kScrollBarWidth + 1;
vScrollRect.right += 1;

```

```

        scrollBarH = NewControl( window, &vScrollRect,
                                kEmptyTitle, kVisible, kStartValue, kMinValue,
                                numPictures, scrollBarProc, kNilRefCon );
    }

```

ScrollProc() gets called whenever a mouseDown occurs in the page-up, page-down, up-arrow, or down-arrow region of the scroll bar. As you'll see, our code doesn't call ScrollProc() directly. Instead, a pointer to the function ScrollProc() is passed as a parameter to TrackControl() after the mouseDown is detected. TrackControl() will continue calling ScrollProc() as long as the mouse button remains down.

```

/***** ScrollProc *****/
pascal void ScrollProc( ControlHandle theControl, short
                        partCode )
{
    short          curCtlValue, maxCtlValue, minCtlValue;
    WindowPtr      window;

```

A handle to the scroll bar is passed in to ScrollProc() as the parameter theControl. partCode indicates the scroll bar part currently being clicked on.

maxCtlValue, curCtlValue, and minCtlValue are set to the maximum, current, and minimum values of theControl. window is set to the scroll bar's owning window, which happens to be Pager's only window.

```

    maxCtlValue = GetCtlMax( theControl );
    curCtlValue = GetCtlValue( theControl );
    minCtlValue = GetCtlMin( theControl );

    window = (**theControl).ctrlOwner;

```

If the mouse click was inPageDown or inDownButton, increase the value of the control by 1. If the mouse click was inPageUp or inUpButton, decrease the value of the control by 1. In either case, if the scroll bar's value was changed, UpdateWindow() is called to draw the current picture.

```

    switch ( partCode )
    {
        case inPageDown:
        case inDownButton:

```



```

        if ( curCtlValue < maxCtlValue )
        {
            curCtlValue += 1;
            SetCtlValue( theControl, curCtlValue );
            UpdateWindow( window );
        }
        break;
    case inPageUp:
    case inUpButton:
        if ( curCtlValue > minCtlValue )
        {
            curCtlValue -= 1;
            SetCtlValue( theControl, curCtlValue );
            UpdateWindow( window );
        }
    }
}

```



User interface issue: Notice that if the scroll bar's thumb is at the top and you click the up arrow, nothing happens. This algorithm avoids the annoying "bouncing" effect of a scroll bar whose thumb is at the end of its travel.

`EventLoop()` works much the same as in earlier *Primer* programs.

```

/***** EventLoop *****/

void    EventLoop( void )
{
    EventRecord    event;

    gDone = false;

    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event,
                           kSleep, nil ) )
            DoEvent( &event );
    }
}

```

DoEvent() acts as a dispatcher for mouseDown and updateEvs. mouseDowns are handled by HandleMouseDown().

```
/****** DoEvent *****/
```

```
void      DoEvent( EventRecord *eventPtr )
{
    WindowPtr      window;

    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
```

In the case of an updateEvt, BeginUpdate() is called. Then, all of the window's controls are redrawn via a call to DrawControls(). In this case, the Pager window has a single control, the scroll bar. Next, UpdateWindow() is called to draw the current picture. Finally, the update ends with the usual call to EndUpdate().

```
        case updateEvt:
            window = (WindowPtr)eventPtr->message;

            BeginUpdate( window );
            DrawControls( window );
            UpdateWindow( window );
            EndUpdate( window );
            break;
    }
}
```

HandleMouseDown() looks the same at the start:

```
/****** HandleMouseDown *****/
```

```
void      HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr      window;
    short           thePart;
    Point           thePoint;
    ControlHandle   theControl;
```

```

thePart = FindWindow( eventPtr->where, &window );
switch ( thePart )
{
    case inSysWindow :
        SystemClick( eventPtr, window );
        break;
    case inDrag :
        DragWindow( window, eventPtr->where,
                    &screenBits.bounds );
        break;

```

The big change comes when a `mouseDown` occurs in the content region (`inContent`) of the Pager window. The `mouseDown`'s location (`eventPtr->where`) is copied into a local variable (`thePoint`), then translated into the window's local coordinate system.

```

case inContent:
    thePoint = eventPtr->where;
    GlobalToLocal( &thePoint );

```

The localized point is passed to `FindControl()`, which returns a handle to the selected control (in the parameter `theControl`) and a part code indicating what part of the control was selected.

```

thePart = FindControl( thePoint, window,
                      &theControl );

```

The `WindowPeek` field `controlList` is a handle to the first control in the window's control list. Since the Pager window contains only a single control, we can use this field to tell if the control found by `FindControl()` was indeed the scroll bar.

```

if ( theControl ==
    ((WindowPeek>window)->controlList )
{

```

If `theControl` is your scroll bar, find out if it was in the thumb. If it was, call `TrackControl()` to drag an outline of the thumb up and down the scroll bar. When the thumb is released, force an update event, so the Pager window will be redrawn using the new control value.

```

if ( thePart == inThumb )
{
    thePart = TrackControl( theControl,
                           thePoint, kNilActionProc );
    InvalRect ( &(window->portRect) );
}

```

If any other part of the control was used, call `TrackControl()` with a pointer to `ScrollProc()`. `TrackControl()` maintains control until the mouse button is released.



When we pass the address of `ScrollProc()` to `TrackControl()`, we're asking `TrackControl()` to call `ScrollProc()` directly. In this case, `ScrollProc()` is known as a **call-back function**. Whenever the Toolbox calls a call-back function, it uses the Pascal standards for passing parameters on the stack. That's why the pascal keyword had to be used in `ScrollProc()`'s declaration.

Use a call-back with `TrackControl()` if you want to act on the control while the mouse button is still down, as in this case, where we continuously scroll while the mouse is down in the down arrow. If you pass `nil` as the call-back address, the control will animate, but its value will not change until the mouse button is released.

```

else
    thePart = TrackControl( theControl,
                           thePoint, &ScrollProc );
}
break;
case inGoAway :
    gDone = true;
    break;
}
}

```

`UpdateMyWindow()` starts by creating a new region. The window's clip region is copied into this region using `GetClip()`.

```

/***** UpdateWindow *****/

void      UpdateWindow( WindowPtr window )
{
    PicHandle      currentPicture;
    Rect           windowRect;
    RgnHandle      tempRgn;

    tempRgn = NewRgn();
    GetClip( tempRgn );

```

Then, the window's content region, minus the area covered by the scroll bar, is erased.

```

    windowRect = window->portRect;
    windowRect.right -= kScrollBarWidth;
    EraseRect( &windowRect );

```

Next, the clip region is set to this adjusted rectangle.

```

    ClipRect( &windowRect );

```

After that, `GetCtlValue()` is used to retrieve the scroll bar's current value, which is used to load the appropriate PICT resource.



For example, if there were 30 PICT resources available, the scroll bar would run from 1 to 30. If the current thumb setting was 10, the call to `GetIndResource()` would return a handle to the tenth PICT resource. Since `GetIndResource()` returns a handle, you can use C's typedefing mechanism to convert it to a `PicHandle`.

Note that only one PICT at a time is ever loaded into memory. When the scroll bar's value changes, a replacement PICT is loaded, not an additional one.

```

    currentPicture = (PicHandle)GetIndResource( 'PICT',
        GetCtlValue( ((WindowPeek)window)->controlList ) );

```

If the PICT cannot be loaded, an appropriate message is passed to `DoError()`. Otherwise, the picture is centered and drawn, and the original clip region is restored.



If we hadn't limited the clip region, a large enough picture would have obscured the scroll bar. If the original clip region was not restored, any attempts to redraw the scroll bar (when its value changed, for example) would be clipped. Try it!

```

    if ( currentPicture == nil )
        DoError( "\\pCan't Load PICT resource!", true );

    CenterPict( currentPicture, &windowRect );
    DrawPicture( currentPicture, &windowRect );

    SetClip( tempRgn );
    DisposeRgn( tempRgn );
}

CenterPict() is the same as it ever was.

/***** CenterPict *****/

void    CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top );
    OffsetRect( &pictRect, (windRect.right -
                pictRect.right)/2, (windRect.bottom -
                pictRect.bottom)/2 );
    *destRectPtr = pictRect;
}

```

**DoError()** takes two parameters. The first is the error string. **DoError()** will use **ParamText()** to place the error string into the alert brought up by **StopAlert()**. If the second parameter is true, **DoError()** will exit to the Finder.



We'll use `DoError()` throughout the rest of the chapter. Though `DoError()` is effective for our purposes, it represents a fairly simple error-handling strategy. As your programs get larger, you'll want to implement your own error-handling scheme.

```

/***** DoError *****/

void    DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString,
               kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}

```

## The Scrap Manager

Whenever you use the Mac's copy, cut, or paste facilities, you're making use of the **Scrap Manager**. The Scrap Manager manages the **desk scrap**, more commonly known as the **Clipboard**. Our next program, **ShowClip**, will use the Scrap Manager Toolbox routines to open the Clipboard and display the contents in a window.

### Scrap Manager Basics

Data copied to the desk scrap is stored in two basic flavors, `TEXT` and `PICT`. Data stored in `TEXT` format consists of a series of ASCII characters. Data stored in `PICT` format consists of a QuickDraw picture. `ShowClip` will handle both `TEXT` and `PICT` data types.

The desk scrap normally resides in memory. If space is tight, however, the Scrap Manager can create a temporary file to write the scrap out to disk.

The Scrap Manager consists of six routines: `InfoScrap()`, `UnloadScrap()`, `LoadScrap()`, `ZeroScrap()`, `PutScrap()`, and `GetScrap()`. With the exception of `InfoScrap()`, each of these functions returns a long containing a result code (I:457).

When your application processes a **Cut** or **Copy** command, you'll want to write the cut or copied data to the desk scrap. You'll call `ZeroScrap()` to initialize the scrap, and `PutScrap()` to load the data into the scrap. Similarly, you'll use `GetScrap()` to load data from the scrap in response to a **Paste** command.

You can use `InfoScrap()` to find out if the scrap is currently resident in memory. If so, you can call `UnloadScrap()` to write the scrap back out to disk. Conversely, you can call `LoadScrap()` to load a disk-based scrap back into memory.

Here's a little more detail on the six Scrap Manager routines.

## InfoScrap()

`InfoScrap()` returns a pointer to a struct of type `ScrapStuff`:

```
typedef struct    ScrapStuff
{
    long          scrapSize;
    Handle        scrapHandle;
    int           scrapCount;
    int           scrapState;
    StringPtr     scrapName;
    ScrapStuff, *PScrapStuff;
}
```

A `ScrapStuff` struct contains information about the current scrap. The `scrapSize` field contains the actual size, in bytes, of the desk scrap. The `scrapHandle` field contains a handle to the desk scrap (if it currently resides in memory). The `scrapCount` field is changed every time `ZeroScrap()` is called (we'll get to `ZeroScrap()` later). The `scrapState` field is positive if the desk scrap is memory resident, zero if the scrap is on disk, and negative if the scrap has not yet been initialized. The `scrapName` field contains a pointer to the name of the scrap disk file (usually called the Clipboard file).

## UnloadScrap() and LoadScrap()

If the scrap is currently in memory, `UnloadScrap()` copies the scrap to disk and releases the scrap's memory. If the scrap is currently disk-based, `UnloadScrap()` does nothing.

If the scrap is currently on disk, `LoadScrap()` allocates memory for the scrap and copies it from disk. If the scrap is currently memory resident, `LoadScrap()` does nothing.



## ZeroScrap()

If the desk scrap does not yet exist, `ZeroScrap()` creates it in memory. If it does exist, `ZeroScrap()` clears it. As we mentioned before, `ZeroScrap()` always changes the `scrapCount` field of the `ScrapStuff` struct.

## PutScrap()

`PutScrap()` puts the data pointed to by `source` into the scrap:

```
long PutScrap( long length, ResType theType, Ptr source )
```

The parameter `length` specifies the length of the data, and `theType` specifies its type (PICT or TEXT, for example). You must call `ZeroScrap()` immediately before each call to `PutScrap()`.

## GetScrap()

`GetScrap()` resizes the handle `hDest` and stores a copy of the scrap in this resized block of memory:

```
long GetScrap( Handle hDest, ResType theType, long *offset )
```

Specify the type of data you want to retrieve in the parameter `theType`. The `offset` parameter is set to the returned data's offset in bytes from the beginning of the desk scrap. `GetScrap()` returns a `long` containing the length of the data in bytes.

You can actually put and get data types other than TEXT and PICT to and from the scrap (I:461). For the most part, however, the TEXT and PICT data types should serve your needs.

## ShowClip

`ShowClip` demonstrates scrap handling basics. If you use **Cut** or **Copy** to move some text or a picture to the scrap, `ShowClip` will display the cut or copied data in a window. `ShowClip` works like this:

- Initializes the Toolbox.
- Initializes a window.
- Retrieves whatever data is in the scrap, drawing the data in the window.
- Waits for a mouse click to exit.

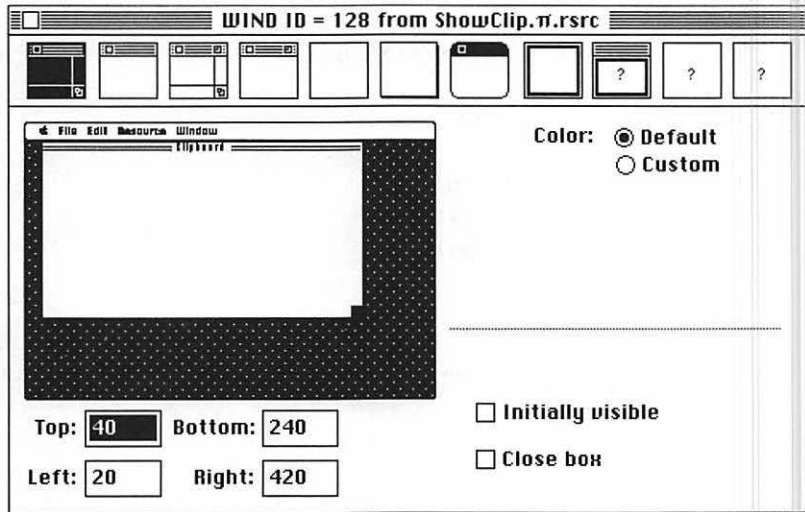
`ShowClip` also does error checking. It warns if the WIND resource is missing, or if the scrap is empty.

## ShowClip Resources

Create a folder called ShowClip inside your Development folder. Use ResEdit to create a new file called ShowClip.π.rsrc inside the ShowClip folder. Since ShowClip uses the same error-handling mechanism as Pager, open the file Pager.π.rsrc and copy both the ALRT and the DITL resources, pasting them in ShowClip.π.rsrc.

Next, create a WIND resource according to the specifications in Figure 7.16. Select **Set 'WIND' Characteristics...** from the **WIND** menu, set the **Window title:** field to **Clipboard**, and click the **OK** button. Change the WIND's resource ID to 128 and check the **purgeable** check box. Close the windows associated with the WIND resource.

Quit ResEdit, saving your changes.



**Figure 7.16** Specifications for the WIND resource.

## The ShowClip Project

Now you're ready to launch THINK C. Create a new project in the ShowClip folder. Call it ShowClip.π. Use **Add...** from the **Source** menu to add MacTraps to the project.

Once MacTraps is added, open a new source code window and enter the program:

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L
```

```

#define kEmptyString      "\p"
#define kNilFilterProc    nil

#define kErrorAlertID     kBaseResID

/*****
/*  Functions  */
*****/

void      ToolBoxInit( void );
void      WindowInit( void );
void      MainLoop( void );
void      CenterPict( PicHandle picture, Rect *destRectPtr );
void      DoError( Str255 errorString, Boolean fatal );

/***** main *****/

void      main( void )
{
    ToolBoxInit();
    WindowInit();
    MainLoop();
}

/***** ToolBoxInit *****/

void      ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

```
/***** WindowInit *****/  
void      WindowInit( void )  
{  
    WindowPtr      window;  
  
    window = GetNewWindow( kBaseResID, nil, kMoveToFront );  
  
    if ( window == nil )  
        DoError( "\pCan't load the WIND resource!", true );  
  
    ShowWindow( window );  
    SetPort( window );  
}  
  
/**** MainLoop ****/  
void      MainLoop( void )  
{  
    Rect          pictureRect;  
    Handle         clipHandle;  
    long           length, offset;  
    WindowPtr      window;  
  
    clipHandle = NewHandle( 0 );  
    window = FrontWindow();  
  
    if ( ( length = GetScrap( clipHandle, 'TEXT', &offset  
                               ) ) < 0 )  
    {  
        if ( GetScrap( clipHandle, 'PICT', &offset ) < 0 )  
            DoError(  
                "\pThere's no PICT and no text in the scrap..."  
                , true );  
        else  
        {  
            pictureRect = window->portRect;  
            CenterPict( (PicHandle)clipHandle,  
                        &pictureRect );  
            DrawPicture( (PicHandle)clipHandle,  
                         &pictureRect );  
        }  
    }  
}
```

```

        else
        {
            HLock( clipHandle );
            TextBox( *clipHandle, length, &(window->portRect),
                    teJustLeft );
            HUnlock( clipHandle );
        }

        while ( !Button() ) ;
    }

/***** CenterPict *****/

void    CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top);
    OffsetRect( &pictRect, (windRect.right -
                pictRect.right)/2,
                (windRect.bottom -
                pictRect.bottom)/2);
    *destRectPtr = pictRect;
}

/***** DoError *****/

void    DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString,
                kEmptyString );

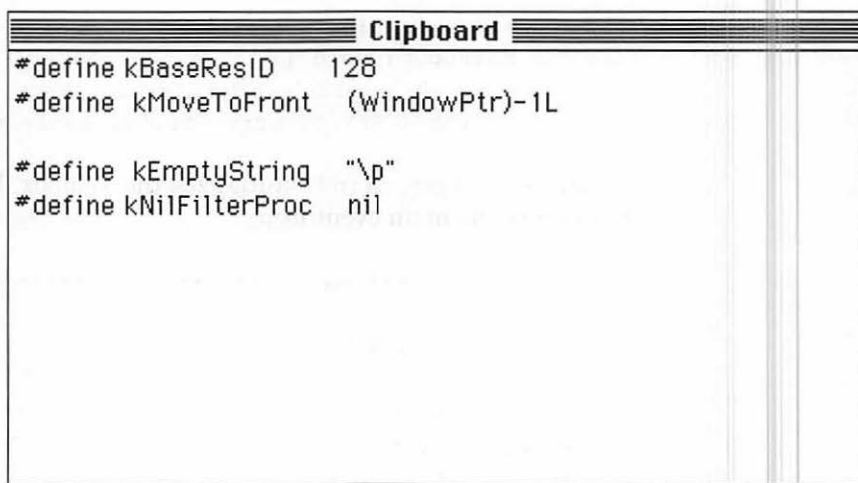
    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}

```

## Running ShowClip

Once you've finished typing in the code, save it as `ShowClip.c` and add it to the project using **Add** from the **Source** menu. Before you run the program, however, do a **Cut** or **Copy** operation on the `ShowClip.c` file, or copy a picture from the Scrapbook; otherwise, you'll get an alert telling you that the scrap is empty. Now, select **Run** from the **Project** menu, clicking **Yes** to the question **Bring the project up to date?** `ShowClip` should bring up a window displaying the text or picture that you cut or copied (Figure 7.17).



**Figure 7.17** `ShowClip` in action.

Quit by clicking the mouse. Take `ShowClip` out for a test drive. Try copying a color `PICT` and then running `ShowClip`. Try out some varying sizes and styles of text. Hmmm, color pictures seem to work all right, but all of the text appears in a single font, size, and style.

Take a look at the code and you'll see why.

---

## Walking Through the ShowClip Code

---

`ShowClip` starts with a subset of the `#defines` used in `Pager.c`.

```
#define kBaseResID    128
#define kMoveToFront  (WindowPtr)-1L
```

```
#define kEmptyString      "\p"
#define kNilFilterProc    nil

#define kErrorAlertID     kBaseResID
```

As usual, each function is prototyped.

```
/* *****
 * Functions
 * *****
 */

void    ToolBoxInit( void );
void    WindowInit( void );
void    MainLoop( void );
void    CenterPict( PicHandle picture, Rect *destRectPtr );
void    DoError( Str255 errorString, Boolean fatal );
```

Just as in *Pager*, `main()` initializes the Toolbox, loads the window, then enters the main event loop.

```
/* ***** main ***** */

void    main( void )
{
    ToolBoxInit();
    WindowInit();
    MainLoop();
}
```

There aren't any changes in `ToolBoxInit()`:

```
/* ***** ToolBoxInit ***** */

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}
```

WindowInit() loads the WIND resource. If it can't be loaded, DoError() is called.

```

/***** WindowInit *****/

void      WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
        DoError( "\pCan't load the WIND resource!", true );

    ShowWindow( window );
    SetPort( window );
}

```

MainLoop() is where the action is. You use NewHandle() (II:32) to create minimum-sized blocks of storage for your PICT or TEXT data. GetScrap() will resize these memory blocks for you, as needed.

```

/***** MainLoop *****/

void      MainLoop( void )
{
    Rect        pictureRect;
    Handle       clipHandle;
    long         length, offset;
    WindowPtr    window;

    clipHandle = NewHandle( 0 );

```

Since ShowClip has only a single window, we know that FrontWindow() will return a pointer to it.

```

    window = FrontWindow();

```

The call to GetScrap() looks in the desk scrap for some TEXT data. If it can't find any, GetScrap() will return a negative result.

```

    if ( ( length = GetScrap( clipHandle, 'TEXT', &offset
                             ) ) < 0 )
    {

```



In that case, we'll call `GetScrap()` again, looking for some PICT data. If that fails, call `DoError()` with an appropriate message.

```
if ( GetScrap( clipHandle, 'PICT', &offset ) < 0 )
    DoError(
        "\pThere's no PICT and no text in the scrap..."
        , true );
```

If we found some PICT data, center it and draw it in the ShowClip window.

```
else
{
    pictureRect = window->portRect;
    CenterPict( (PicHandle)clipHandle,
                &pictureRect );
    DrawPicture ( (PicHandle)clipHandle,
                  &pictureRect );
}
```

If we found the TEXT data in the scrap, lock `clipHandle` with `Hlock()`, then call `TextBox()` to draw the text in the ShowClip window. After that, unlock the block with a call to `clipHandle`.

```
else
{
    HLock( clipHandle );
    TextBox( *clipHandle, length, &(window->portRect),
             teJustLeft );
    HUnlock( clipHandle );
}
```



Here's the answer to an earlier question. All text appears in a single font, size, and style, in part because all of ShowClip's text drawing is done via `TextBox()`. Even more important is the fact that TEXT data in the scrap has no special formatting information embedded with it. If you want to copy and paste formatted text, you'll have to create your own scrap type or, better yet, make use of some third party's predefined type.

Finally, wait for a mouse click to exit.

```
while ( !Button() ) ;
}
```

CenterPict() is the same routine you've used in the other *Primer* PICT drawing programs:

```
/****** CenterPict *****/

void      CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect      windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top);
    OffsetRect( &pictRect, (windRect.right -
                pictRect.right)/2,
                (windRect.bottom -
                pictRect.bottom)/2);
    *destRectPtr = pictRect;
}
```

DoError() is the same as the one found in Pager.

```
/****** DoError *****/

void      DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString,
                kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}
```



For a more sophisticated example of proper desk scrap handling, check out the FormEdit program in Volume II of the *Macintosh C Programming Primer*.

---

## The Sound Manager

---

As anyone who's ever fired up a MacRecorder will agree, recording and playing back your own sounds on the Mac is fun. Nowadays, most Macs come with a built-in sound recording capability. System 7 includes some Toolbox routines that, combined with the appropriate hardware, will allow you to play and record from within your own applications. Our next program will show you how.

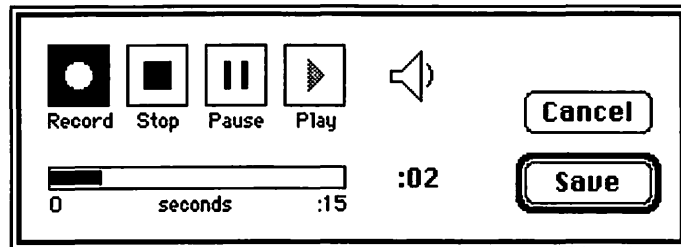
---

## SoundMaker

---

System 7's **Sound Manager** includes a new function called `SndRecord()`. `SndRecord()` puts up the dialog box shown in Figure 7.18, automating the process of recording sounds. To record a sound via `SndRecord()`, you'll need a Mac with sound recording hardware (either built in or by way of a third-party solution like MacRecorder).

Once your Mac is suitably equipped, you're ready to program!



**Figure 7.18** Using `SndRecord()` to record a sound.

## SoundMaker Resources

**SoundMaker** makes use of two resources, the ALRT and DITL used by `DoError()`. Create a folder called **SoundMaker** inside your Development folder. Use ResEdit to create a new file called `SoundMaker.π.rsrc` inside the **SoundMaker** folder. Open the file `Pager.π.rsrc` and copy both the ALRT and the DITL resources, pasting them in `SoundMaker.π.rsrc`.

Quit ResEdit, saving your changes.

## The SoundMaker Project

Launch THINK C and create a new project in the SoundMaker folder. Call it SoundMaker.π. Use **Add...** from the **Source** menu to add MacTraps to the project.

Once MacTraps is added, open a new source code window and enter the program:

```
#include <Sound.h>
#include <SoundInput.h>
#include <GestaltEqu.h>

#define kBaseResID      128

#define kNilSoundChannel  nil
#define kSynchronous     false

#define kEmptyString     "\p"
#define kNilFilterProc    nil

#define kErrorAlertID     kBaseResID

/*****/
/*  Functions  */
/*****/

void      ToolBoxInit( void );
Handle    RecordSound( void );
void      PlaySound( Handle soundHandle );
void      DoError( Str255 errorString, Boolean fatal );

/***** main *****/

void      main( void )
{
    Handle    soundHandle;
    long      feature;
    OSErr     err;

    MaxApplZone();
    ToolBoxInit();
```

```

err = Gestalt( gestaltSoundAttr, &feature );

if ( err != noErr )
    DoError( "\pError returned by Gestalt!", true );

if ( feature & (1 << gestaltHasSoundInputDevice) )
{
    soundHandle = RecordSound();
    PlaySound( soundHandle );
    DisposHandle( soundHandle );
}
else
    DoError( "\pSound input device not available!!!",
            true );
}

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** RecordSound *****/

Handle  RecordSound( void )
{
    OSErr    err;
    Point    upperLeft;
    Handle    soundHandle;

    SetPt( &upperLeft, 50, 50 );

    soundHandle = nil;

    err = SndRecord( nil, upperLeft, siBestQuality,
                    &soundHandle );

```

```

        if ( err == userCanceledErr )
            DoError( "\pRecording canceled...", true );

        if ( err != 0 )
            DoError( "\pError returned by SndRecord()...",
                    true );

        return( soundHandle );
    }

/***** PlaySound *****/

void    PlaySound( Handle soundHandle )
{
    OSErr    err;

    err = SndPlay( kNilSoundChannel, soundHandle,
                  kSynchronous );

    if ( err != noErr )
        DoError( "\pError returned by SndPlay()...", true );
}

/***** DoError *****/

void    DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString,
              kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}

```

## Running SoundMaker

Once you've finished typing in the code, save it as `SoundMaker.c` and add it to the project using **Add** from the **Source** menu. Next, select **Run** from the **Project** menu, clicking **Yes** to the question **Bring the project up to date?** If the source code compiles correctly, the sound recording dialog box, shown in Figure 7.18, will appear.



If an error message appears, your sound input device may not be set up correctly. To verify your sound input device, go to the **Sound** control panel and click on the **Add...** button. Once you can record a new sound in this control panel, your sound input hardware is set up properly.

When SoundMaker's record dialog box appears, click on the **Record** button to record a sound. You can pause and resume recording with the **Pause** button and stop recording with the **Stop** button. When you've recorded a sound, you can preview it using the **Play** button.

When you're done playing, you have two choices. If you press the **Save** button, the record dialog will exit, passing your sound back to SoundMaker. At this point, SoundMaker will play your recording back to you and then exit.

If you press the **Cancel** button, SoundMaker will display a message telling you that the recording was canceled. Try some of these options. When you're done, let's walk through the code.

## Walking Through the SoundMaker Code

SoundMaker starts with three `#includes`. `Sound.h` contains the definitions relating to the sound-playing function. `SoundInput.h` contains the definitions relating to the sound-recording function. Finally, `GestaltEqu.h` contains the definitions needed to call `Gestalt()`.

```
#include <Sound.h>
#include <SoundInput.h>
#include <GestaltEqu.h>
```

You've seen most of these `#defines` before. `kNilSoundChannel` and `kSynchronous` are both used in the function `PlaySound()`'s call of `SndPlay()`.

```
#define kBaseResID          128

#define kNilSoundChannel    nil
#define kSynchronous        false
```

```
#define kEmptyString          "\p"
#define kNilFilterProc        nil

#define kErrorAlertID         kBaseResID
```

Here are the four function prototypes:

```
/* *****
 * Functions
 * *****
 */

void    ToolBoxInit( void );
Handle  RecordSound( void );
void    PlaySound( Handle soundHandle );
void    DoError( Str255 errorString, Boolean fatal );
```

`main()` starts by calling `MaxApplZone()`, then initializing the Toolbox. `MaxApplZone()` (II:30) maximizes the amount of memory available to your application. Since we will be allocating a large block of memory when we record a sound, we'll want to start with the largest memory block we can.

```
/* ***** main ***** */

void    main( void )
{
    Handle  soundHandle;
    long    feature;
    OSErr   err;

    MaxApplZone();
    ToolBoxInit();
```

Next, we'll call `Gestalt()` to see if the Mac is equipped with a sound input device.

```
err = Gestalt( gestaltSoundAttr, &feature );

if ( err != noErr )
    DoError( "\pError returned by Gestalt!", true );

if ( feature & (1 << gestaltHasSoundInputDevice) )
{
```



If the sound input device is available, we'll call `RecordSound()` to bring up the sound recording dialog box. Then, we'll pass the recorded sound on to `PlaySound()`, demonstrating the proper way to play a recorded sound. If you wanted to, you could use the techniques in `ResWriter` (plus a call to `AddResource()`, (I:124)) to save the sound as a resource. Next, call `DisposHandle()` to free up the allocated memory.

```

        soundHandle = RecordSound();
        PlaySound( soundHandle );
        DisposHandle( soundHandle );
    }

```

If the sound input device was not available, call `DoError()`.

```

        else
            DoError( "\pSound input device not available!!!",
                    true );
    }

```

```

/***** ToolboxInit *****/

```

```

void    ToolboxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

`RecordSound()` calls the Sound Manager function `SndRecord()`. The first parameter is an optional function pointer, which `SndRecord()` will use to filter user actions in the dialog box. This filter function is identical in nature to the filter function associated with `ModalDialog()`. For now, we'll use the default function by passing `nil` as the first parameter.

The second parameter is a `Point`, used to determine the position of the upper left corner of the dialog box. The third parameter determines the quality of the sound. Better quality sound takes up more memory.

The final parameter is a handle to a block of storage space for the recorded sound. By passing a pointer to a nil handle, we're asking SndRecord() to allocate memory for us.

```
/****** RecordSound *****/
```

```
Handle RecordSound( void )
{
    OSErr    err;
    Point    upperLeft;
    Handle    soundHandle;

    SetPt( &upperLeft, 50, 50 );

    soundHandle = nil;

    err = SndRecord( nil, upperLeft, siBestQuality,
                    &soundHandle );
```

If the recording was canceled, call DoError().

```
if ( err == userCanceledErr )
    DoError( "\pRecording canceled...", true );
```

If an error occurred while recording, call DoError().

```
if ( err != 0 )
    DoError( "\pError returned by SndRecord()...",
            true );
```

If the recording process went well, return the handle to the recorded sound.

```
return( soundHandle );
}
```

PlaySound() calls the Sound Manager function SndPlay(). The first parameter allows you to specify a sound channel to play the sound on. The Sound Manager can handle more than one simultaneous sound channel. In our case, we've asked the Sound Manager to allocate a channel for us.

The second parameter is a handle to the sound we'd like played. The third parameter tells the Sound Manager to play this sound synchronously; that is, to not return until the sound finishes playing.

```

/***** PlaySound *****/

void    PlaySound( Handle soundHandle )
{
    OSErr    err;

    err = SndPlay( kNilSoundChannel, soundHandle,
                  kSynchronous );

    If SndPlay() returns an error, call DoError().

    if ( err != noErr )
        DoError( "\pError returned by SndPlay()...", true );
}

/***** DoError *****/

void    DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString,
              kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}

```

SoundMaker demonstrates how easy it is to record and play back a sound using System 7's version of the Sound Manager. What SoundMaker doesn't demonstrate is vast. The Sound Manager is big; it's comprehensive. The Sound Manager supports synthesizers, sound compression, and QuickTime.

If you are interested in learning more about the Sound Manager, get *Inside Macintosh*, Volume VI, and start reading. There's a lot to learn, but the Sound Manager is so cool, it's worth the investment.

---

## Working with Macintosh Files

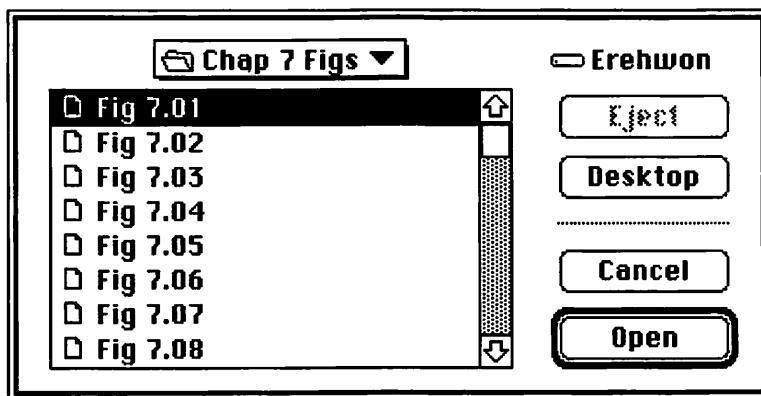
---

The next program, **OpenPICT**, demonstrates several important techniques for working with Macintosh files. OpenPICT makes use of the **Standard File Package**, a set of Toolbox functions that make it

easy to retrieve and save Macintosh files. Once you've selected a file to open or a file name and directory you'd like to save under, you'll want to make use of the File Manager to read or write your file's contents.

## The Standard File Package

The Standard File Package is used by most Macintosh applications to support the **Open**, **Save**, and **Save As...** File menu items. The function `StandardGetFile()` brings up a dialog box similar to the one shown in Figure 7.19. Use this call to allow the user to select a file to be opened.



**Figure 7.19** Standard Get File dialog box.

Here's the function prototype for `StandardGetFile()`:

```
void StandardGetFile( FileFilterProcPtr fileFilter,  
                     short numTypes, STypeList typeList,  
                     StandardFileReply *replyPtr );
```

The first three parameters together determine what files will appear in the dialog box's scrolling list. The third parameter, `typeList`, is an array of type `OSType` (a 4-byte file type signature), whose length is determined by the second parameter, `numTypes`. To limit the list to files of type 'PICT', set `numTypes` to 1 and pass, as `typeList`, an `OSType` array of length 1. To list files of all types, set `numTypes` to -1.

The first parameter, `fileFilter`, is a pointer to a function you provide, which can further filter the list of available file types. Pass

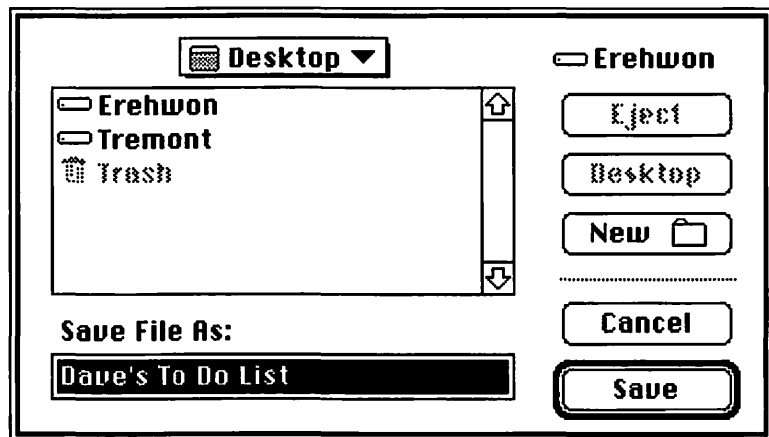
nil as the first parameter if you don't have a need for a more sophisticated filter procedure.

The fourth parameter, `replyPtr`, points to a record of type `StandardFileReply`:

```
struct StandardFileReply
{
    Boolean      sfGood;
    Boolean      sfReplacing;
    OSType       sfType;
    FSSpec       sfFile;
    ScriptCode   sfScript;
    short        sfFlags;
    Boolean      sfIsFolder;
    Boolean      sfIsVolume;
    long         sfReserved1;
    short        sfReserved2;
};
```

You can read about this struct in detail in (VI:26-4). Basically, this is where the selected file is described. The `sfGood` field tells you whether the user clicked the **Cancel** or **Open** button to dismiss the dialog. If `sfGood` is true, pass the fourth field, `sfFile`, on to the appropriate File Manager function to open the file. We'll get to the File Manager in a minute.

The function `StandardPutFile()` brings up a dialog box similar to the one shown in Figure 7.20. Use this call to allow the user to name and save a file.



**Figure 7.20** Standard Put File dialog box.

Here's the function prototype for `StandardPutFile()`:

```
void StandardPutFile( Str255 prompt, Str255 defaultName,  
                    StandardFileReply *replyPtr );
```

The `prompt` parameter specifies the text string to appear above the text field. The dialog box in Figure 7.20 uses a prompt of **Save File As:**. The `defaultName` parameter specifies the text to appear in the editable text field. Typically, your application will use the current name of the file or, in the case of an unnamed file, a string like "Untitled" as a `defaultName`. `StandardPutFile()` will return the user's input to you in a struct of type `StandardFileReply`.

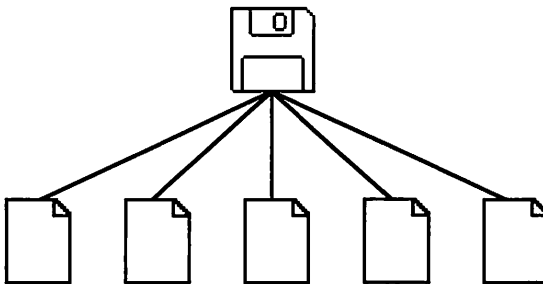
As before, the `sfGood` field tells you whether the user clicked **Save** or **Cancel** to exit the dialog box. If `sfGood` is true, you'll pass the `replyPtr` on to the appropriate File Manager routine when you open the file for writing.

---

## The File Manager

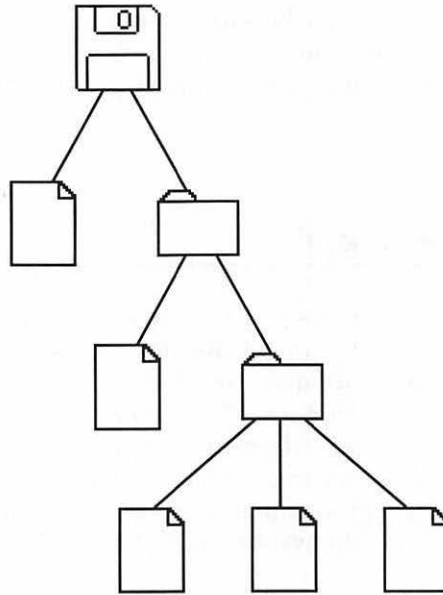
---

There are a few key terms you should know before you use the File Manager. **Volumes** are the media used to store files. When users press the **Desktop** button in the `StandardGetFile()` dialog box, they'll be presented with a list of all mounted volumes, in addition to all items on the desktop. Macintosh floppy and hard disks are both examples of volumes. In the original Macintosh (the one with 64K ROMs), all the files on a volume were organized in a flat file format called the Macintosh File System (MFS) (Figure 7.21).



**Figure 7.21** Flat files.

The concept of **folders** existed on these “flat” Macs, but internally the files on a volume were all stored in one big list. The folders were an illusion maintained by the Finder. On flat volumes, users can’t have two files with the same name, even if they’re in different folders. The Mac Plus (with 128K ROMs) introduced a new method for organizing files: the **Hierarchical File System** (HFS) (Figure 7.22). HFS is truly hierarchical, allowing folders within folders within folders. By the way, as you read about the File Manager, you’ll frequently see the term **directory** used interchangeably with the term folder.



**Figure 7.22** Hierarchical files.



The File Manager was completely remade when the Mac Plus came out. The original Macintosh Filing System (MFS) was inadequate to handle the number of files that hard disks could hold. The Hierarchical Filing System (HFS) replaced it, and the new HFS Toolbox calls were written into *Inside Macintosh*, Volume IV.

With the release of System 7, the File Manager was modified once again. These changes are described in Volume VI of *Inside Macintosh*. To learn everything there is to know about the File Manager, glance through the File Manager chapter in Volume I, then carefully read the File Manager chapters in Volumes IV and VI.

## Using the File Manager

OpenPICT makes use of some high-level File Manager calls to open a PICT file, read in some data, and close the file again. Though these calls represent a small portion of the File Manager's capabilities, the techniques demonstrated by OpenPICT should get you started.

Once the user has chosen a PICT file to open (by way of `StandardGetFile()`), OpenPICT uses the File Manager routine `FSpOpenDF()` to open the file for reading:

```
OSErr FSpOpenDF( FSSpec spec, SignedByte permission,  
                 short *refNumPtr );
```

`FSpOpenDF()` is designed to open a file's data fork (as opposed to its resource fork). The first parameter is an `FSSpec`, otherwise known as a **file system specification**. The `FSSpec` uses three things to identify a specific file:

- The reference number of the file's volume.
- The directory ID of the file's parent directory.
- The file's name.

If you use `StandardGetFile()` to prompt for a file to open, you won't need to worry about the elements of an `FSSpec`. `StandardGetFile()` uses the selected file to create an `FSSpec` in the `sfFile` field of the `StandardFileReply`. You'll see how this is done when we walk through the OpenPICT source code.

The second parameter to `FSpOpenDF()` specifies the **mode** you'll use to access the file. Our example opens the file for exclusive read permission. The other permissions are listed on (VI:25-32).

The third parameter is a pointer to a **file reference number**, a single 2-byte value that can be used to access an open file. OpenPICT passes this reference number on to File Manager routines, such as `GetEOF()`, `FSRead()`, and `FSClose()`.

OpenPICT uses `GetEOF()` to find out how large the file is, `FSRead()` to read in the PICT, and `FSClose()` to close the file. Again, you'll see how this is done as we walk through the OpenPICT source code. The key point being made here is to use `StandardGetFile()` whenever you need to open a file.



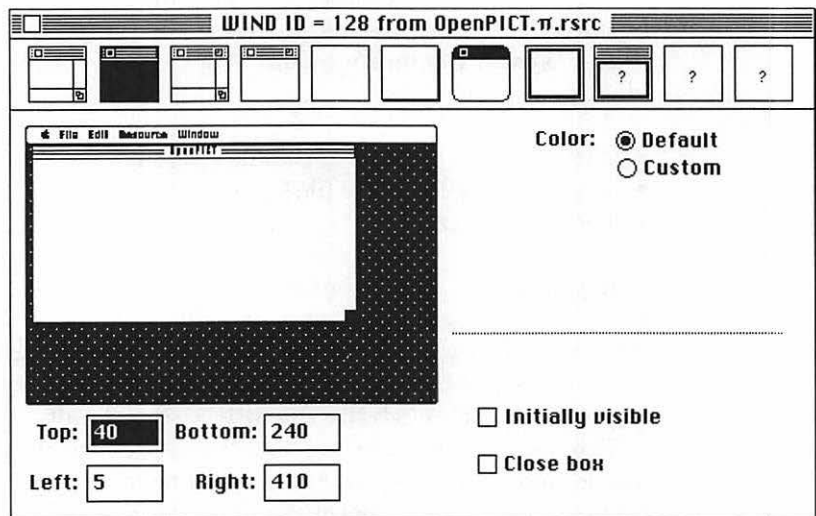
By the way, don't confuse `StandardGetFile()` with the function it replaces, `FSGetFile()`. `FSGetFile()` is the old way of doing things. `StandardGetFile()` is part of the new, improved File Manager that rode in on the heels of System 7.



## OpenPICT Resources

OpenPICT makes use of three resources, the ALRT and DITL used by DoError(), and a WIND used to create a picture window. Create a folder called OpenPICT inside your Development folder. Use ResEdit to create a new file called OpenPICT.π.rsrc inside the OpenPICT folder. Open the file SoundMaker.π.rsrc and copy both the ALRT and the DITL resources, pasting them in OpenPICT.π.rsrc.

Next, create a WIND resource according to the specifications in Figure 7.23. Make sure the WIND's resource ID is 128, change the window's title to OpenPICT, and make sure the **Purgeable** check box is checked. Quit ResEdit, saving your changes.



**Figure 7.23** Specifications for OpenPICT's WIND resource.

## The OpenPICT Project

Launch THINK C and create a new project in the OpenPICT folder. Call it OpenPICT.π. Use **Add...** from the **Source** menu to add MacTraps to the project.

Once MacTraps is added, open up a new source code window and enter the program:

```
#include <GestaltEqu.h>

#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L
```

```

#define kEmptyString      "\p"
#define kNilFilterProc    nil

#define kErrorAlertID     kBaseResID

#define kPICTHeaderSize   512

/*****/
/*  Functions  */
/*****/

void      ToolBoxInit( void );
void      GetFileName( StandardFileReply *replyPtr );
PicHandle LoadPICTFile( StandardFileReply *replyPtr );
void      WindowInit( void );
void      DrawMyPicture( PicHandle picture );
void      CenterPict( PicHandle picture, Rect
                    *destRectPtr );
void      DoError( Str255 errorString, Boolean fatal );

/***** main *****/

void      main( void )
{
    PicHandle      picture;
    StandardFileReply  reply;

    ToolBoxInit();

    GetFileName( &reply );

    if ( reply.sfGood )
    {
        picture = LoadPICTFile( &reply );

        if ( picture != nil )
        {
            WindowInit();
            DrawMyPicture( picture );

            while ( ! Button() );
        }
    }
}

```

```

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** GetFileName *****/

void    GetFileName( StandardFileReply *replyPtr )
{
    SFTYPEList    typeList;
    short         numTypes;
    long          feature;
    OSErr         err;

    err = Gestalt( gestaltStandardFileAttr, &feature );

    if ( err != noErr )
        DoError( "\pError returned by Gestalt!", true );

    if ( feature & ( 1 << gestaltStandardFile58 ) )
    {
        typeList[ 0 ] = 'PICT';
        numTypes = 1;

        StandardGetFile( kNilFilterProc, numTypes,
                        typeList, replyPtr );
    }
    else
    {
        DoError( "\pThe new Standard File routines \
are not supported by this OS!", true );
    }
}

```

```

/***** LoadPICTFile *****/

PicHandle LoadPICTFile( StandardFileReply *replyPtr )
{
    short      srcFile;
    PicHandle  picture;
    char       pictHeader[ kPICTHeaderSize ];
    long       pictSize, headerSize;
    long       feature;
    OSErr      err;

    err = Gestalt( gestaltFSAttr, &feature );

    if ( err != noErr )
        DoError( "\pError returned by Gestalt!", true );

    if ( feature & (1 << gestaltHasFSSpecCalls) )
    {
        if ( FSpOpenDF( &(replyPtr->sfFile), fsRdPerm,
                        &srcFile )
            != noErr )
        {
            DoError( "\pCan't open file...", false );
            return( nil );
        }

        if ( GetEOF( srcFile, &pictSize ) != noErr )
        {
            DoError( "\pError returned by GetEOF()...",
                    false );
            return( nil );
        }

        headerSize = kPICTHeaderSize;

        if ( FSRead( srcFile, &headerSize, pictHeader )
            != noErr )
        {
            DoError( "\pError reading file header...",
                    false );
            return( nil );
        }

        pictSize -= kPICTHeaderSize;
    }
}

```

```

        if ( ( picture = (PicHandle)NewHandle( pictSize )
              ) == nil )
        {
            DoError(
                "\pNot enough memory to read picture..."
                , false );
            return( nil );
        }

        HLock( (Handle)picture );

        if ( FSRead( srcFile, &pictSize, *picture ) !=
              noErr )
        {
            DoError( "\pError returned by FSRead()...",
                    false );
            return( nil );
        }

        HUnlock( (Handle)picture );
        FSClose( srcFile );

        return( picture );
    }
    else
    {
        DoError( "\pThe new FSSpec File Manager routines \
are not supported by this OS!", true );
    }
}

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
        DoError( "\pCan't load WIND resource...", true );

    ShowWindow( window );
    SetPort( window );
}

```

```

/***** DrawMyPicture *****/

void    DrawMyPicture( PicHandle picture )
{
    Rect        pictureRect;
    WindowPtr    window;

    window = FrontWindow();

    pictureRect = window->portRect;

    CenterPict( picture, &pictureRect );
    DrawPicture( picture, &pictureRect );
}

/***** CenterPict *****/

void    CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect        windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top );
    OffsetRect( &pictRect, (windRect.right -
                pictRect.right)/2, (windRect.bottom
                - pictRect.bottom)/2 );
    *destRectPtr = pictRect;
}

/***** DoError *****/

void    DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString,
                kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

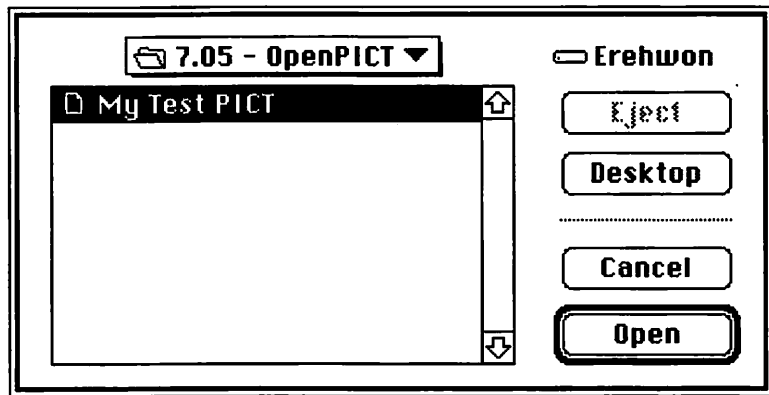
    if ( fatal )
        ExitToShell();
}

```

## Running OpenPICT

Once you've finished typing in the code, save it as `OpenPICT.c` and add it to the project using **Add** from the **Source** menu. Before you run your new program, use your favorite graphics program (Studio 8, Canvas, whatever) to create a PICT file. It's important that the file have a file type of PICT, otherwise our call to `StandardGetFile()` won't find it.

Once your PICT file is created, select **Run** from THINK C's **Project** menu, clicking **Yes** to the question **Bring the project up to date?**. If the source code compiles correctly, the Standard Get File dialog box, shown in Figure 7.24, will appear.



**Figure 7.24** The Standard Get File dialog box, prompting for a PICT file.

If you hit the **Cancel** button, the dialog box will disappear and OpenPICT will exit. If you hit the **Open** button, OpenPICT will open the selected file and a window displaying your PICT will appear.

---

## Walking Through the OpenPICT Code

---

OpenPICT starts with the `#include` needed to call `Gestalt()`. We'll need `Gestalt()` to check if the System 7 Standard File routines are available.

```
#include <GestaltEqu.h>
```

You've seen most of these #defines before. kPictHeaderSize is set to the standard length of a PICT file's header. We'll use this information when we read in the header in the function LoadPictFile().

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

#define kEmptyString    "\p"
#define kNilFilterProc   nil

#define kErrorAlertID    kBaseResID

#define kPictHeaderSize  512
```

As always, the function prototypes.

```
/* *****
 * Functions
 * *****
 */

void      ToolBoxInit( void );
void      GetFileName( StandardFileReply *replyPtr );
PicHandle LoadPictFile( StandardFileReply *replyPtr );
void      WindowInit( void );
void      DrawMyPicture( PicHandle picture );
void      CenterPict( PicHandle picture, Rect
                    *destRectPtr );
void      DoError( Str255 errorString, Boolean fatal );
```

main() initializes the Toolbox, then calls GetFileName() to prompt the user for the name of a PICT file to open.

```
/* ***** main ***** */

void      main( void )
{
    PicHandle      picture;
    StandardFileReply  reply;

    ToolBoxInit();

    GetFileName( &reply );
```



If the user clicked the **Open** button, `reply.sfGood` will be true.

```
if ( reply.sfGood )
{
```

In that case, call `LoadPICTFile()` to retrieve the PICT from the file. Note that the `reply` was loaded by `GetFileName()`, then passed on to `LoadPICTFile()`.

```
    picture = LoadPICTFile( &reply );
```

If the picture was successfully retrieved, create a window and draw the picture in it.

```
        if ( picture != nil )
        {
            WindowInit();
            DrawMyPicture( picture );

            while ( ! Button() ) ;
        }
    }
}
```

`ToolBoxInit()` still hasn't changed. Sigh.

```
/****** ToolBoxInit *****/
```

```
void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}
```

`GetFileName()` starts with a call to `Gestalt()`, checking for the Standard File routines that came with System 7. If an error is returned by `Gestalt()`, or the routines are not present, `DoError()` is called.

```

/***** GetFileName *****/

void      GetFileName( StandardFileReply *replyPtr )
{
    SFTypelist    typelist;
    short         numTypes;
    long          feature;
    OSErr         err;

    err = Gestalt( gestaltStandardFileAttr, &feature );

    if ( err != noErr )
        DoError( "\pError returned by Gestalt!", true );

    if ( feature & (1 << gestaltStandardFile58) )
    {

```

If the System 7 Standard File routines are present, create a `typelist` array with a single entry, 'PICT'. This will limit the Standard Get File dialog to display only PICT files.

```

        typelist[ 0 ] = 'PICT';
        numTypes = 1;

```

Next, `StandardGetFile()` is called, placing its result in the struct pointed to by `replyPtr`.

```

        StandardGetFile( kNilFilterProc, numTypes,
                        typelist, replyPtr );
    }
    else
    {
        DoError( "\pThe new Standard File routines \
are not supported by this OS!", true );
    }
}

```

`LoadPICTFile()` also calls `Gestalt()`, this time looking for the presence of FSSpec savvy calls. The File System Specification known as FSSpec was also introduced with System 7.

```

/***** LoadPICTFile *****/

```

```

PicHandle    LoadPICTFile( StandardFileReply *replyPtr )
{
    short      srcFile;
    PicHandle   picture;
    char       pictHeader[ kPICTHeaderSize ];
    long       pictSize, headerSize;
    long       feature;
    OSErr      err;

    err = Gestalt( gestaltFSAttr, &feature );

    if ( err != noErr )
        DoError( "\pError returned by Gestalt!", true );

    if ( feature & (1 << gestaltHasFSSpecCalls) )
    {

```

If the FSSpec calls are present, call FSpOpenDF() to open the specified PICT file with read permission. Call DoError() if the file could not be opened.

```

        if ( FSpOpenDF( &(amp;replyPtr->sfFile), fsRdPerm,
                        &srcFile ) != noErr )
        {
            DoError( "\pCan't open file...", false );
            return( nil );
        }

```

Next, call GetEOF() to get the size of the file, in bytes. Again, if there is a problem with this call, call DoError(). Notice that the reference number returned by FSpOpenDF() is all we need to supply to GetEOF() to specify the file we want to perform this operation on.

```

        if ( GetEOF( srcFile, &pictSize ) != noErr )
        {
            DoError( "\pError returned by GetEOF()...",
                    false );
            return( nil );
        }

```

Next, call FSRead() to read in the PICT's header. The header contains information about the PICT that we won't make use of here. For now, we'll read in the header, storing it in the buffer pictHeader.

Again, we used the reference number stored in `srcFile` to specify the PICT file. The second parameter plays two roles. As an input parameter, we'll use `headerSize` to specify the number of bytes we'd like to read. As an output parameter, `FSRead()` uses `headerSize` to tell us how many bytes were actually read in.

```
headerSize = kPICTHeaderSize;

if ( FSRead( srcFile, &headerSize, pictHeader )
    != noErr )
{
    DoError( "\pError reading file header...",
            false );
    return( nil );
}
```

Since `pictSize` is the size of the entire PICT file, and we don't need the header information, we'll reduce `pictSize` to the size of the PICT itself. We'll use this value to allocate enough memory for the picture.

```
pictSize -= kPICTHeaderSize;

if ( ( picture = (PicHandle)NewHandle( pictSize ) )
    == nil )
{
    DoError(
        "\pNot enough memory to read picture..."
        , false );
    return( nil );
}
```

Next, we'll lock this new block into memory, then pass a pointer to it on to `FSRead()`. Unless an error occurs, the PICT will be loaded into our new block of memory.

```
HLock( (Handle)picture );

if ( FSRead( srcFile, &pictSize, *picture ) !=
    noErr )
{
    DoError( "\pError returned by FSRead()...",
            false );
    return( nil );
}
```

Once the PICT is loaded, unlock the block, close up the file, and return the handle to the picture.

```

        HUnlock( (Handle)picture );
        FSClose( srcFile );

        return( picture );
    }
    else
    {
        DoError( "\pThe new FSSpec File Manager routines \
are not supported by this OS!", true );
    }
}

```

WindowInit() is pretty straightforward.

```

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
        DoError( "\pCan't load WIND resource...", true );

    ShowWindow( window );
    SetPort( window );
}

```

You've seen DrawMyPicture(), CenterPICT(), and DoError() before.

```

/***** DrawMyPicture *****/

void    DrawMyPicture( PicHandle picture )
{
    Rect        pictureRect;
    WindowPtr    window;

    window = FrontWindow();
}

```

```

        pictureRect = window->portRect;

        CenterPict( picture, &pictureRect );
        DrawPicture( picture, &pictureRect );
    }

/***** CenterPict *****/

void      CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top );
    OffsetRect( &pictRect, (windRect.right -
                pictRect.right)/2, (windRect.bottom -
                pictRect.bottom)/2 );
    *destRectPtr = pictRect;
}

/***** DoError *****/

void      DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString,
                kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}

```

---

## The Printing Manager

---

Chapter 7's final program, **PrintPICT**, demonstrates the proper use of the **Printing Manager**. PrintPICT loads a PICT from the resource file and prints the PICT on the currently selected printer.

## Calling the Printing Manager

Prepare the Printing Manager for use by calling `PrOpen()`. Then, allocate a new print record using `NewHandle()`. The print record contains information the Printing Manager needs to print your job, including **page setup** information and information specific to the **print job**.

You can prompt the user to fill in the page setup information by calling `PrStlDialog()`. Prompt the user for job-specific information with a call to `PrJobDialog()`. Each of these routines displays the appropriate dialog box and fills the newly allocated print record with the results.

Then, call `PrOpenDoc()` to set up a printing `GrafPort`. The printing `GrafPort` acts just like any other `GrafPort`, but exists only in memory, not on screen. `PrOpenDoc()` calls `SetPort()`, so any `QuickDraw` calls you make after this will apply to the printing `GrafPort`.

The printing `GrafPort` is made up of pages. You'll call `PrOpenPage()` to start a new page, then make a set of `QuickDraw` calls (like `DrawPicture()`) to fill the page with graphics. Next, call `PrClosePage()` to close the current page. Call `PrOpenPage()` and `PrClosePage()` for each page you want to create.

When you've drawn all your pages, close the document with a call to `PrCloseDoc()`. Now, it's time to print your document. Do this with a call to `PrPicFile()`. If background printing (spooling) is set up on your computer, you'll need to call `PrPicFile()` to start the printing process. If spooling is not set up, the call to `PrCloseDoc()` will start the process for you.

The Printing Manager is described in detail in *Inside Macintosh*, Volume II, Chapter 5 and was extended to handle Color `QuickDraw` in Volume V, Chapter 22. If you plan on writing an application that supports printing, read this chapter thoroughly.

Now, let's look at `PrintPICT`.

---

## PrintPICT

Since the "paperless society" seems to be receding rapidly into the distance, it's reasonable to expect a Mac application to be able to print. `PrintPICT` shows you how to print a `PICT` resource.



PrintPICT prints a PICT by drawing it in the printing GrafPort. Printing text is a little more complex. You'll draw the text in the printing GrafPort, but you'll need to worry about things like breaking text along the margins properly, and starting a new page once you hit the end of the current page. The Printing Manager doesn't care what you draw on the printing GrafPort—that's your responsibility

## PrintPICT Resources

PrintPICT makes use of three resources, the ALRT and DITL, used by `DoError()`, and the PICT that will be printed. Create a folder called PrintPICT inside your Development folder. Use ResEdit to create a new file called `PrintPICT.π.rsrc` inside the PrintPICT folder. Open the file `OpenPICT.π.rsrc` and copy both the ALRT and the DITL resources, pasting them in `PrintPICT.π.rsrc`.

Next, use your favorite graphics program to create a PICT resource. Unless you are using a color printer, you'll probably want to create a black and white image. Make sure the PICT's resource ID is 128 and make sure the **Purgeable** check box is checked. Quit ResEdit, saving your changes.

## The PrintPICT Project

Launch THINK C and create a new project in the PrintPICT folder. Call it `PrintPICT.π`. Use **Add...** from the **Source** menu to add MacTraps to the project.

Once MacTraps is added, open a new source code window and enter the program:

```
#include <PrintTraps.h>

#define kBaseResID      128

#define kDontScaleOutput  nil

#define kEmptyString    "\p"
#define kNilFilterProc   nil

#define kErrorAlertID    kBaseResID
```



```

/*****/
/* Functions */
/*****/

void      ToolBoxInit( void );
PicHandle LoadPICT( void );
THPrint   PrintInit( void );
Boolean   DoDialogs( THPrint printRecordH );
void      PrintPicture( PicHandle picture, THPrint
                        printRecordH );
void      CenterPict( PicHandle picture, Rect
                        *destRectPtr );
void      DoError( Str255 errorString, Boolean fatal );

/***** main *****/

void      main( void )
{
    PicHandle   picture;
    THPrint     printRecordH;

    ToolBoxInit();

    picture = LoadPICT();
    printRecordH = PrintInit();

    if ( DoDialogs( printRecordH ) )
        PrintPicture( picture, printRecordH );
}

/***** ToolBoxInit *****/

void      ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEFInit();
    InitDialogs( nil );
    InitCursor();
}

```

```

/***** LoadPICT *****/

PicHandle    LoadPICT( void )
{
    PicHandle    picture;

    picture = GetPicture( kBaseResID );

    if ( picture == nil )
        DoError( "\pCan't load PICT resource...", true );

    return( picture );
}

/***** PrintInit *****/

THPrint PrintInit( void )
{
    THPrint printRecordH;

    printRecordH = (THPrint)NewHandle( sizeof( TPrint ) );

    if ( printRecordH == nil )
        DoError(
            "\pNot enough memory to allocate print record"
            , true );

    PrOpen();
    PrintDefault( printRecordH );

    return( printRecordH );
}

/***** DoDialogs *****/

Boolean DoDialogs( THPrint    printRecordH )
{
    Boolean confirmed = true;

    confirmed = PrStdDialog( printRecordH );

    if ( confirmed )
        confirmed = PrJobDialog( printRecordH );

    return( confirmed );
}
```

```

/***** PrintPicture *****/

void    PrintPicture( PicHandle picture, THPrint
                    printRecordH )
{
    TPrPort    printPort;
    Rect       pictureRect;
    TPrStatus   printStatus;

    printPort = PrOpenDoc( printRecordH, nil, nil );

    PrOpenPage( printPort, kDontScaleOutput );

    if ( PrError() != noErr )
        DoError( "\pError returned by PrOpenPage()...",
                true );

    pictureRect = (**printRecordH).prInfo.rPage;

    CenterPict( picture, &pictureRect );
    DrawPicture( picture, &pictureRect );

    PrClosePage( printPort );
    PrCloseDoc( printPort );

    if ( (**printRecordH).prJob.bJDocLoop == bSpoolLoop )
        PrPicFile( printRecordH, nil, nil, nil,
                  &printStatus );

    PrClose();
    DisposHandle( (Handle)printRecordH );
}

/***** CenterPict *****/

void    CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top);
}

```

```

OffsetRect( &pictRect, (windRect.right -
                    pictRect.right)/2, (windRect.bottom
                    - pictRect.bottom)/2);
*destRectPtr = pictRect;
}

/***** DoError *****/

void DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString,
               kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

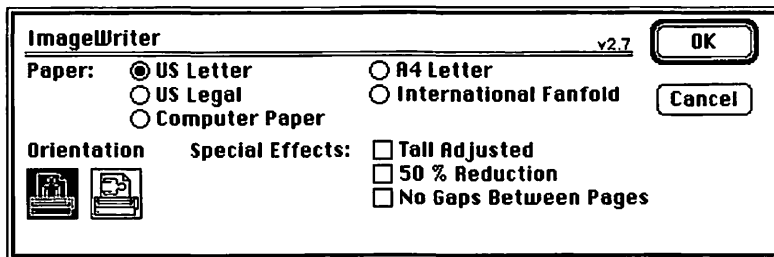
    if ( fatal )
        ExitToShell();
}

```

## Running PrintPICT

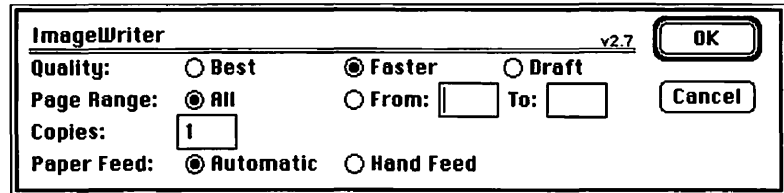
Once you've finished typing in the code, save it as `PrintPICT.c` and add it to the project using **Add** from the **Source** menu. Before you run your new program, use the **Chooser** to make sure your printer is on-line and ready for printing.

Once your printer is ready, select **Run** from THINK C's **Project** menu, clicking **Yes** to the question **Bring the project up to date?**. If the source code compiles correctly, a **Page Setup** dialog designed for your printer will appear (Figure 7.25).



**Figure 7.25** The ImageWriter's Page Setup dialog box.

If you click the **Cancel** button, PrintPICT will exit without printing your picture. If you click **OK**, a Job dialog box for your printer will appear (Figure 7.26). Again, clicking **Cancel** exits the program. If you click **OK**, your PICT image should start printing on your printer.



**Figure 7.26** The ImageWriter's Job dialog box.

## Walking Through the PrintPICT Code

OpenPICT starts with the `#include` containing the definitions necessary to use the Printing Manager.

```
#include <PrintTraps.h>
```

Most of these `#defines` should be familiar to you. `kDontScaleOutput` is used in the routine `PrintPicture()`, in its call to `PrOpenPage()`.

```
#define kBaseResID          128

#define kDontScaleOutput    nil

#define kEmptyString        "\p"
#define kNilFilterProc      nil

#define kErrorAlertID       kBaseResID
```

Ahhh, the function prototypes.

```

/*****/
/*  Functions  */
/*****/

void          ToolBoxInit( void );
PicHandle     LoadPICT( void );
THPrint       PrintInit( void );
Boolean       DoDialogs( THPrint printRecordH );
void          PrintPicture( PicHandle picture, THPrint
                        printRecordH );
void          CenterPict( PicHandle picture, Rect
                        *destRectPtr );
void          DoError( Str255 errorString, Boolean fatal );

```

main() calls LoadPICT() to load the PICT from the resource file. Next, PrintInit() is called to set up the printing environment.

```

/***** main *****/

void          main( void )
{
    PicHandle   picture;
    THPrint     printRecordH;

    ToolBoxInit();

    picture = LoadPICT();
    printRecordH = PrintInit();

```

After that, DoDialogs() is called to put up the **Page Setup** and **Print Job** dialog boxes. If either one exits by way of the **Cancel** button, DoDialogs() will return false. If DoDialogs() returns true, the picture is printed by PrintPicture().

```

        if ( DoDialogs( printRecordH ) )
            PrintPicture( picture, printRecordH );
}

```

Ahhh, the familiar ToolBoxInit().

```

/***** ToolBoxInit *****/

```

```

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

LoadPICT() uses GetPicture() to load the PICT resource. If the resource wasn't found, DoError() is called. Otherwise, a handle to the picture is returned.

```

/***** LoadPICT *****/

```

```

PicHandle    LoadPICT( void )
{
    PicHandle    picture;

    picture = GetPicture( kBaseResID );

    if ( picture == nil )
        DoError( "\pCan't load PICT resource...", true );

    return( picture );
}

```

PrintInit() starts by allocating a TPrint record. All the information relating to printing (including **Page Setup** and **Print Job** information) will be stored in this record. A handle to this record is stored in printRecordH and will be returned by PrintInit().

```

/***** PrintInit *****/

```

```

THPrint    PrintInit( void )
{
    THPrint    printRecordH;

    printRecordH = (THPrint)NewHandle( sizeof( TPrint ) );
}

```

```

if ( printRecordH == nil )
DoError(
    "\pNot enough memory to allocate print record"
    , true );

```

Next, `PrOpen()` is called to start up the Printing Manager. After that, `PrintDefault()` is called to load the print record with reasonable default values.

```

PrOpen();
PrintDefault( printRecordH );

```

Once the print record is initialized, a handle to it is returned.

```

return( printRecordH );
}

```

`DoDialogs()` calls `PrStlDialog()` to bring up the **Page Setup** dialog box. If the **OK** button is pressed, `PrStlDialog()` will return `true`.

```

/***** DoDialogs *****/

```

```

Boolean DoDialogs( THPrint    printRecordH )
{
    Boolean confirmed = true;

    confirmed = PrStlDialog( printRecordH );

```

In that case, `PrJobDialog()` is called to bring up the **Print Job** dialog box. If its **OK** button is pressed, `PrJobDialog()` will return `true`.

```

if ( confirmed )
    confirmed = PrJobDialog( printRecordH );

return( confirmed );
}

```



Normally, your application would bring up the **Page Setup** dialog in response to a **Page Setup...** menu selection and the **Print Job** dialog in response to a **Print...** menu selection. `PrintPICT` calls both dialogs for demonstration purposes only.



`PrintPicture()` calls `PrOpenDoc()` to open a new printing `GrafPort`.

```
/****** PrintPicture *****/
```

```
void      PrintPicture( PicHandle picture, THPrint
                        printRecordH )
{
    TPPrPort      printPort;
    Rect          pictureRect;
    TPrStatus     printStatus;

    printPort = PrOpenDoc( printRecordH, nil, nil );
```

`PrOpenPage()` is called to open a new page. The second parameter is used only for print spooling. If it is not `nil`, it points to a `Rect`, in the printing `GrafPort`'s local coordinates, to use as the frame for this page. All drawing will be scaled to fit inside the frame. Since we don't want our picture scaled, we passed `nil` as the second parameter.

```
PrOpenPage( printPort, kDontScaleOutput );
```

`PrError()` returns any error conditions caused by the last Printing Manager call. In general, you'll want to call `PrError()` after each Printing Manager call. We've done it here so you can see it in context. If an error exists, call `DoError()`.

```
if ( PrError() != noErr )
    DoError( "\pError returned by PrOpenPage()...",
            true );
```

So far, so good. Next, use the `rPage` field as a centering rectangle for the picture. `rPage` defines the boundary `Rect` for the current page.

```
pictureRect = (**printRecordH).prInfo.rPage;
```

Since the printing `GrafPort` is the current port, call `CenterPict()` and `DrawPicture()` to draw the picture on the current page.

```
CenterPict( picture, &pictureRect );
DrawPicture( picture, &pictureRect );
```

Finally, close the page and the doc and, if spooling is set up, call `PrPicFile()` to print the current document.

```

PrClosePage( printPort );
PrCloseDoc( printPort );

if ( (**printRecordH).prJob.bJDocLoop == bSpoolLoop )
    PrPicFile( printRecordH, nil, nil, nil,
               &printStatus );

```

Now that the document is printed, close the print driver and dispose of the memory you allocated for the print record.

```

PrClose();
DisposeHandle( (Handle)printRecordH );
}

```

CenterPict() and DoError() remain their same, old, humble selves.

```

/***** CenterPict *****/

void    CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (**( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top    - pictRect.top);
    OffsetRect( &pictRect, (windRect.right -
                            pictRect.right)/2, (windRect.bottom -
                            pictRect.bottom)/2);
    *destRectPtr = pictRect;
}

/***** DoError *****/

void    DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString,
               kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}

```

## In Review

---

We covered a lot in this chapter. Each of the six programs we presented involved a different part of the Mac Toolbox. If you're unsure about any of the concepts discussed, take the time to read about them in their respective *Inside Macintosh* chapters. The Resource Manager is covered in Volume I, Chapter 5.

The Control Manager is covered in Volume I, Chapter 10; Volume VI, Chapter 7; and Volume V, Chapter 12. Scroll bars make up a small part of these chapters, but the concepts implemented in Pager will carry through to other types of controls.

The Scrap Manager is covered in Volume I, Chapter 15 and Volume IV, Chapter 11. The Sound Manager is covered in Volume VI, Chapter 22, and completely replaces the discussions in Volume V, Chapter 27 and Volume I, Chapter 8.

The Standard File Package is covered in Volume I, Chapter 20 and updated in Volume IV, Chapter 15 and Volume VI, Chapter 26. The File Manager is covered in several places. Start with Volume VI, Chapter 25, then read Volume IV, Chapter 19. (*Warning:* the File Manager section in Volume II, Chapter 4, has been completely replaced by Chapter 19 of Volume IV.) Finally, the Printing Manager is covered in Volume II, Chapter 5 and Volume V, Chapter 22.

Now that you've completed your whirlwind tour of the Toolbox, you've probably already thought of 17 programs that people would pay big bucks for. Before you slap your homegrown software on a disk, read Chapter 8 so you can add custom icons to your application and documents.

See you there!

---

# Finishing Touches

*Now that you've mastered the Toolbox, you're ready to start building your own Macintosh applications. This chapter discusses some of the things you need to know before you send your application out into the world.*

---

By Now, you should have a good grip on the most important aspects of Macintosh application programming. We've described how to handle events, access files, and display pictures and text. You've worked with menus, windows, and dialogs.

This chapter discusses some issues that become important after you have your basic programming problems in hand. For starters, we'll show you the proper way to turn your application into a standalone application. You'll learn about the resources your application will need to interact smoothly with the Finder. Finally, we'll present some System 7-savvy sample code that shows you the correct way to respond to the required Apple events.

Let's get to work.

---

## Building a Standalone Application

---

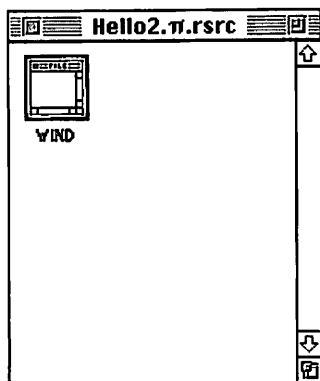
After you compile your debugged application, but before you announce your first stock offering, you'll want to turn your code into a Finder-savvy, standalone application. To do this, you'll use a set of resources known as the **Finder resources**.

The Finder resources serve many purposes. One group of Finder resources tells the Finder which icon to attach to your application. Another group helps define the text that appears in your application's **Get Info** window. One Finder resource determines how your application will interface with the Mac Operating System. As you'll see, the Finder resources play a critical role when you turn your code into a standalone Macintosh application.

### Creating the Finder Resources

Using ResEdit, we'll create a set of Finder resources and use them to turn Chapter 3's Hello2 program into a standalone application, complete with its own custom icon.

Fire up ResEdit, navigate over to the Hello2 folder, and open the file Hello2.π.rsrc. At this point, the file contains a single resource of type WIND (Figure 8.1). The first Finder resource you're going to create is the BNDL resource. The BNDL resource acts as a little database, tying together all of your application's Finder resources.



**Figure 8.1** The file `Hello2.π.rsrc`, before we add the Finder resources.

Select **Create New Resource** from the **Resource** menu and create a BNDL resource. The BNDL editing window will appear. Select **Extended View** from the **BNDL** menu, so that your window matches the one in Figure 8.2.

The first field you'll see in the BNDL editing window is the **Signature:** field. Your application's signature serves to distinguish your application from all others.

FREF			Finder Icons							
local	res ID	Type	local	res ID	ICN#	ic14	ic18	ics#	ics4	ics8

**Figure 8.2** ResEdit's BNDL editing window.

## Your Application's Signature

Every Macintosh application must have a **signature**, otherwise known as a **creator ID**. The signature is a 4-byte field, specified in THINK C's **Set Project Type...** dialog box. All of your application's files, whether it's the application itself or a document created by your application, share the same signature. When you double-click on a document's icon, the Finder notes the document's signature, then looks in its database for an application with the same signature. When it finds one, it launches the application, sending it a `kAEOpenDocuments` Apple event asking the application to open the specified document.

You can use just about any four-character sequence (numbers and symbols are fine) as a signature. There is one restriction. Apple has reserved all signatures consisting of four lower-case letters. `Eggo` is fine, `egGo` is fine, but `eggo` is off limits.

Once you've decided on a signature for your application, send a note to Apple's Developer Technical Support group, asking them to reserve your signature. (You can reach them through AppleLink at MacDTS.) You'll hear more about AppleLink and MacDTS in Chapter 9. For now, it's important to register your signature to avoid collisions with other applications.

We've registered a signature for the Hello2 application with MacDTS. The signature is `HELO`. You'll use this signature throughout the rest of this chapter.

## Editing the BNDL Resource

Back in the BNDL editing window, enter `HELO` in the **Signature:** field. Next, enter some text in the **© String:** field. This field is intended as a copyright notice, and will appear in the Finder's **Get Info** window for your application. Our **© String:** was:

**©1992, by D. Mark & C. Reed**

Hint: the © character is created by holding down the Option key and typing *g*.



Once you enter your **© String:**, the BNDL editor will create a new resource, using your application's signature as the resource type. The contents of the **© String:** field will be placed in this resource in the form of a Pascal string. In our case, the BNDL editor will create a `HELO` resource with a resource ID of 0. This resource is known as your **signature resource**.

Next, select **Create New File Type** from the **Resource** menu. A new line of six fields will appear in the bottom half of the window (Figure 8.3). This line represents a file type associated with your application. The six fields in the line tie an icon to that file type. For example, your application has a file type of **APPL**, which you set in THINK C's **Set File Type...** dialog. You'll need one line in your **BNDL** to tie an icon to the **APPL** file type.

The screenshot shows a window titled "BNDL ID = 128 from Hello2.p.rsrc". Inside, there are three input fields: "Signature:" with the value "HEL0", "ID:" with the value "0" (and a note "(should be 0)"), and "© String:" with the value "©1992, by D. Mark & C. Reed". Below these fields is a table with two main sections: "FREF" and "Finder Icons".

FREF			Finder Icons							
local	res ID	Type	local	res ID	ICN#	ic14	ic18	ics#	ics4	ics8
0	128	????	0	0						

**Figure 8.3** The **BNDL** editor, after **Create New File Type** was selected for the first time.

You'll create one line in your **BNDL** for every file type associated with your application. For example, if your application supports documents of type **PICT** and **TEXT**, you'll need two additional file type lines—one each for files of type **PICT** and **TEXT**.



Most applications create their own unique file type. For example, Microsoft Word saves its documents under the file type **WDBN**. If you require your own custom file type, add a line for it to the **BNDL** resource. And, oh yes, make sure you register the type with MacDTS. Just like signatures, unique file types must be registered with Apple.



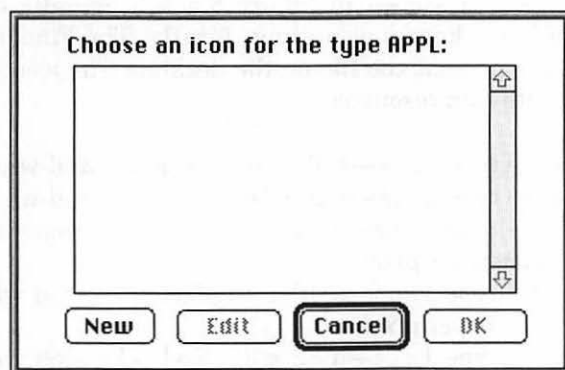
Let's take a closer look at the six fields in a **BNDL** file type line. The first three fields will be used to create the **FREF** Finder resource. The **FREF** turns a file type into a local ID. In a minute, we'll assign an icon to that ID. The **res ID** field determines the resource ID of the **FREF** resource. It should be set to 128.



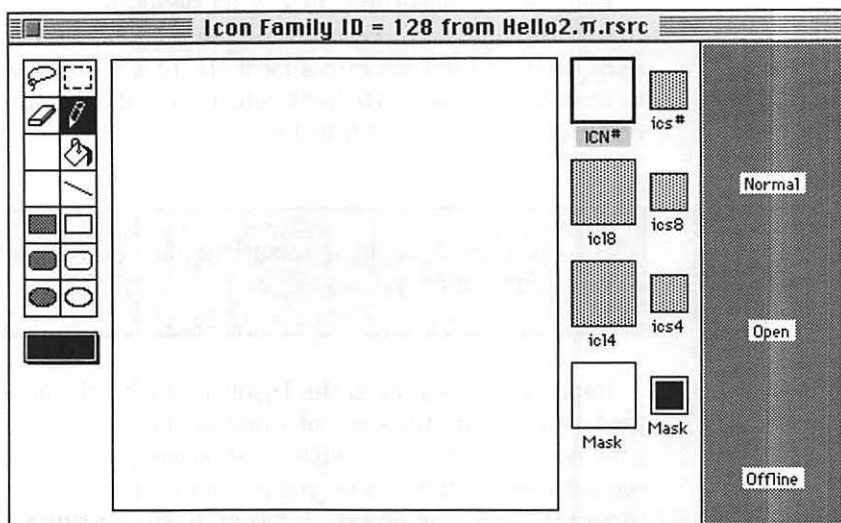
Basically, the local ID is used internally by the **BNDL**. You should never have to change its value.

Replace the four ?s in the **Type** field with the characters **APPL**. The next three fields tie a set of icons to the local ID. Since we want our icon resources to start with a resource ID of 128, enter 128 in the second **res ID** field. Now you're ready to create some icons. Click on the sixth field (the empty icons) and select **Choose Icon...** from the **BNDL** menu. The dialog in Figure 8.4 should appear, asking you to select an icon for file type **APPL**.

Since we haven't created an icon yet, click **New**. An icon editing window will appear (Figure 8.5).



**Figure 8.4** Choosing an icon for file type **APPL**.



**Figure 8.5** The icon editing window.

## Creating an Icon Family

The icon editor shown in Figure 8.5 will actually be used to edit a series of icons, known as an **icon family**. The Finder uses a file's icon family to represent the file on the desktop. The icon family consists of six separate icon resources:

- Resource type **ICN#**—a 32 x 32-pixel black-and-white icon.
- Resource type **ics#**—a 16 x 16-pixel black-and-white icon.
- Resource type **icl8**—a 32 x 32-pixel color icon with 8 bits of color information per pixel.
- Resource type **ics8**—a 16 x 16-pixel color icon with 8 bits of color information per pixel.
- Resource type **icl4**—a 32 x 32-pixel color icon with 4 bits of color information per pixel.
- Resource type **ics4**—a 16 x 16-pixel color icon with 4 bits of color information per pixel.

The Finder uses the member of the icon family that's appropriate for a given situation. For example, on a Macintosh with a 1-bit black-and-white monitor, the Finder makes use of the black-and-white icons found in the **ICN#** and **ics#** resources. In a color environment, the Finder makes use of the appropriate color resources.

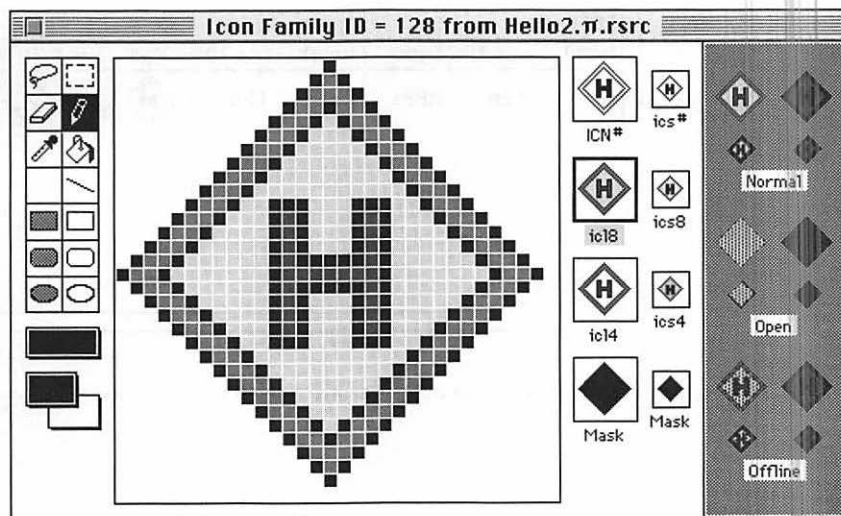
The large empty rectangle in the center of the icon editing window is the **icon edit panel**. To the left of the edit panel is a set of

MacPaint-like tools you'll use to create your icon images. To the right of the edit panel are two sets of icon images. The left set shows the six icon resources and a pair of **icon masks**, one 64 x 64 pixels and one 32 x 32 pixels. An icon mask is used to layout the clickable area of an icon. If the mask were blank, the user wouldn't be able to click on an icon. Usually, an icon's mask matches the outline of the icon and is filled in completely, leaving no holes.

The set of icons on the gray background on the right edge of the window show how the icons look normally, open, and off-line. The icons are shown in pairs. In each pair, the one on the left is unselected and the one on the right is selected. In each view you'll see four icons, two large and two small. The small ones are obtained by scaling down the large ones. Basically, your job is to edit the six icons and two masks until the views on the right look good.

Click on a resource and start editing. Start with the icon labeled ICN#. Since this is a black-and-white icon, no color tools are supported. However, if you click on one of the color icons, a color popup will appear, allowing you to draw in color.

Figure 8.6 shows the icon family we created for our Hello2 application. Notice that the masks are the same shape as the icon with the center completely filled in. Here are a few icon editing tips. Copy one icon onto another by clicking and dragging the first icon, releasing the mouse button when your cursor is over its destination. This technique is especially useful when creating a mask. Start with your ICN# icon. Once you are happy with it, drag it over to its little brother,



**Figure 8.6** Our Hello2 icon family, currently editing the icl8 resource.

the `ics#` icon. Once you get those two, drag them to the `ic18` and `ics8` icons. Now's the time to colorize your icons.

Once you're happy with your icon family, close the icon editing window. The icon family you just created should appear in the BNDL editing window, as shown in Figure 8.7. Don't let the shades of gray in our icon figures fool you—icons look great in living color!

Select **Create New File Type** in the **Resource** menu to create entries for the PICT and TEXT file types. Figure 8.8 shows our completed BNDL resource, with entries for files of type APPL, PICT, and TEXT. Notice that the PICT and TEXT entries have the familiar dog-eared document shape, while the APPL icon family has the standard application diamond shape.







Once your BNDL resource is complete, close the BNDL editing window and the BNDL picker window. Your main window should look like Figure 8.9. Notice that the BNDL editor created nine new resource types: the BNDL, FREF, and HELO resources, plus the six icon family resource types. Always use the BNDL editor to create these resources. The BNDL editor makes it easy to keep track of all the different Finder resource types.

**BNDL ID = 128 from Hello2.π.rsrc**

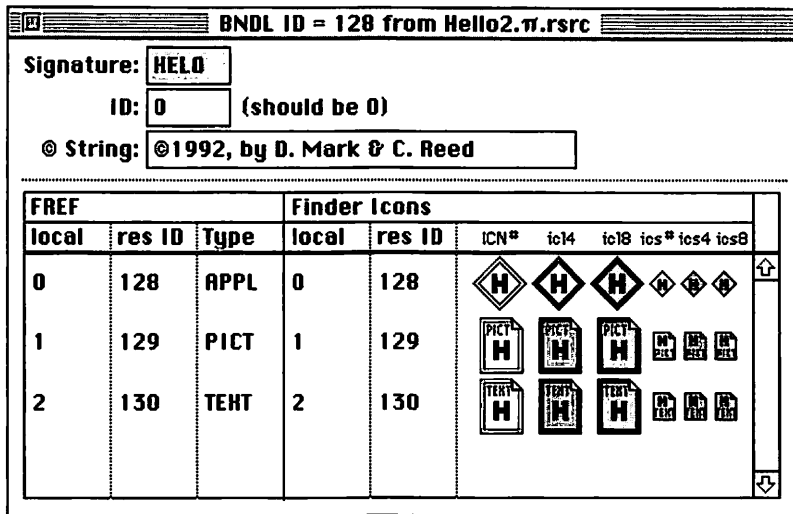
Signature: **HELO**

ID: **0** (should be 0)

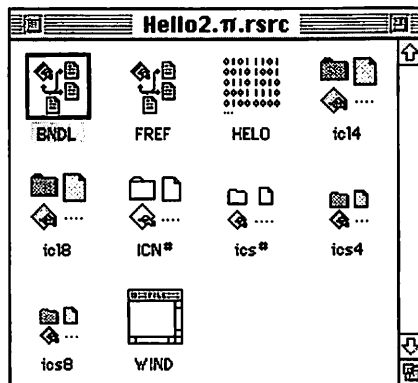
© String: **©1992, by D. Mark & C. Reed**

FREF			Finder Icons							
local	res ID	Type	local	res ID	ICN#	ic14	ic18	ics#	ics4	ics8
0	128	APPL	0	128						

**Figure 8.7** The BNDL resource with one icon family.



**Figure 8.8** The completed BNDL resource, with entries for APPL, PICT, and TEXT files.



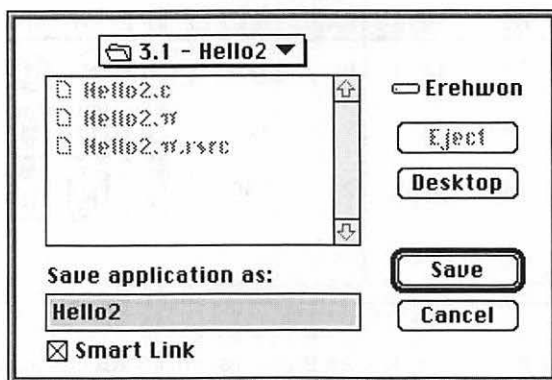
**Figure 8.9** Hello2.π.rsrc, after the BNDL resource is completed.

## Testing Your New Icon

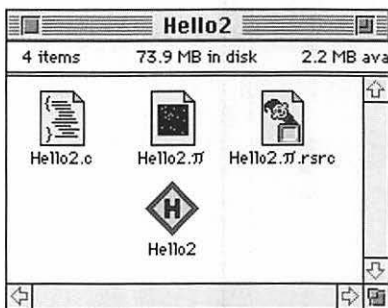
Quit ResEdit, saving your changes. Go into your Hello2 folder and double-click on the file Hello2.π, launching THINK C in the process. Select **Set Project Type...** from the **Project** menu. When the **Set Project Type...** dialog appears, enter **HEL0** in the **Creator** field, then click the **OK** button.

Next, select **Build Application...** from the **Project** menu. When the **Build Application...** dialog appears (Figure 8.10), enter **Hello2** in the **Save application as:** field and click the **Save** button.

THINK C will create an application file with the name **Hello2**. **Hello2** will have a type of **APPL** and a creator of **HELO**. Figure 8.11 shows the new application in a Finder window.



**Figure 8.10** THINK C's **Build Application...** dialog.



**Figure 8.11** Your new icon, in the **Hello2** folder.



If your custom icon didn't appear, carefully check your resources. If they all look OK, try rebuilding your desktop by rebooting, then holding down the command and option keys.

## Testing the PICT and TEXT Icons

You've tested the APPL portion of the Hello2 BNDL resource. Next, you'll use ResEdit to create document files with the HEL0 signature. You'll create one document with a type of PICT and one with a type of TEXT.

Quit THINK C and launch ResEdit. When the animated jack-in-the-box appears, click once and, when prompted for a file to open, click the **New** button. When prompted for a file name, navigate over to the Hello2 folder, enter **Hello2 PICT File**, and click the **New** button. ResEdit will create a new file and create a window showing the empty resource fork.

In the **File** menu, select **Get Info for Hello2 PICT File**. This will bring up a **Get Info** dialog box for your new file (Figure 8.12). Enter **PICT** in the **Type:** field and **HEL0** in the **Creator:** field.

**Info for Hello2 PICT File**

File: **Hello2 PICT File** ☐ Locked

Type: **rsrc** Creator: **RSED**

☐ File Locked ☐ Resources Locked File In Use: **Yes**  
☐ Printer Driver MultiFinder Compatible File Protected: **No**

Created: **Sat, Feb 22, 1992** Time: **4:48:08 PM**  
Modified: **Sat, Feb 22, 1992** Time: **4:48:09 PM**

Size: 286 bytes in resource fork  
0 bytes in data fork

---

Finder Flags: ☒ 7.x ☐ 6.0.x

☐ Has BNDL ☐ No INITs Label: **None** ▼  
☐ Shared ☐ Initied ☐ Invisible  
☐ Stationery ☐ Alias ☐ Use Custom Icon

**Figure 8.12** The **Get Info** window for Hello2 PICT File.

Click on the close box. When asked to save the changes, click **Yes**. After a slight delay, the icon for Hello2 PICT File should change to the one shown in Figure 8.13.



Hello2 PICT File

**Figure 8.13** The icon for Hello2 PICT File.

Now let's create a TEXT file with the HELO signature. Back in ResEdit, select **New** from the **File** menu. When prompted for a file name, enter **Hello2 TEXT File**. Now select **Get Info for Hello2 TEXT File** from the **File** menu. When the **Get Info** dialog box appears, enter **TEXT** in the **Type:** field and **HELO** in the **Creator:** field. Click on the close box. When asked to save the changes, click **Yes**. After a slight delay, the icon for Hello2 TEXT File should change to the one shown in Figure 8.14.



Hello2 TEXT File

**Figure 8.14** The icon for Hello2 TEXT File.

## Testing the Finder's Database

Once you've finished basking in the glory of your new icons, let's take them for a test drive. Quit ResEdit, going back to the Finder. Double-click on the Hello2 PICT File. Notice that the Finder launched the Hello2 application. Click once to exit Hello2, then try the same trick with Hello2 TEXT File.





If your Hello2 program contained an event loop, the Finder would have redrawn the application icon in the open mode, then sent a `kAEOpenDocuments` Apple event to Hello2, asking it to open Hello2 PICT File.

To try this yourself, throw the Hello2 application (not the two documents) into the trash, then empty it. Next, make a copy of Chapter 4's EventTracker folder, then open `EventTracker.π.rsrc` in ResEdit. Open `Hello2.π.rsrc` and copy all of the resources, *except* the WIND resource, into `EventTracker.π.rsrc`. Quit ResEdit, saving your changes.

Next, open your copy of `EventTracker.π` in THINK C. Use **Set Project Type...** to set the project type to `APPL` and the creator to `HELO`. Make sure the **SIZE Flags** are set to 5840. Use **Build Application...** to build an application, saving it as `EventTracker`. Quit THINK C.

The EventTracker icon should look like the Hello2 icon. (And why not? You're using the same resources!) Double-click on `EventTracker`. You'll see the familiar EventTracker window. Click on the desktop to return to the Finder.

First, take a look at the `EventTracker` icon. It should be displayed in the open mode, appearing as a gray-filled diamond. Double-click on the file `Hello2 TEXT File`. A `kAEOpenDocuments` Apple event should appear in the EventTracker window.

Click in the close box to exit `EventTracker`.

---

## More Finder Resources

---

At this point, you've seen the power of the Finder resources. You've used a `BNDL` resource to attach an icon to a file. More importantly, you've used the `BNDL` resource to alter the Finder's database. The `PICT` and `TEXT` files you created didn't contain any resources, yet they appeared with the correct icons and launched the correct programs.

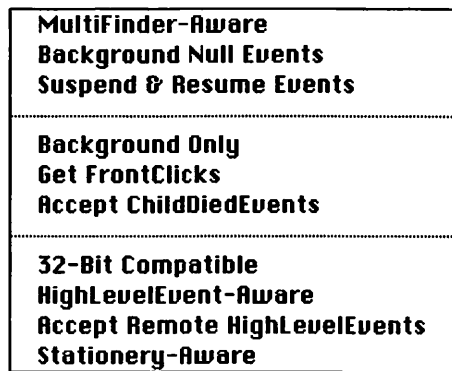
Before we move on to the proper handling of the required Apple events, we'll take a few moments to examine some of the other Finder resources.

## The SIZE Resource

There's one Finder resource you've already seen several times. The **SIZE** resource is created for you automatically by THINK C's **Set Project Type...** dialog. When the Finder launches your application, it looks for a **SIZE** resource with an ID of 0. If it can't find one, it looks for a **SIZE** resource with an ID of -1. If either is found, the Finder uses the flags found in the resource as a guide when launching the application.

THINK C automatically creates a **SIZE** resource for your application with an ID of -1. In certain cases, the Finder will create a **SIZE** resource for your application. For example, when you use the **Get Info** command to change your application's default memory requirements, the Finder will copy your existing **SIZE** resource (if one exists), creating a new one with an ID of 0. The changes specified in the **Get Info** window will be applied only to the **SIZE** resource with an ID of 0.

Figure 8.15 shows the **SIZE Flags** menu.



**Figure 8.15** The **SIZE Flags** menu from THINK C's **Set Project Type...** dialog.

The **MultiFinder-Aware** flag and the **Suspend & Resume Events** flag go hand in hand. If one is set, the other should be set. Together, they say that your application knows how to handle suspend and resume events, and is capable of intelligently making the shift from the foreground to the background. These two flags will almost always be set.

**Background Null Events** says that your program wants nullEvts while it is in the background. Set this flag if you need processing time, even when you are not in the foreground.

**Background Only** means that your program cannot run in the foreground and has no user interface.

If **Get FrontClicks** is set, your application will receive any `mouseDown` and `mouseUp` events used to bring your application to the foreground. If the flag is not set, your application will be brought to the foreground, but it will not see any mouse events used to bring it to the foreground.

**Accept ChildDiedEvents** tells the Mac OS to tell you if any processes you've launched died unexpectedly.

The **32-Bit Compatible** flag tells the Mac OS that you've tested your program and that it runs just fine in 32-bit mode. Don't set this flag unless you've tested your application in 32-bit mode.

**HighLevelEvent-Aware** asks the Mac OS to send you all high-level events, including Apple events. From now on, you'll be writing only high-level aware applications.

**Accept Remote HighLevelEvents** asks the Mac OS to send you high-level events that come over the network. If you set the **HighLevelEvent-Aware** bit, you'll usually set this bit as well.

Finally, **Stationery-Aware** tells the Finder that your application can handle stationery pads. What are stationery pads? Glad you asked...

## Stationery Pads

A **stationery pad** is a document used as a template by your application. For example, in a word-processing package, users might create a document with their corporate logo on the top, saving the document as a stationery pad. Whenever they click on the stationery pad's icon, a new, untitled document appears *with the corporate logo already in it*. Stationery pads allow your users to create a set of canned documents they can use again and again.

As you design your application, you'll have to decide which file types the user can save as stationery pads. For example, suppose your application supports two file types, `PICT` and `TEXT`. You might allow the user to use a `TEXT` document as the basis for stationery, but not a `PICT` document.

You'll need to design a unique icon family for each file type that supports stationery. To link a stationery icon family to its parent file type, add the icon family to the `BNDL` resource, using an `FREF` type that matches the parent file type, with the first letter changed to *s*. For example, to support `TEXT` stationery, add an icon family with a type of `sEXT` to the `BNDL`. To support `PICT` stationery, add an icon family with a type of `sICT` to the `BNDL`.



If your application supports stationery, you'll have to make sure the file types you support are unique in their last three characters.

When users indicate that they'd like to save a document as stationery, they'll also have to indicate the file type they'd like the stationery based on. If you support more than one file type, you might try the interface shown in Figure 8.16. The **File Format:** popup allows the user to select the file type, and the **Save as stationery pad** check box specifies whether the file is to be saved as stationery.

If the document is to be saved as stationery, save the document using the selected type (PICT, not sICT; TEXT, not sEXT) and mark the document as stationery. The File Manager maintains a set of **Finder flags** for every file and folder it knows about. One of these flags is the `isStationery` flag. When saving a document as stationery, you'll set the `isStationery` flag. We'll show you how to do this in the Finder information section, coming up next.

Before the Finder draws a file's icon, it checks to see if that file's `isStationery` flag is set. If so, it substitutes an *s* for the first letter of the file type, then looks in the file's signature BNDL for an icon family with this modified type. For example, if it encounters a file with a signature of HELO, a type of TEXT, and a set `isStationery` flag, the Finder looks for an icon family with a signature of HELO and a type of sEXT. If found, this stationery icon family is used to draw the file's icon. If a stationery icon family is not found, a default stationery icon is drawn.

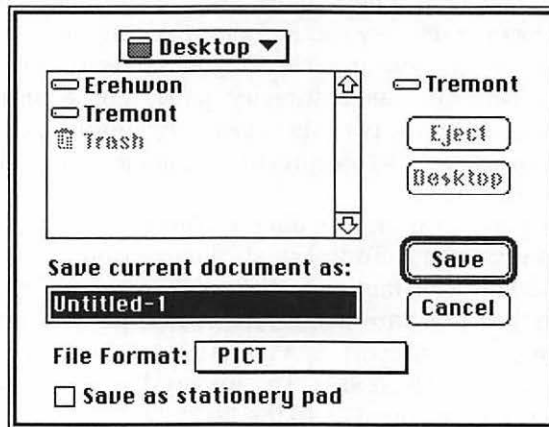


Figure 8.16 A sample stationery pad **Save As...** dialog.

In summary, if you plan on supporting stationery, create an icon family for each file type that will serve as a basis for stationery, and make sure you set the **Stationery-Aware** flag in your application's SIZE resource. You'll find more information about stationery pads in *Inside Macintosh* (VI: 9-26–9-27).

## Finder Information

The File Manager maintains a series of data structures that describe the files and folders accessible by the Finder. One of these data structures, the `FInfo`, contains the Finder flags referred to earlier:

```
struct FInfo
{
    OSType          fdType;
    OSType          fdCreator;
    unsigned short  fdFlags;
    Point           fdLocation;
    short           fdFldr;
};
```

A file's `FInfo` holds its type and creator (`fdType` and `fdCreator`), its Finder flags (`fdFlags`), its location within its Finder window (`fdLocation`), and a short specifying whether the file is in the trash, on the desktop, or in a window (`fdFldr`).

The `fdFlags` field is an unsigned short, referenced by bits numbered 0 through 15:

- **Bit 0** is reserved.
- **Bits 1–3** indicate the file's color.
- **Bits 4–5** are reserved.
- **Bit 6** is set only if the file is an application and can be opened by multiple users.
- **Bit 7** is set only if the file contains no INIT resources.
- **Bit 8** is set when the Finder has stored the Finder information in its desktop database.
- **Bit 9** is reserved.
- **Bit 10** is set if the file has a custom icon. A user can create a custom icon by selecting **Get Info** for a file, then pasting a picture into the **Get Info** window. Custom icons are described in *Inside Macintosh* (VI: 9-28).
- **Bit 11** is set if the file is a stationery pad. This is the `isStationery` bit we referred to earlier.
- **Bit 12** is set if the file's name is locked and can't be changed.
- **Bit 13** is set if the file contains a BNDL resource.

- **Bit 14** is set if the file is invisible.
- **Bit 15** is set if the file is an alias.

To change a file or folder's Finder information, go into ResEdit and select **Get File/Folder Info...** from the **File** menu. When prompted, select a file or folder. The **Get Info** window for that file or folder will appear. When you were editing the Hello2 BNDL, you saw an example of this window (Figure 8.12).

To change the Finder information from within your program, use the File Manager routines `GetFInfo()` and `SetFInfo()` (IV:113).

## The vers Resources

Next on our list of Finder resources is the `vers` resource. You'll create two `vers` resources for your applications. The `vers` resource with an ID of 1 describes version information for a particular file. This version information includes a version number, the version's release level, country code, and a text string that will appear in the Finder's **Get Info** window for the file.

The `vers` resource with an ID of 2 contains the same information, but is used to describe a set of files, instead of a single file. For example, the `vers 1` resource might describe a configuration file for your application, citing the release number and date of the configuration file. The `vers 2` resource, on the other hand, would describe the version information of the application itself.

Try this yourself. Use ResEdit to build `vers 1` and `vers 2` resources for your Hello2 application. Figure 8.17 shows a sample `vers 1`

The screenshot shows a window titled "vers ID = 1 from Hello2.rsrc". Inside the window, there are several fields for version information:

- Version number:** Three input boxes containing "1", "0", and "0" separated by dots.
- Release:** A dropdown menu showing "Final".
- Non-release:** An input box containing "0".
- Country Code:** A dropdown menu showing "00 - USA".
- Short version string:** An input box containing "1.0, ©1992, Mark & Reed".
- Long version string (visible in Get Info):** A larger input box containing "1.0 (US), ©1992 by Dave Mark and Cartwright Reed".

**Figure 8.17** The `vers 1` resource for Hello2.

resource for the Hello2 application. Notice that the file name isn't included in the version string. The Finder will do that for you. Be sure to include both a short and a long version string. The Finder uses the long string in the **Get Info** window, but there will be times when the short string is used.



By the way, if the Finder can't find a `vers 1` resource, it will use the string found in the signature resource instead.

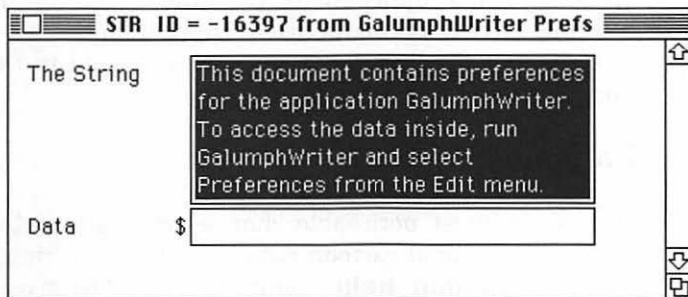
You'll find more information on the `vers` resource in *Inside Macintosh* (VI: 9-23-9-24).

## Some Useful STR Resources

When you double-click on a document icon and the Finder can't find an application with a matching signature, the Finder takes one of two actions. If the document is of type `PICT` or `TEXT` and TeachText is present, the document is opened with TeachText. If the document is of another type, or if TeachText is not present, the Finder puts up an **Application Not Found** alert.

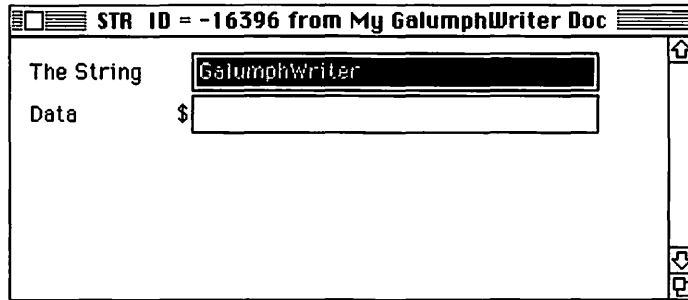
There are two different `'STR '` resources you can add to a document that alter the appearance of this alert. The first, a `'STR '` resource with a resource ID of `-16397`, is known as the **message string resource**. The second, a `'STR '` with an ID of `-16396`, is known as the **name string resource**.

If the document was not meant to be opened, include a message string resource in the document explaining the proper way to work with the document. Figure 8.18 shows a sample message string resource.



**Figure 8.18** A message string resource from the GalumphWriter preferences file.

If the document was meant to be opened, include a name string resource consisting of the name of your application (Figure 8.19). The Finder will include the string in an alert of the form, “**The document xxx could not be opened, because the application yyy could not be found,**” where **xxx** is the document’s name and **yyy** is the name string resource.



**Figure 8.19** A name string resource from a GalumphWriter document.

---

## The Help and Edition Managers

---

Before we move on to Apple events, there are a few more Toolbox managers you need to know about. The Help Manager implements a fun and informative help mechanism taken straight from the Sunday comics. Point the mouse at an item on the screen and a cartoon balloon appears, describing the object in question. This mechanism is known as **balloon help**.

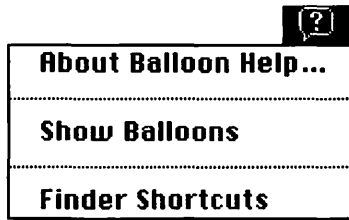
The Edition Manager allows your application to **publish** data as an **edition**. Other applications can **subscribe** to that same edition, creating a live data link between applications.

These two Toolbox managers are discussed in the next few pages. Then, it's on to Apple events!

### Balloon Help

One of the most noticeable changes brought on by System 7 is the addition of a small cartoon balloon icon on the right side of the menu bar. The **Balloon Help** menu (Figure 8.20) gives Macintosh users access to fast, friendly help whenever they drag their mouse over an item that supports balloon help. Balloon help is turned on by selecting

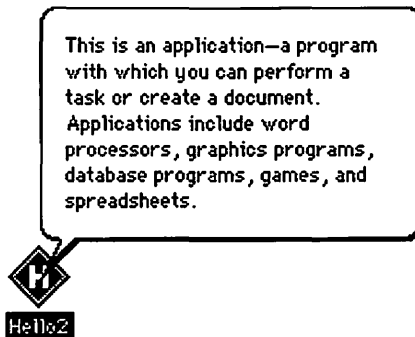




**Figure 8.20** The Balloon Help menu.

**Show Balloons** from the **Balloon Help** menu. The **Balloon Help** menu is always available, even when a modal dialog is running.

Figure 8.21 shows an example of balloon help in action. This particular balloon appears in the Finder when the mouse moves over an application icon. The Finder provides help balloons for practically everything you see on the screen: menu titles, menu items, icons, even windows.



**Figure 8.21** The balloon help offered by the Finder when the mouse passes over the Hello2 icon.

You can (and should) include help balloons in your own applications. To do this, create a series of **help resources**, each of which attaches a help balloon to a part of your program's interface.

The current version of ResEdit doesn't come with built-in help resource templates, and that's a problem when it comes to help resource design. Luckily, there are a few alternatives. Apple's BalloonWriter program is specifically designed to construct help balloons and attach them to your files. You can buy BalloonWriter directly from APDA (APDA is discussed in Chapter 9) for a modest fee.

You can also create and edit help resources in Resourcerer, the alternative resource editor from Mathemaesthetics, in Chestnut Hill, Massachusetts. Resourcerer is a very powerful resource editor and is definitely worth checking out.

A complete discussion of the Help Manager is contained in *Inside Macintosh*, Volume VI, Chapter 11. If you're going to write a truly System 7-savvy program, you must read this chapter.

## Edition Manager

Before we hit Apple events, there's one more topic to cover: the Edition Manager. As we mentioned earlier, the Edition Manager allows your application to **publish** data as an **edition**. Other applications can **subscribe** to that same edition, creating a live data link between applications.

For example, suppose you wanted to include a spreadsheet table in a word-processing document. Normally, you'd open the spreadsheet, copy the table, open the word processor, then paste the table into the word-processing document. If the data in your table changes, you have to repeat this process all over again.

If your spreadsheet and word processor support the Edition Manager, you can save yourself a lot of work. Open the spreadsheet and publish the table as an edition. Next, open the word-processing document and subscribe to the edition, placing the table where you want it. The live data link provided by your spreadsheet subscription means that if that table ever changes, your word-processing document will be updated as well.

You can read all about the Edition Manager in *Inside Macintosh*, Volume VI, Chapter 4. If you decide to add publish and subscribe to an application, you'll need to add an icon family for each edition file type you'll support to the BNDL resource (VI: 9-27).

---

## Responding to the Required Apple Events

---

Earlier in the chapter, we used some Finder resources to link the Hello2 application to some PICT and TEXT documents, all of which shared a common signature. One key point that emerged from this discussion was the sequence of events that occur when you select a document and select **Open** from the Finder's **File** menu.

First, the Finder looks for an application whose signature matches the selected document. Next, the Finder launches the application, sending it a `kAEOpenApplication` Apple event. Finally, the Finder sends the application a `kAEOpenDocuments` Apple event, asking the

application to open the specified document.

In the same vein, if **Print** was selected from the **File** menu instead of **Open**, the Finder follows these same steps, ending with a `kAEPrintDocuments` Apple event instead of the `kAEOpenDocuments` Apple event.

Our goal for the remainder of this chapter is to follow this algorithm all the way to its conclusion. Once your program receives one of the required Apple events, how should it respond? Chapter 4's `EventTracker` program showed you how to install event handlers for the required Apple events. Now we'll take a look inside the handlers themselves.

## A Quick Review of the EventTracker Code

Before entering its main event loop, `EventTracker` uses `AEInstallEventHandler()` to install an event handler for each of the four required Apple events:

```
err = AEInstallEventHandler( kCoreEventClass,
                             kAEOpenApplication, DoOpenApp, 0L,
                             false );
if ( err != noErr )
    DrawEventString(
        "\pkAEOpenApplication Apple event notavailable!" );

err = AEInstallEventHandler( kCoreEventClass,
                             kAEOpenDocuments, DoOpenDoc, 0L,
                             false );
if ( err != noErr )
    DrawEventString(
        "\pkAEOpenDocuments Apple event not available!" );

err = AEInstallEventHandler( kCoreEventClass,
                             kAEPrintDocuments, DoPrintDoc, 0L,
                             false );
if ( err != noErr )
    DrawEventString(
        "\pkAEPrintDocuments Apple event not available!" );

err = AEInstallEventHandler( kCoreEventClass,
                             kAEQuitApplication,
                             DoQuitApp, 0L, false );
if ( err != noErr )
    DrawEventString(
        "\pkAEQuitApplication Apple event not available!" );
```

The four Apple events `kAEOpenApplication`, `kAEOpenDocuments`, `kAEPrintDocuments`, and `kAEQuitApplication` will be sent to the four routines `DoOpenApp()`, `DoOpenDoc()`, `DoPrintDoc()`, and `DoQuitApp()` respectively.

Here are the prototypes for the four handlers:

```
pascal OSErr DoOpenApp( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon );
pascal OSErr DoOpenDoc( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon );
pascal OSErr DoPrintDoc( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon );
pascal OSErr DoQuitApp( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon );
```

In `EventTracker`, each handler printed a message in the `EventTracker` window. In real life, you'll need to do a bit more than that. Let's take a fresh look at each of these handlers.

## Responding to `kAEOpenApplication`: `DoOpenApp()`

`DoOpenApp()` is probably the easiest of the four required Apple event handlers to write. If you want your application to start up with a new, untitled document window, `DoOpenApp()` should call some code that creates a new, untitled document window. Here's a sample:

```
#define kCantCreateWindowErr    -1

pascal OSErr DoOpenApp( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon )
{
    WindowPtr    window;

    window = CreateWindow();

    if ( window == nil )
        return( kCantCreateWindowErr );

    ShowWindow( window );

    return( noErr );
}
```

In this example, the parameters to `DoOpenApp()` are ignored. The routine `CreateWindow()` should create a new, untitled document window, placing it properly on the screen.



In this example, `CreateWindow()` returns `nil` if the window could not be created. In that case, `DoOpenApp()` returns an error code #defined above. Whether you adopt this particular error-handling model or not, be sure to integrate your Apple event handlers into your overall error-handling strategy.

## Responding to `kAEOpenDocuments: DoOpenDoc()`

The first parameter to `DoOpenDoc()`, `theAppleEvent`, contains, among other things, a list of files to open. `DoOpenDoc()`'s job is to pull out the list of files, one by one, and pass the file specification on to a routine that will read in the file, displaying its contents in a new window.

The second parameter, `reply`, is also an Apple event, though at this point it is empty. Many Apple events require a reply from the handler. Sometimes the reply contains data requested in the original Apple event, sometimes the reply is sent to inform the originator of an error. The four required events don't require a reply.

The third parameter, `refCon`, contains whatever value was passed to it when the handler was installed with `AEInstallEventHandler()`. You can use this value as you like.

```
pascal OSErr DoOpenDoc( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon )
{
```



Both `theAppleEvent` and `reply` refer to structs, and would normally be passed using a pointer. In this case, however, the routine is declared using the `pascal` keyword, and all parameters will be passed using the Pascal parameter-passing mechanism.

Remember, this rule applies only to routines declared with the `pascal` keyword. `DoOpenDoc()` is declared this way because it is called by the Toolbox and not directly by our own code.

The variables declared below will be discussed in context.

```
AEDescList  fileSpecList;
FSSpec      file;
OSErr       err;
DescType    type;
Size        actual;
```

```
long      count;  
AEKeyword keyword;  
long      index;  
WindowPtr window;
```

Apple events contain two types of information. An Apple event **attribute** describes the event class (`kCoreEventClass`), event ID (`kAEOpenDocuments`), or some other characteristics of the Apple event. `DoOpenApp()` knows everything it needs to about this event. It wouldn't be called unless a `kAEOpenDocuments` event occurred. If we needed more information about the attributes of the `AppleEvent`, we could call `AEGetAttributePtr()` or `AEGetAttributeDesc()`.

The second type of information embedded in an Apple event is known as a **parameter**. A parameter contains the Apple event's data. In this case, the parameter we're interested in is a list of files to open. Typically, parameters are wrapped in data structures known as **descriptors**. A descriptor contains a data type, followed by the data of that type.



Since the Apple Event Manager provides a set of routines to access both parameters and attributes, there's no need to worry about the details of the different Apple event data structures. We strongly recommend, however, that you read the Apple Event Manager chapter in Volume VI of *Inside Macintosh*.

`AEGetParamDesc()` takes a pointer to an Apple event and returns a list of parameter descriptors embedded in the Apple event. `keyDirectObject` tells `AEGetParamDesc()` that the requested data is found in the descriptor record itself. `typeAEList` tells `AEGetParamDesc()` that the data will be in the form of a list.

Finally, the data is returned in a parameter of type `AEDescList`, a list of descriptors, each of which contains a file specification.

```
err = AEGetParamDesc( &theAppleEvent, keyDirectObject,  
                      typeAEList, &fileSpecList );
```

If `AEGetParamDesc()` returns an error, you'll want to call an error-handling routine of your own design. Depending on your error-handling strategy, you might want to send a return Apple event that describes the problem, you might want to log the error, or you might want to ask the user for guidance. As you learn more and more about Apple events, your error-handling strategy will get more and more sophisticated. For now, you might want to write a `DoAEEError()` that beeps once and then returns.

```

if ( err != noErr )
{
    DoAError( &theAppleEvent, &reply, err );
    return( err );
}

```

Once the file specification list is pulled out of the Apple event, you need to make sure you received only those parameters you were supposed to. In this case, we should receive only a single parameter, the list of file specifications. If any additional parameters were received, call `DoAError()`. The code for `GotRequiredParams()` is listed following `DoOpenDoc()`.

```

err = GotRequiredParams( &theAppleEvent );
if ( err != noErr )
{
    DoAError( &theAppleEvent, &reply, err );
    return( err );
}

```

`AECCountItems()` takes a list of descriptors and returns the number of descriptors in the list. In this case, `count` will contain the number of files we were asked to open.

```

err = AECCountItems( &fileSpecList, &count );
if ( err != noErr )
{
    DoAError( &theAppleEvent, &reply, err );
    return( err );
}

```

Next, enter a loop, counting from 1 to the number of files to be processed.

```

for ( index = 1; index <= count; index++ )
{

```

Inside the loop, call `AEGetNthPtr()`, passing it the descriptor list. `AEGetNthPtr()` returns the appropriate file in the form of a `FSSpec`.

```

    err = AEGetNthPtr( &fileSpecList, index,
        typeFSS, &keyword, &type, (Ptr)&file,
        sizeof( FSSpec ), &actual );

```

```

    if ( err != noErr )
    {
        DoAError( &theAppleEvent, &reply, err );
        return( err );
    }

```

If no problem occurred, pass the FSSpec on to a routine that will read in the file, displaying its contents in a window.

```

    if ( window = CreateWindowFromFile( &file ) )
        ShowWindow( window );

```

If `CreateWindowFromFile()` returns nil, return the same error code #defined for `DoOpenApp()`.

```

    else
    {
        DoAError( &theAppleEvent, &reply,
                  kCantCreateWindowErr );
        return( kCantCreateWindowErr );
    }

    return( noErr );
}

```

Here's the code for `GotRequiredParams()`, mentioned earlier:

```

OSErr    GotRequiredParams( AppleEvent *appleEventPtr )
{
    DescType    returnedType;
    Size        actualSize;
    OSErr       err;

```

`AEGetAttributePtr()` looks for the specified attribute, returning an error that indicates the attribute's status. This set of parameters will return the first required parameter that we haven't already retrieved. Since we think we've pulled the required parameters already, we're hoping `AEGetAttributePtr()` returns the error `errAEDescNotFound`, indicating that there are no more required parameters.

```

    err = AEGetAttributePtr( appleEventPtr,
                             keyMissedKeywordAttr, typeWildCard,
                             &returnedType, nil, 0, &actualSize );

```



If `AEGetAttributePtr()` returns `noErr`, another parameter was found and we had better return an error.

```

    if ( err == errAEDescNotFound )
        return( noErr );
    else if (err == noErr )
        return( errAEEventNotHandled );
    else
        return( err );
}

```

## Responding to `kAEPrintDocuments: DoPrintDoc()`

`DoPrintDoc()` is virtually identical to `DoOpenDoc()`:

```

pascal OSErr DoPrintDoc( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon )
{
    AEDescList      fileSpecList;
    FSSpec          file;
    OSErr           err;
    DescType        type;
    Size            actual;
    long            count;
    AEKeyword       keyword;
    long            index;
    WindowPtr       window;

```

First, pull the file descriptor list out of the Apple event.

```

    err = AEGetParamDesc( &theAppleEvent, keyDirectObject,
                        typeAEList, &fileSpecList);
    if ( err != noErr )
    {
        DoAError( &theAppleEvent, &reply, err );
        return( err );
    }

```

Next, make sure all of the required parameters were handled.

```

    err = GotRequiredParams( &theAppleEvent );
    if ( err != noErr )
    {

```

```

        DoAError( &theAppleEvent, &reply, err );
        return( err );
    }

```

Count the number of files to print.

```

    err = AECCountItems( &fileSpecList, &count );
    if ( err != noErr )
    {
        DoAError( &theAppleEvent, &reply, err );
        return( err );
    }

```

Loop on this number, pulling out the FSSpecs, one at a time.

```

    for ( index = 1; index <= count; index++ )
    {
        err = AEGGetNthPtr( &fileSpecList, index, typeFSS,
                           &keyword, &type, (Ptr)&file,
                           sizeof( FSSpec ), &actual );

        if ( err != noErr )
        {
            DoAError( &theAppleEvent, &reply, err );
            return( err );
        }
    }

```

Here's the difference. You'll still load the data from the file, creating a (still invisible) window. Next, you'll call a routine that prints the window's contents. Once the data is printed, dispose of the window.

```

        if ( window = CreateWindowFromFile( &file ) )
        {
            PrintWindow( window );
            CloseWindow( window );
        }
        else
        {
            DoAError( &theAppleEvent, &reply,
                     kCantCreateWindowErr );
            return( kCantCreateWindowErr );
        }
    }

    return( noErr );
}

```

## Responding to kAEQuitApplication: DoQuitApp()

DoQuitApp() gets called when someone wants your application to exit. For example, if your application is running and the user selects **Restart** or **Shutdown** from the Finder's **Special** menu, the Finder will send your application a kAEQuitApplication Apple event, and your DoQuitApp() handler will be called.

```
pascal OSErr DoQuitApp( AppleEvent theAppleEvent,
                        AppleEvent reply, long refCon )
{
    OSErr err;
```

Typically, you'll write a single routine to handle quitting. In this case, DoQuitApp() calls a routine called CloseDown(). CloseDown() walks through the list of application windows, closing each one in turn. If a window's contents have not been saved, the user will be prompted to save or discard the contents.

If CloseDown() was able to close all windows, it returns true, and DoQuitApp() returns noErr. If a problem is encountered, CloseDown() will return false, and DoQuitApp() will return an error. In either case, gDone is set to true and the program exits.

```
    if ( CloseDown() )
        err = noErr;
    else
        err = errAEWaitCanceled;
    gDone = true;

    return( err );
}
```

---

## In Review

---

We've covered a lot of material in this chapter. Most of the concepts are described in great detail in *Inside Macintosh*, Volume VI. Although we've tried to get you started, you owe it to yourself to read Volume VI cover to cover. Here are a few key chapters:

Start with Chapters 1 and 2. Read these straight through. Chapter 1 is an overview of the book and is quite short. Chapter 2 discusses important user interface issues.

Next, check out Chapter 9, the Finder Interface chapter. This chapter details the Finder resources and adds a few we didn't cover in this chapter.

Chapter 6 presents the Apple Event Manager. There is a lot of detail in this chapter, so don't try to absorb it all in one sitting. Before you start, take the concepts in this chapter and try your hand at a small application that implements the required Apple event handlers. If you like, use the coupon in the back of this book to send away for the *Mac Primer* source code disk. We've included a complete Apple events program that you can use as a model.

Next, check out Chapters 4 (the Edition Manager) and 11 (the Help Manager). Once that's done, start on the rest of the chapters.

**Our** last chapter discusses the issues you'll face as you start developing your own Macintosh applications. We'll start by taking a look at a few Mac periodicals you may find useful. We'll talk about *Inside Macintosh* and other Apple technical references, and we'll look at Apple's support apparatus for Macintosh programmers and developers.

---

---

# The Final Chapter

*To successfully develop software for the Macintosh, you need current technical information. You need to know how to use the standard Macintosh references effectively. You also need to know about the different technical support programs Apple offers. In this chapter, we'll discuss these and other Mac development issues.*

---

---

THE BASICS OF programming the Macintosh have been laid out in the eight preceding chapters. Familiarity with these basics is half the job of becoming a successful developer. The other half is understanding how the Mac programming world works and knowing where to get the information you, as a Macintosh software developer, will need.

This chapter investigates the periodicals that are your link to the Macintosh community. It looks at *Inside Macintosh* and other Mac technical texts, as well as software tools, from compilers to debuggers. The chapter also examines Apple's support programs for Macintosh software developers.

The *Macintosh Programming Primer* is your passport to Mac application programming. When you've finished reading this book, join a local Mac user group, and buy a copy of the best Mac programmer's magazine, *MacTutor*. Get involved and write some code!

---

## Macintosh Periodicals

---

Whether you're interested in creating a commercial product or a shareware product, or whether you just want to know the latest news from the Mac community, read the trade magazines. *MacWeek* is great, and *PCWeek* and *InfoWorld* are good, if less oriented to the Macintosh computer line. All three magazines deliver timely dollops of news: the new software packages, scoops on company goings-on, and juicy industry gossip.

The Macintosh programming journal is *MacTutor*, an invigorating monthly discourse on the art of Mac programming. Popular Mac magazines include *MacUser* and *MacWorld*. Their broad viewpoint can show you what's of interest to Macintosh users and what's available.

While you wait for the idea that will make you the seventh richest person in the world, you need to learn the Macintosh inside and out. To do this, you need *Inside Macintosh*.

---

## The Essential *Inside Macintosh*

---

The *Inside Macintosh* technical reference series is written by Apple and published by Addison-Wesley. As this book went to press, Apple was hard at work reorganizing the entire series. The new *Inside Macintosh* series will consist of 16 books published between summer of 1992 and spring of 1993. The series is described in more detail in

Appendix H. Although the new version of *Inside Macintosh* will have a new look, the information in the old series should remain valid for quite some time.

Since the new version hasn't hit the bookstores yet, here's a description of the seven books that make up the current series (Volumes I–VI and the *Inside Macintosh X-Ref*).

Volumes I, II, and III represent the Mac technical world as it was before the Mac Plus was introduced. All three volumes focus on the original 128K Mac, describing interfaces to the ROM routines, memory management, hardware specs, and more.

Volume IV was released after the Mac Plus and the Mac 512KE were introduced. Both of these new Macs sported 128K ROMs (as opposed to the 128K Mac's 64K ROMs). These larger ROMs contain the routines that handle the Hierarchical File System (HFS), routines that interface to the SCSI (Small Computer System Interface) port, and updates to most of the 64K ROM routines. Volume IV covers all of these changes.

Volume V was released after the introduction of the Mac SE and the Mac II. The Mac II and the SE have 256K ROMs and support features such as pop-up, hierarchical, and scrolling menus; a sophisticated sound manager; new text-edit routines; and more. Perhaps the biggest change was the addition of color support to the Mac II series.

Finally, Volume VI explains the enhancements provided by the long-awaited System 7: InterApplication Communication (IAC), virtual memory, AppleEvents, a redesigned Finder, new printing, database routines, and more!

## The Typical *Inside Macintosh* Chapter

One of the best features of the *Inside Macintosh* volumes is their consistency. Each chapter starts with a table of contents, followed by the "About This Chapter" section, which gives you an overview of what the chapter covers and what you should already be familiar with before you continue.

The next section, or sections, gives an overview of the chapter's technical premise; for example, "About the Event Manager," or "About the Window Manager." The fundamental concepts are explained in great detail. At first, you may be overwhelmed by the wealth of detail, but after a few readings (and a little experimentation), you'll warm to the concept.

Next, the chapter's data structures, constants, and essential variables are detailed. These are presented in Pascal and/or assembly language. Then come the chapter's Toolbox routines. Each routine's calling sequence is presented in Pascal, along with a detailed explanation of the uses of the routine. This section includes notes and warnings, as appropriate.

Some chapters follow the Toolbox routines section with a few additional sections. Among these extras are a description of the resources pertinent to that chapter and, perhaps, a description of extensions available to the advanced programmer.

Finally, there's a chapter summary, with unadorned lists of constants, data types, routines, and variables.

## Appendixes and Special Sections

The preface from *Inside Macintosh*, Volume VI is a must read. It contains an overview of each *Inside Macintosh* volume and a road map that leads you through the remainder of Volume VI, chapter by chapter. The road map lists related chapters from earlier *Inside Macintosh* volumes, making clear which volume has the final say on which topic.

Volume III contains three chapters, some appendixes, a glossary, and an index. Chapter 1 discusses the Finder (with an emphasis on Finder-related resources). Chapter 2 discusses the pre-Mac Plus hardware. Chapter 3 is a compendium of all the summary sections from Volumes I and II. Appendix A is a handy, if occasionally inaccurate, table of result codes from the functions defined in Volumes I and II. The rest of the appendixes in Volume III have been superseded by the appendixes in the second edition of the *Inside Macintosh X-Ref*.

The second edition of the *Inside Macintosh X-Ref* starts with a general index covering all six *Inside Mac* volumes, plus *Inside the Macintosh Communications Toolbox*; *Designing Cards and Drivers for the Macintosh Family*; *Guide to the Macintosh Family Hardware*; *Programmer's Introduction to the Macintosh Family*; and *Technical Introduction to the Macintosh Family*. The general index is followed by an index of constants and field names. Appendix A of the *X-Ref* lists every Toolbox routine that may move or purge memory.

Appendix B of the *Inside Macintosh X-Ref*, second edition, consists of four lists. The first is a list of Toolbox routines presented alphabetically by name, with each name followed by the routine's **trap address**, which is the 4-byte instruction the compiler generates to call the routine. The second is a list of the trap addresses, in order, with each trap address followed by the routine name. The third lists traps that have more than one routine associated with them, and the fourth lists these same traps ordered by routine name. This information is extremely useful if you ever have to look at code in hexadecimal format, a likely event if you use TMON or MacsBug, two Mac debuggers.

Appendix C of the *Inside Macintosh X-Ref* lists most of the operating system global variables, with their memory location and a brief description. Appendix D contains a table showing the standard roman character set, listing each character's hex and decimal value, along



with the character's PostScript name. Finally, Appendix D is followed by a glossary of terms presented in Volumes I through VI.

---

## Apple Technical References

---

In the first few years of the Mac era, *Inside Macintosh* was the only definitive reference on the Macintosh. Over the years, however, Apple has published many additional reference texts for the Macintosh, including *Inside the Macintosh Communications Toolbox*; *Designing Cards and Drivers for the Macintosh Family*; *Guide to the Macintosh Family Hardware*; *Programmer's Introduction to the Macintosh Family*; and *Technical Introduction to the Macintosh Family*. These books are all part of Addison-Wesley's Apple Technical Library. Another excellent source of technical information is the *Macintosh Technical Notes*.

### Macintosh Technical Notes

*Macintosh Technical Notes* are published on a regular basis by Apple and distributed to Apple Partners and Associates (we'll discuss both in a moment) free of charge. The *Tech Notes* are a necessity for serious Mac developers. They contain technical information that was not yet available when the latest volume of *Inside Macintosh* went to press. For example, *Tech Note #184* described the Notification Manager (used in Chapter 6) well before Volume VI of *Inside Macintosh* hit the bookstore shelves. Without this *Tech Note*, developers wouldn't even know the Notification Manager existed, let alone know how to use it.



A timely way to receive *Tech Notes* if you are not a Partner or Associate is to order them through APDA, the Apple Programmers and Developers Association. APDA sells almost everything on the Mac's technical side. They sell the *Tech Notes* in both hard copy and disk formats. Call APDA at (800) 282-2732, and ask them to send you a catalog. You'll be glad you did.

If you don't want to order the tech notes from APDA, you can still get *Tech Notes* by downloading them from Mac-based bulletin boards around the country.

Another important technical guide from APDA benefits developers using Apple events. If you plan on going beyond the required Apple

events, buy the *Apple Event Registry*, which contains detailed information on all the currently defined Apple Events.

## Other Books

There are several excellent books on Macintosh programming. One classic title is Scott Knaster's *How to Write Macintosh Software*, now in its third edition. This book is a little too advanced for the beginner, but it's worth the struggle to get through it. If you plan on writing a lot of Mac code, read this book.

Addison-Wesley has published a number of excellent Macintosh programming books under the banner of the *Macintosh Inside-Out* series. These books include such titles as *Programming for System 7* and *ResEdit Complete*.

Finally, you might want to try the *Macintosh C Programming Primer*, Volume II, by Dave Mark. Object programming, Color QuickDraw, INITs, cdevs, and other interesting Toolbox routines are examined, with lots of examples and code walkthroughs.

---

## Apple's Developer Programs

---

Apple's Partner and Associate programs were designed to give full-time Macintosh developers additional technical support from Apple. The Apple Partners program offers complete Apple technical documentation, system software updates, access to training classes, and discounts on Apple hardware and software. Partners also get a year's subscription to AppleLink, Apple's electronic communication network, and access to Macintosh Developer Technical Support (see next section). If you have a CD-ROM drive, you may want to take advantage of the Developer CD Series, which is a set of CD-ROMs shipped to developers every few months, containing sample code, utility programs, and an electronic version of *Inside Macintosh*!

The only disadvantage of being a developer is parting with the check you include with your Apple Partners application (currently \$600).

You don't have to be a Fortune 500 company to qualify as an Apple Partner, but Apple is looking specifically for developers of Apple hardware and software who intend to resell their products. If you are interested in developing software but don't have an immediate plan to market it, you might consider the Apple Associates Program, another support program from Apple.

The Apple Associates Program is aimed at educators, in-house developers, and shareware programmers. It provides a basic level of support, including AppleLink (one month prepaid), system software

upgrades, *Tech Notes*, and access to other technical information. The Associates program currently costs \$350 a year.

If you plan on writing a product for the Mac, the information you receive in either program is invaluable. Call the Developer Programs Hotline at (408) 974-4897 and ask them to send you an application.

If you are a developer, there's nothing more satisfying than talking to people who have solved, or at least are aware of, the technical problems you encounter in writing programs. At Apple, these people come from Macintosh Developer Technical Support, or MacDTS.

---

## Macintosh Developer Technical Support and AppleLink

---

Macintosh Developer Technical Support is a team of talented Mac software engineers dedicated to helping developers with their technical problems. To work with MacDTS, you must first join the Apple Partners program. Once you're an Apple Partner, you can send your technical questions to Developer Technical Support through AppleLink (the AppleLink address is MacDTS).

AppleLink offers access to Apple product, pricing, support programs, and policy information. If you write to Developer Technical Support (their AppleLink address is MacDTS), they will make every possible effort to answer your question promptly.

Both Apple Partners and Apple Associates receive subscriptions to AppleLink: Apple Partners receive a full year's subscription with the minimum monthly fees prepaid; Apple Associates receive one month of the minimum monthly fee prepaid.

Besides access to MacDTS, AppleLink gives you access to a lot of other services. You can download the new system utilities or look at the Help Wanted ads posted on the bulletin board. You can send beta versions of your products to your evangelist at Apple or to other developers. AppleLink makes you a part of the developer community.

## CompuServe and America Online

If you don't have access to AppleLink, there are several other ways to hook up with the Macintosh programming community. For starters, there's the CompuServe Information Service. CompuServe has one of the strongest Macintosh followings of any on-line service. By logging on to CompuServe, you have access to a vast quantity of Macintosh technical information, from the latest System files to a guru to help you with your latest programming project. When logging on to CompuServe, type GO MACDEV, then stop by the Learn Programming area (Section 11) and say hello.

Another electronic service is America Online. America Online offers access to most of the same services, information, and people as CompuServe, and at a lower price. It is definitely worth checking out.

To use either of these services, you'll need a modem and some special software. For CompuServe access, any telecommunications package will work, but you'll save time and money if you pick up a copy of either the CompuServe Information Manager or CIS Navigator. Both of these programs help you organize your on-line time.

To access America Online, you'll need the America Online program, available from Quantum Computer Services, in Vienna, Virginia. Occasionally, you'll spot CompuServe and America Online starter kits in your favorite technical bookstore.

---

## **Software Development Tools**

---

All the applications presented in this book were written in C, using the THINK C development environment from Symantec. The advantages of THINK C lie primarily in its ease of use and debugging facilities. Symantec also makes a powerful, yet friendly, Pascal development environment called THINK Pascal.

Both THINK environments are basically nonextensible. This means you can't create shell scripts to back up your files automatically, or rebuild an older version of your project. You also can't create custom menu items that automate your development process. THINK environments handle most of the development cycle so thoroughly that you may not miss these features. If you do, you may want to take a look at the Macintosh Programmer's Workshop (MPW) from Apple.

### **MPW from Apple**

MPW is an extremely powerful development environment that is totally extensible—so extensible, in fact, that several third parties have produced compilers that run under MPW. MPW is like a Mac-based UNIX shell. You can write shell scripts, tie them to your own menus, and create tools that have total access to the Toolbox yet run inside the Toolbox environment with access to all of your data. The catch is that MPW is more complex than THINK C and, therefore, more difficult to master. MPW is also not cheap, typically costing more than three times as much as THINK C or Pascal.

Both MPW and THINK have many followers and are supported by MacDTS. Whichever way you go, you'll be in good company.

## **Debugging with THINK C, TMON, The Debugger, and MacsBug**

Debugging on any computer can be a tedious and frustrating experience. Luckily, there are some excellent tools you can use to fix up your code.

Normally, the THINK C debugger will prove more than adequate to track down most bugs. If you need lower-level support, however, there are several other options available.

MacsBug is an object-level debugger developed by Motorola for the 68000 family of processors. For a long time, it was the only debugger available for the Mac.

If you need a little more horsepower than MacsBug offers, consider either TMON from ICOM Simulations, or Jasik Designs' excellent combination, MacNosy and the Debugger. Both of these products are professional debugging tools. Instead of running as a normal application, both of these products take over the processor when they run. They preserve your program's run-time environment by not calling any of the Mac Toolbox routines (which might alter the state of your program). Instead, each implements its own window and menu handlers. Although each is somewhat difficult to learn, they're worth it. When you run into an exasperatingly unexplainable bug, pop into TMON or the Debugger and step through your program. You can set breakpoints, disassemble your executable image, and even make changes to your program and data. For debugging drivers, INITs, and DAs, these two programs can't be beat.

## **Resourcerer: An Alternative to ResEdit**

Although ResEdit is powerful, it has its limitations. A resource-editing alternative is Resourcerer, from Mathemaesthetics. Resourcerer offers most of the same features as ResEdit, with lots of extras. Here's a sampling.

When you open a CODE resource, Resourcerer will disassemble it for you. The CODE editor includes symbolic names, and allows you to edit and patch CODE resources in a variety of formats. Unlike ResEdit, Resourcerer knows how to edit all balloon help resources and lets you try out your help balloons in both dialogs and menus. You can edit your data fork as a resource, write your own resource editing templates, and compare two resource files.

To find out more about Resourcerer, call Mathemaesthetics at (617) 738-8803.

## CMaster: Customizing Your THINK C Environment

One final product we'd like to mention is CMaster, a THINK C customizer. Once you've installed CMaster, you'll find yourself with several new features when you run THINK C. The most noticeable change is evidenced by the menus and icons that appear in each of your source code windows. One menu lists each of the functions in the file. Select a function from the menu and CMaster jumps to that section of code.

When you click and drag in the thumb of a source code window, CMaster scrolls your code as you move the thumb. (This may not seem like much, but it's really cool to watch.) There's a find icon that searches up or down in the file, depending on whether you click on the top or bottom half of the icon.

CMaster has a boatload of features. To find out more, call Jersey Scientific at (212) 736-0406.

---

## Source Code Bounty

---

Tired of burning the midnight oil struggling with some weighty Toolbox concept? Put down those pruning shears and pay close attention. Chances are, someone, somewhere has already done what you're trying to do.

America Online and CompuServe both feature a treasure trove of useful source code examples. Get the current issue of MacTutor. Now go to the bookstore and buy all of the back issues, marketed under the title *The Best of MacTutor*. There are several volumes, all of them useful.

Check out your local Macintosh user group. Most user groups have a special interest group dedicated to programming. Some, like the Berkeley Macintosh user group, offer source code disks through mail order.

Finally, check out the selection offered by Intelligence At Large. For starters, they feature a complete line of *Mac Primer* source code disks. They also offer the *Mac Programming 101* series, source code for aspiring Mac programmers that shows the basics for text editing, picture displaying, sound making, Apple events, and more. To place an order, or for more information, call Intelligence At Large at (215) 387-6002.



## To Boldly Go

---

The Macintosh world is accelerating.

New names, functions, and features appear with startling speed: Taligent, QuickTime, voice recognition, and synthesized speech are all new parts of an exciting Macintosh revolution. The Macintosh operating system has hit its stride with System 7. As a testbed for innovation and excitement, the Macintosh stands alone.

A new generation of Macintosh developers is coming on board.

And you ain't seen nothing yet!

# Appendix A

---

---

## Glossary

**A5 world:** An area of memory in the application's partition that contains QuickDraw™ global variables and the application's global variables, parameters, and jump table—all of which are accessed through the A5 register.

**access path:** A description of the route that the File Manager follows to access a file; created when a file is opened.

**action procedure:** A procedure, used by the Control Manager function TrackControl, that defines an action to be performed repeatedly for as long as the mouse button is held down.

**activate event:** An event generated by the Window Manager when a window changes from active to inactive or vice versa.

**active application:** The application currently interacting with the user. Its icon appears on the right side of the menu bar. See also **current process, foreground process.**

**active control:** A control that responds to the user's actions with the mouse.

**active field:** The target of keyboard input in a dialog box.

**active window:** The front-most window on the desktop.

**address:** A number used to identify a location in the computer's address space. Some locations are allocated to memory, others to I/O devices.

**address descriptor record:** A descriptor record that contains the address of the target or source of an Apple event.



**AEIMP:** See **Apple Event Interprocess Messaging Protocol**.

**AE record:** A record of data type **AERecord** that contains a list of parameters for an Apple event. See also **Apple event parameter**.

**alert:** A warning or report of an error in the form of an alert box, a sound from the Macintosh® speaker, or both.

**alert box:** A box that appears on the screen to give a warning or report an error during a Macintosh application.

**alert template:** A resource that contains information from which the Dialog Manager can create an alert.

**alert window:** The window in which an alert box is displayed.

**alias:** An object on the desktop that represents another file, directory, or volume. An alias looks like the icon of its **target**, but its name is displayed in a different font style. The style depends on the **system script**; for Roman and most other scripts, alias names are displayed in italic. Aliases give users more flexibility in organizing their desktops and offer a convenient way to store local copies of large or dynamic files that reside on file servers.

**alias file:** A file that contains a record that points to another file, directory, or volume. An alias file is displayed by the Finder™ as an **alias**.

**alias record:** A data structure created by the Alias Manager to identify a file, directory, or volume.

**alternate rectangle:** A rectangle used by the Help Manager (under some circumstances) for transposing a help balloon's **tip** when trying to fit the balloon on screen. For all help resources except the 'hdlg' resource, the Help Manager moves the tip to different sides of the hot rectangle. For 'hdlg' resources, however, the Help Manager allows you to specify alternate rectangles for transposing balloon tips. You can also specify alternate rectangles when you use the **HMShowBalloon** and **HMShowMenuBalloon** functions.

**Apple event:** A high-level event that adheres to the **Apple Event Interprocess Messaging Protocol**. An Apple event consists of attributes (including the event class and event ID, which identify the event and its task) and, usually, parameters (which contain data used by the target application of the event). See also **Apple event attribute**, **Apple event parameter**.

**Apple event attribute:** A **keyword-specified descriptor record** that identifies the event class, event ID, target application, or some other characteristic of an Apple event. Taken together, the attributes of an Apple event identify the event and denote the task to be performed on the data specified in the Apple event's parameters. Compared to parameters (which contain data used only by the target application of the Apple event), attributes contain information that can be used by both the Apple Event Manager and the target application. See also **Apple event parameter**.

**Apple event dispatch table:** A table that the Apple Event Manager uses to map Apple events to application-defined functions called **Apple event handlers**.

**Apple event handler:** An application-defined function that extracts pertinent data from an Apple event, performs the action requested by the Apple event, and returns a result.

**Apple Event Interprocess Messaging Protocol (AEIMP):** A standard defined by Apple Computer, Inc., for communication and data sharing among applications. High-level events that adhere to this protocol are called **Apple events**.

**Apple event parameter:** A keyword-specified descriptor record that contains data that the target application of an Apple event must use. Compared to attributes (which contain information that can be used by both the Apple Event Manager and the target application), parameters contain data used only by the target application of the Apple event. See also **Apple event attribute**, **direct parameter**, **optional parameter**, **required parameter**.

**Apple event record:** A record of data type `AppleEvent` that contains a list of **keyword-specified descriptor records**. These descriptor records describe—at least—the attributes necessary for an Apple event; they may also describe parameters for the Apple event. Apple Event Manager functions are used to add parameters to an Apple event record.

**Apple Menu Items folder:** A directory located in the System Folder for storing desk accessories, applications, folders, and aliases that the user wants to display in and access from the Apple menu.

**application font:** The font your application uses unless you specify otherwise—Geneva, by default.

**application heap:** An area of memory in the application **partition** that contains the application's 'CODE' segment 1, data structures, resources, and other code segments as needed.

**application heap limit:** The boundary between the space available for the application heap and the space available for the stack.

**application heap zone:** The heap zone initially provided by the Memory Manager for use by the application program and the Toolbox; initially equivalent to the application heap, but may be subdivided into two or more independent heap zones.

**application window:** A window created as the result of something done by the application, either directly or indirectly (as through the Dialog Manager).

**auto-key event:** An event generated repeatedly when the user presses and holds down a character key on the keyboard or keypad.

**auto-key rate:** The rate at which a character key repeats after it's begun to do so.

- auto-key threshold:** The length of time a character key must be held down before it begins to repeat.
- background activity:** A program or process that runs while the user is engaged with another application.
- background process:** A process that isn't currently interacting with the user. Compare **foreground process**.
- bit image:** A collection of bits in memory that have a rectilinear representation. The screen is a visible bit image.
- bitmap:** A set of bits that represents the positions and states of a corresponding set of items, such as pixels.
- bitmapped font:** A collection of bitmapped glyphs in a particular typeface, size, and style.
- bundle:** A resource that maps local IDs of resources to their actual resource IDs; used to provide mappings for file references and icon lists needed by the Finder.
- bundle bit:** A flag in a file's FInfo record that informs the Finder that a 'BNDL' resource exists for the file. A file's FInfo record is stored in a volume's **catalog**. The Finder uses the information in the 'BNDL' resource to associate icons with the file.
- button:** A standard Macintosh control that causes some immediate or continuous action when clicked or pressed with the mouse. See also **radio button**.
- caret:** A generic term meaning a symbol that indicates where something should be inserted in text. The specific symbol used is a vertical bar (|).
- caret-blink time:** The interval between blinks of the caret that marks an insertion point.
- cdev:** A resource file containing device information, used by the Control Panel (in system software prior to version 7.0) or stored in the Control Panels folder inside the System Folder (in version 7.0). See also **control panel**.
- cell:** The basic component of a list from a structural point of view; a cell is a box in which a list element is displayed.
- character code:** A hexadecimal number from \$00 through \$FF that represents the character that a key or key combination stands for.
- character key:** A key that generates a keyboard event when pressed; any key except Shift, Caps Lock, Command, Option, Control, or Esc.
- check box:** A standard Macintosh control that displays a setting, either checked (on) or unchecked (off). Clicking inside a check box reverses its setting.
- Chooser:** A desk accessory that provides a standard interface for device drivers to solicit and accept specific choices from the user.
- clipping:** Limiting drawing to within the bounds of a particular area.
- clipping region:** Same as **clipRgn**.
- clipRgn:** The region to which an application limits drawing in a grafPort.

- closed file:** A file without an access path. Closed files cannot be read from or written to.
- content region:** The area of a window that the application draws in.
- control:** An object in a window on the Macintosh screen with which the user, using the mouse, can cause instant action with visible results or change settings to modify a future action.
- control definition ID:** A number passed to control-creation routines to indicate the type of control. It consists of the control definition function's resource ID and a variation code.
- control list:** A list of all the controls associated with a given window.
- Control Manager:** The part of the Toolbox that provides routines for creating and manipulating controls (such as buttons, check boxes, and scroll bars).
- control panel:** A dialog box defined by a file of file type 'cdev'. A control panel allows the user to set or control some feature of hardware or software, such as the volume of the speaker or the number of colors displayed on screen.
- control panel file:** A file of file type 'cdev'. See also **control panel**.
- Control Panels folder:** A directory located in the System Folder for storing control panels, which allow users to modify the work environment of their Macintosh computer.
- control record:** The internal representation of a control, where the Control Manager stores all the information it needs for its operations on that control.
- coordinate plane:** A two-dimensional grid. In QuickDraw, the grid coordinates are integers ranging from -32767 to 32767, and all grid lines are infinitely thin.
- core Apple event:** An Apple event that nearly all applications can use to communicate. The suite of core Apple events is described in the *Apple Event Registry*; Apple Computer, Inc., recommends that all applications support the core Apple events.
- current process:** The process that is currently executing and whose **A5 world** is valid; this process can be in the background or foreground.
- current resource file:** The last resource file opened, unless you specify otherwise with a Resource Manager routine.
- cursor:** A 16-by-16 bit image that appears on the screen and is controlled by the mouse; called the "pointer" in Macintosh user manuals.
- custom Apple event:** An Apple event defined by you for use by your own applications. You should register all of your custom Apple events with Macintosh Developer Technical Support. You can choose to publish your Apple events in the *Apple Event Registry* so that other applications can share them, or you may choose to keep them unpublished for exclusive use by your own applications.

- customized icon:** An icon created by the user or by an application and stored with a resource ID of -16455 in the resource fork of a file. A file with a customized icon has the `hasCustomIcon` bit set in its Finder flags field.
- data fork:** The part of a file that contains data accessed via the File Manager.
- default button:** In an alert box or modal dialog box, the button whose effect occurs if the user presses Return or Enter. In an alert box, it's boldly outlined; in a modal dialog box, it's boldly outlined or it's the OK button.
- descriptor record:** A record of data type `AEDesc` that consists of a handle to data and a **descriptor type** that identifies the type of the data referred to by the handle. Descriptor records are the fundamental structures from which Apple events are constructed.
- descriptor type:** An identifier for the type of data referred to by the handle in a **descriptor record**.
- desk accessory:** A "mini-application," implemented as a device driver, that can be run at the same time as a Macintosh application.
- Desk Manager:** The part of the Toolbox that supports the use of desk accessories from an application.
- desk scrap:** The place where data is stored when it's cut (or copied) and pasted among applications and desk accessories.
- desktop:** The screen as a surface for doing work on the Macintosh.
- dial:** A control with a moving indicator that displays a quantitative setting or value. Depending on the type of dial, the user may be able to change the setting by dragging the indicator with the mouse.
- dialog:** Same as **dialog box**.
- dialog box:** A box that a Macintosh application displays to request information it needs to complete a command, or to report that it's waiting for a process to complete.
- dialog hook function:** A function supplied by your application for handling item hits in a dialog box.
- Dialog Manager:** The part of the Toolbox that provides routines for implementing dialog boxes and alert boxes.
- dialog record:** The internal representation of a dialog box, where the Dialog Manager stores all the information it needs for its operations on that dialog box.
- dialog template:** A resource that contains information from which the Dialog Manager can create a dialog box.
- dialog window:** The window in which a dialog box is displayed.
- dimmed:** Drawn in gray rather than black.
- direct parameter:** The parameter in an Apple event that contains the data to be used by the server application. For example, a list of documents to be opened is specified in the direct parameter of the Open Documents event. See also **Apple event parameter**.

- directory:** A subdivision of a volume, available in the hierarchical file system (HFS). A directory can contain files and other directories.
- directory ID:** A unique number assigned to a directory, which the File Manager uses to distinguish it from other directories on the volume. (It's functionally equivalent to the file number assigned to a file; in fact, both directory IDs and file numbers are assigned from the same set of numbers.)
- disk-inserted event:** An event generated when the user inserts a disk in a disk drive or takes any other action that requires a volume to be mounted.
- display rectangle:** A rectangle that determines where an item is displayed within a dialog or alert box.
- empty handle:** A handle that points to a NIL master pointer, signifying that the underlying relocatable block has been purged.
- event:** A notification to an application of some occurrence that the application may want to respond to.
- event class:** An attribute that identifies a group of related Apple events. The event class appears in the message field of the Apple event's event record. In conjunction with the event ID attribute, the event class specifies what action an Apple event performs. See also **Apple event attribute**.
- event code:** An integer representing a particular type of event.
- event ID:** An attribute that identifies a particular Apple event within a group of related Apple events. The event class appears in the where field of the Apple event's event record. In conjunction with the event class attribute, the event ID specifies what action an Apple event performs. See also **Apple event attribute**.
- event mask:** A parameter passed to an Event Manager routine to specify which types of events the routine should apply to.
- event message:** A field of an event record containing information specific to the particular type of event.
- event queue:** The Operating System Event Manager's list of pending events.
- event record:** The internal representation of an event, through which your program learns all pertinent information about that event.
- file:** A named, ordered sequence of bytes; a principal means by which data is stored and transmitted on the Macintosh.
- file ID:** An unchanging number assigned by the File Manager to identify a file on a volume. When it establishes a file ID, the File Manager records the filename and parent directory ID of the file. The Alias Manager records a file's ID to help identify it if it is moved or renamed.
- filename:** A sequence of up to 255 printing characters, excluding colons (:), that identifies a file.

- file number:** A unique number assigned to a file, which the File Manager uses to distinguish it from other files on the volume. A file number specifies the file's entry in a file directory.
- file reference:** A resource that provides the Finder with file and icon information about an application.
- file system specification (FSSpec) record:** A record that identifies a stored file or directory by volume reference number, parent directory ID, and name. The file system specification record is the file-identification convention adopted by system software version 7.0.
- file type:** A four-character sequence, specified when a file is created, that identifies the type of file.
- Finder flags:** Bits in the fdFlags field of a file's FInfo record; these bits are used by the Finder and by applications for setting and reading certain information about the file, such as whether the file is an alias file, whether it has a bundle resource, whether it is a stationery pad, and whether it has a customized icon.
- Finder information:** Information that the Finder provides to an application upon starting it up, telling it which documents to open or print.
- font:** (1) For bitmapped fonts, a complete set of characters in one typeface, size, and style. (2) For outline fonts, a complete set of characters in one typeface and style. See also **bitmapped font**, **outline font**.
- Font Manager:** The part of the Toolbox that supports the use of various fonts for QuickDraw when it draws text.
- font number:** The number by which you identify a font to QuickDraw or the Font Manager.
- font scaling:** The process of changing a glyph from one size or shape to another. The Font Manager can scale bitmapped and outline fonts in three ways: changing a glyph's point size on the same display device, modifying the glyph but keeping the point size constant when using a different display device, and altering the shape of a glyph.
- font size:** The size of the glyphs in a font in points, measured from the base line of one line of text to the base line of the next line of single-spaced text.
- font style:** Stylistic variations in the appearance of a typeface, such as italic, bold, and underline.
- foreground process:** The process currently interacting with the user; it appears to the user as the active application. The foreground process displays its menu bar, and its windows are in front of the windows of all other applications. Compare **background process**.
- fork:** One of the two parts of a file; see **data fork** and **resource fork**.
- functional-area Apple event:** An Apple event supported by applications with related features—for example, an Apple event related to text manipulation for word-processing applications, or an Apple event related to graphics manipulation for drawing applications. Func-

tional-area Apple events are defined by Apple Computer, Inc., in consultation with interested developers, and they are published in the *Apple Event Registry*.

**global coordinate system:** The coordinate system based on the top left corner of the bit image being at (0,0).

**go-away region:** A region in a window frame. Clicking inside this region of the active window makes the window close or disappear.

**grafPort:** A complete drawing environment, including such elements as a bitmap, a subset of it in which to draw, a font, patterns for drawing and erasing, and other pen characteristics.

**graphics environment:** The combination of one or more grafPorts, which contain information about windows, and graphics device records, which contain information about display devices attached to a computer system.

**gray region:** The region that defines the desktop, or the display area of all active devices, excluding the menu bar on the main screen and the rounded corners on the outermost screens. It is the area in which windows can be moved. See also **main screen**.

**GrayRgn:** The global variable that in the multiple screen desktop describes and defines the desktop, the area on which windows can be dragged.

**grow region:** A window region, usually within the content region, where dragging changes the size of an active window.

**handle:** A pointer to a master pointer, which designates a relocatable block in the heap by double indirection.

**hierarchical menu:** A menu that includes, among its various menu choices, the ability to display a submenu. In most cases the submenu appears to the right of the menu item used to select it, and is marked with a filled triangle indicator.

**high-level event:** An event that your application can send to another application to transmit some information, to receive from it some information, or to have it perform some action.

**icon:** An image that graphically represents an object, concept, or message.

**icon family:** The set of icons that represent an object, such as an application or document, on the desktop. An entire icon family consists of large (32-by-32 pixel) and small (16-by-16 pixel) icons, each with a mask, and each available in three different versions of color: black and white, 4 bits of color data per pixel, and 8 bits of color data per pixel.

**icon list:** A resource consisting of a list of icons.

**idle state:** A state in which the Macintosh Portable computer slows from its normal 16 MHz clock speed to a 1 MHz clock speed. The Power Manager puts the Macintosh Portable in the idle state when the system has been inactive for 15 seconds.



- inactive control:** A control that won't respond to the user's actions with the mouse. An inactive control is highlighted in some special way, such as dimmed.
- inactive window:** Any window that isn't the frontmost window on the desktop.
- insertion point:** An empty selection range; the character position where text will be inserted (usually marked with a blinking caret).
- interapplication communication (IAC):** A collection of features, provided by the Edition Manager, Apple Event Manager, Event Manager, and PPC Toolbox, that help applications work together. You can use these managers to share data, send and receive events, or exchange low-level message blocks.
- item:** In dialog and alert boxes, a control, icon, picture, or piece of text, each displayed inside its own display rectangle. See also **menu item**.
- item list:** A list of information about all the items in a dialog or alert box.
- item number:** The index, starting from 1, of an item in an item list.
- keyboard equivalent:** The combination of a modifier key and another key, used to invoke a menu item from the keyboard.
- keyboard event:** An event generated when the user presses, releases, or holds down a character key on the keyboard or keypad; any key-down, key-up, or auto-key event.
- key code:** An integer representing a key on the keyboard or keypad, without reference to the character that the key stands for.
- key-down event:** An event generated when the user presses a character key on the keyboard or keypad.
- key-up event:** An event generated when the user releases a character key on the keyboard or keypad.
- keyword:** A four-character code used to uniquely identify the **descriptor record** for either an attribute or a parameter in an Apple event. In Apple Event Manager functions, constants are typically used to represent the four-character codes.
- keyword-specified descriptor record:** A record of data type `AEKeyDesc` that consists of a **keyword** and a **descriptor record**. Keyword-specified descriptor records are used to describe the attributes and parameters of an Apple event.
- localization:** The process of adapting software to a particular region, language, and culture. Script and language adaptations are necessary but not sufficient for this process. Localization also includes date and time formats, keyboard resources, and fonts.
- localized system software:** Macintosh system software that has been adapted to a particular region, language, and culture. Japanese system software is the combination of the U.S. system software (including the Roman Script System, the Macintosh Operating System, the Toolbox, and so forth) and the Japanese Script System, all of which are adapted for use in Japan. The French and Turkish versions of the Macintosh system software are examples of localized variations of the

- lock:** To temporarily prevent a relocatable block from being moved during heap compaction.
- lock bit:** A bit in the master pointer to a relocatable block that indicates whether the block is currently locked.
- locked file:** A file whose data cannot be changed.
- main event loop:** In a standard Macintosh application program, a loop that repeatedly calls the Toolbox Event Manager to get events and then responds to them as appropriate.
- main screen:** The screen on which the menu bar appears. QuickDraw uses it to determine global coordinates.
- mark:** A marker used by the File Manager to keep track of where it is during a read or write operation. It is the position of the next byte in a file that will be read or written.
- master pointer:** A single pointer to a relocatable block, maintained by the Memory Manager and updated whenever the block is moved, purged, or reallocated. All handles to a relocatable block refer to it by double indirection through the master pointer.
- Memory Manager:** The part of the Operating System that dynamically allocates and releases memory space in the heap.
- menu:** A list of menu items that appears when the user points to a menu title in the menu bar and presses the mouse button. Dragging through the menu and releasing over an enabled menu item chooses that item.
- menu bar:** The horizontal strip at the top of the Macintosh screen that contains the menu titles of all menus in the menu list.
- menu definition procedure:** A procedure called by the Menu Manager when it needs to perform type-dependent operations on a particular type of menu, such as drawing the menu.
- menu entry:** An entry in a menu color table that defines color values for the menu's title, bar, and items.
- menu ID:** A number in the menu record that identifies the menu.
- menu item:** A choice in a menu, usually a command to the current application.
- menu item number:** The index, starting from 1, of a menu item in a menu.
- menu list:** A list containing menu handles for all menus in the menu bar, along with information on the position of each menu.
- Menu Manager:** The part of the Toolbox that deals with setting up menus and letting the user choose from them.
- menu record:** The internal representation of a menu, where the Menu Manager stores all the information it needs for its operations on that menu.
- menu title:** A word, phrase, or icon in the menu bar that designates one menu.
- minimum partition size:** The actual partition size limit below which your application cannot run.

- minor switch:** The Process Manager switches the **context** of a process to give time to a **background process** without bringing the background process to the front.
- modal dialog box:** A dialog box that requires the user to respond before doing any other work on the desktop.
- modal-dialog filter function:** A function supplied by your application for handling events received from the Event Manager while a dialog box is displayed.
- modeless dialog box:** A dialog box that allows the user to work elsewhere on the desktop before responding.
- modifier key:** A key (Shift, Caps Lock, Option, Command, or Control) that generates no keyboard events of its own, but changes the meaning of other keys or mouse actions.
- mouse-down event:** An event generated when the user presses the mouse button.
- mouse-up event:** An event generated when the user releases the mouse button.
- notification queue:** The Notification Manager's list of pending notification requests.
- notification record:** The internal representation of a notification request, through which you specify how a **notification** is to occur.
- notification request:** A request to the Notification Manager to create a notification.
- notification response procedure:** A procedure that the Notification Manager can execute as the final step in a notification.
- null event:** An event reported when there are no other events to report.
- Open Application event:** An Apple event that asks an application to perform the tasks—such as displaying untitled windows—associated with opening itself; one of the four required Apple events.
- Open Documents event:** An Apple event that requests an application to open one or more documents specified in a list; one of the four required Apple events.
- open file:** A file with an access path. Open files can be read from and written to.
- Operating System:** The lowest-level software in the Macintosh. It does basic tasks such as I/O, memory management, and interrupt handling.
- Operating System Event Manager:** The part of the Operating System that reports hardware-related events such as mouse-button presses and keystrokes.
- optional parameter:** A supplemental parameter in an Apple event used to specify data that the server application should use in addition to the data specified in the **direct parameter**. Optional parameters are listed in the attribute identified by the keyOptionalKeywordAttr keyword. Applications use this attribute to specify or determine

whether data exists in the form of optional parameters. Optional parameters need not be included in an Apple event; default values for optional parameters are part of the event definition. It is the responsibility of the server application that handles the event to supply values if optional parameters are omitted. See also **Apple event attribute**, **Apple event parameter**.

**outline font:** A collection of outline glyphs in a particular typeface and style with no size restriction. The Font Manager can generate thousands of point sizes from the same TrueType font.

**picture:** A saved sequence of QuickDraw drawing commands (and, optionally, picture comments) that you can play back later with a single procedure call; also, the image resulting from these commands.

**pixel:** Short for *picture element*; the smallest dot you can draw on the screen.

**pixel map:** A data structure that contains information about an image's pixels, including their arrangement for display, the number of bits per pixel (its depth), and the colors the image requires.

**point:** (1) A unit of measurement for type. Twelve points equal 1 pica, and 6 picas equal 1 inch; thus, 1 point equals approximately 1/72 inch. (2) The intersection of a horizontal grid line and a vertical grid line on the coordinate plane, defined by a horizontal and a vertical coordinate.

**polygon:** A sequence of connected lines, defined by QuickDraw line-drawing commands.

**pop-up menu:** A menu not located in the menu bar, which appears when the user presses the mouse button in a particular place.

**port:** (1) A portal through which an open application can exchange information with another open application using the PPC Toolbox. A port is designated by a **port name** and a **location name**. An application can open as many ports as it requires so long as each port name is unique within a particular computer. (2) A connection between the CPU and main memory or a device (such as a terminal) for transferring data. (3) A socket on the back panel of a computer where you plug in a cable for connection to a network or a peripheral device.

**portBits:** The bitmap of a grafPort.

**portRect:** A rectangle, defined as part of a grafPort, that encloses a subset of the bitmap for use by the grafPort.

**post:** To place an event in the event queue for later processing.

**Print Documents event:** An Apple event that requests that an application print a list of documents; one of the four required Apple events.

**printing grafPort:** A special grafPort customized for printing instead of drawing on the screen.

- Printing Manager:** The routines and data types that enable applications to communicate with the Printer Driver to print on any variety of printer via the same interface.
- print record:** A record containing all the information needed by the Printing Manager to perform a particular printing job.
- process:** An open application or, in some cases, an open desk accessory. (Only desk accessories that are not opened in the context of another application are considered processes.)
- programming language:** A set of symbols and associated rules or conventions for writing programs. For example, BASIC, Logo, and Pascal are programming languages.
- purge:** To remove a relocatable block from the heap, leaving its master pointer allocated but set to NIL.
- purgeable block:** A relocatable block that can be purged from the heap.
- purge bit:** A bit in the master pointer to a relocatable block that indicates whether the block is currently purgeable.
- purge warning procedure:** A procedure associated with a particular heap zone that's called whenever a block is purged from that zone.
- queue:** A list of identically structured entries linked together by pointers.
- QuickDraw:** The part of the Toolbox that performs all graphics operations on the Macintosh screen.
- Quit Application event:** An Apple event that requests that an application perform the tasks—such as releasing memory, asking the user to save documents, and so on—associated with quitting: one of the four required Apple events. The Finder sends this event to an application immediately after sending it a Print Documents event or if the user chooses Restart or Shut Down from the Finder's Special menu.
- radio button:** A standard Macintosh control that displays a setting, either on or off, and is part of a group in which only one button can be on at a time.
- RAM:** The random access memory, which contains exception vectors, buffers used by hardware devices, the system and application heaps, the stack, and other information used by applications.
- region:** (1) An arbitrary area or set of areas on the QuickDraw coordinate plane. The outline of a region should be one or more closed loops. (2) A linguistic or cultural entity that does not necessarily correspond to a province or nation and is associated with a number, called a **region code**, that indicates a specific localized version of Macintosh system software.
- required Apple event:** One of four core Apple events that the Finder sends to applications. These events are called Open Documents, Open Application, Print Documents, and Quit Application. They are a subset of the **core Apple events**.

**required parameter:** A keyword-specific descriptor record in an Apple event that must be specified. For example, a list of documents to open is a required parameter for the Open Documents event. **Direct parameters** are often required, and other **additional parameters** may be required. **Optional parameters** are never required.

**resource:** Data or code stored in a resource file and managed by the Resource Manager.

**resource attribute:** One of several characteristics, specified by bits in a resource reference, that determine how the resource should be dealt with.

**resource data:** In a resource file, the data that comprises a resource.

**resource file:** The resource fork of a file.

**resource fork:** The part of a file that contains data used by an application (such as menus, fonts, and icons). The resource fork of an application file also contains the application code itself.

**resource ID:** A number that, together with the resource type, identifies a resource in a resource file. Every resource has an ID number.

**Resource Manager:** The part of the Toolbox that reads and writes resources.

**resource name:** A string that, together with the resource type, identifies a resource in a resource file. A resource may or may not have a name.

**resource type:** The type of a resource in a resource file, designated by a sequence of four characters (such as 'MENU' for a menu).

**result code:** An integer indicating whether a routine completed its task successfully or was prevented by some error condition (or other special condition, such as reaching the end of a file).

**scrap:** A place where cut or copied data is stored.

**scrap file:** The file containing the desk scrap (usually named "Clipboard File").

**Scrap Manager:** The part of the Toolbox that enables cutting and pasting between applications, desk accessories, or an application and a desk accessory.

**SCSI:** See **Small Computer Standard Interface**.

**SCSI Manager:** The part of the Operating System that controls the exchange of information between a Macintosh and peripheral devices connected through the Small Computer Standard Interface (SCSI).

**selector code:** A parameter passed to the Gestalt function indicating what information about the operating environment the application currently requires.

**selector function:** The function called by the Gestalt function when an application has called Gestalt to determine information about the operating environment.

- 7.0-compatible:** Said of an application that runs without problems in system software version 7.0.
- 7.0-dependent:** Said of an application that requires the existence of features that are present only in system software version 7.0.
- 7.0-friendly:** Said of an application that is 7.0-compatible and takes advantage of some of the special features of system software version 7.0, but is still able to perform all its principal functions when operating in version 6.0.
- signature:** A resource whose type is defined by a four-character sequence that uniquely identifies an application to the Finder. A signature is located in an application's resource fork.
- Small Computer Standard Interface (SCSI):** A specification of mechanical, electrical, and functional standards for connecting small computers with intelligent peripherals such as hard disks, printers, and optical disks.
- source application:** The application that sends a particular Apple event to another application or to itself. Typically, an Apple event client sends an Apple event requesting a service from an Apple event server; in this case, the client is the source application for the Apple event. The Apple event server may return a different Apple event as a reply—in which case, the server is the source for the reply Apple event.
- source transfer mode:** One of eight transfer modes for drawing text or transferring any bit image between two bitmaps.
- stage:** Every alert has four stages, corresponding to consecutive occurrences of the alert, and a different response may be specified for each stage.
- startup screen:** When the system is started up, one of the display devices is selected as the startup screen, the screen on which the “happy Macintosh” icon appears.
- stationery pad:** A document that a user creates to serve as a template for other documents. The Finder tags a document as a stationery pad by setting the `isStationery` bit in the Finder flags field of the file's FInfo record. An application that is asked to open a stationery pad should copy the template's contents into a new document and open the document in an untitled window.
- submenu delay:** The length of time before a submenu appears as a user drags through a hierarchical main menu; it prevents rapid flashing of submenus.
- System file:** A file, located in the System Folder, that contains the basic system software plus some system resources, such as font and sound resources. In system software version 7.0, the System file behaves like a folder in this regard: although it looks like a suitcase icon, double-clicking it opens a window that reveals movable resource files (such as fonts, sounds, keyboard layouts, and script system resource collections) stored in the System file.

- system font:** The font that the system uses (in menus, for example). Its name is Chicago.
- system font size:** The size of text drawn by the system in the system font; 12 points.
- target:** The file, directory, or volume described by an alias record.
- target address:** An application signature, a process serial number, a session ID, a target ID record, or some other application-defined type that identifies the target of an Apple event.
- target application:** The application addressed to receive an Apple event. Typically, an Apple event client sends an Apple event requesting a service from an Apple event server; in this case, the server is the target application of the Apple event. The Apple event server may return a different Apple event as a reply—in which case, the client is the target of the reply Apple event.
- TextEdit:** The part of the Toolbox that supports the basic text entry and editing capabilities of a standard Macintosh application.
- 32-bit clean:** Said of an application that is able to run in an environment where all 32 bits of a memory address are used for addressing.
- thumb:** The Control Manager's term for the scroll box (the indicator of a scroll bar).
- tick:** A sixtieth of a second.
- tip:** For a help balloon, the point at the side of the rounded rectangle that indicates what object or area is explained in the help balloon.
- Toolbox Event Manager:** The part of the Toolbox that allows your application program to monitor the user's actions with the mouse, keyboard, and keypad.
- Toolbox Utilities:** The part of the Toolbox that performs generally useful operations such as fixed-point arithmetic, string manipulation, and logical operations on bits.
- transaction:** A sequence of Apple events sent back and forth between a client and a server application, beginning with the client's initial request for a service. All Apple events that are part of one transaction must have the same transaction ID.
- transaction ID:** An identifier assigned to a transaction.
- trap dispatcher:** The part of the Operating System that examines a trap word to determine what operation it stands for, looks up the address of the corresponding routine in the trap dispatch table, and jumps to the routine.
- trap dispatch table:** A table in RAM containing the addresses of all Toolbox and Operating System routines in encoded form.
- unlock:** To allow a relocatable block to be moved during heap compaction.
- update event:** An event generated by the Window Manager when a window's contents need to be redrawn.



**update region:** A window region consisting of all areas of the content region that have to be redrawn.

**User Interface Toolbox:** The software in the Macintosh ROM that helps you implement the standard Macintosh user interface in your application.

**version data:** In an application's resource file, a resource that has the application's signature as its resource type, typically a string that gives the name, version number, and date of the application.

**version number:** A number from 0 to 255 used to distinguish between files with the same name.

**virtual memory:** The part of the Operating System that allows any properly configured Macintosh computer with a memory management unit to extend the available amount of memory beyond the limits of physical RAM.

**visRgn:** The region of a grafPort, manipulated by the Window Manager, that's actually visible on the screen.

**volume:** A piece of storage medium formatted to contain files: usually a disk or part of a disk. A 3.5-inch Macintosh disk is one volume.

**window:** An object on the desktop that presents information, such as a document or a message.

**Window Manager:** The part of the Toolbox that provides routines for creating and manipulating windows.

**Window Manager port:** A grafPort that has the entire screen as its portRect and is used by the Window Manager to draw window frames.

**window template:** A resource from which the Window Manager can create a window.

**X-Ref:** An abbreviation for *cross-reference*.

## Appendix B

---

# Code Listings

---

*The following pages contain complete listings of all the source code presented in this book. The listings are presented in order by chapter. Remember, you can send in the coupon in the back of the book for a disk containing the complete set of Macintosh C Programming Primer, Volume I applications.*

## Chapter 2: Hello.c

```
/****** main *****/
```

```
void    main( void )
{
    printf( "Hello, world!" );
}
```

## Chapter 3: Hello2.c

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

#define kHorizontalPixel 30
#define kVerticalPixel   50
```

```
/******
/*  Functions  */
/******
```

```
void    ToolBoxInit( void );
void    WindowInit( void );
```

```
/****** main *****/
```

```
void    main( void )
{
    ToolBoxInit();
    WindowInit();

    while ( !Button() );
}
```

```
/****** ToolBoxInit *****/
```

```
void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
}
```

```

    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** WindowInit *****/

void WindowInit( void )
{
    WindowPtr window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 ); /* Couldn't load the WIND resource!!! */
        ExitToShell();
    }

    ShowWindow( window );
    SetPort( window );

    MoveTo( kHorizontalPixel, kVerticalPixel );
    DrawString("\pHello, world!");
}

```

### Chapter 3: Mondrian.c

```

#define kBaseResID          128
#define kMoveToFront       (WindowPtr)-1L
#define kRandomUpperLimit  32768

/*****/
/* Globals */
/*****/

long gFillColor = blackColor;

```

```

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
void    WindowInit( void );
void    MainLoop( void );
void    DrawRandomRect( void );
void    RandomRect( Rect *rectPtr );
short   Randomize( short range );


/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();
    MainLoop();
}


/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}


/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {

```

```
        SysBeep( 10 );    /* Couldn't load the WIND resource!!! */
        ExitToShell();
    }

    ShowWindow( window );
    SetPort( window );
}

/***** MainLoop *****/

void    MainLoop( void )
{
    GetDateTime( (unsigned long *)&randSeed );

    while ( ! Button() )
    {
        DrawRandomRect();

        if ( gFillColor == blackColor )
            gFillColor = whiteColor;
        else
            gFillColor = blackColor;
    }
}

/***** DrawRandomRect *****/

void    DrawRandomRect( void )
{
    Rect    randomRect;

    RandomRect( &randomRect );
    ForeColor( gFillColor );
    PaintOval( &randomRect );
}

/***** RandomRect *****/

void    RandomRect( Rect *rectPtr )
{
    WindowPtr    window;

    window = FrontWindow();
```

```

rectPtr->left = Randomize( window->portRect.right
                        - window->portRect.left );
rectPtr->right = Randomize( window->portRect.right
                        - window->portRect.left );
rectPtr->top = Randomize( window->portRect.bottom
                        - window->portRect.top );
rectPtr->bottom = Randomize( window->portRect.bottom
                        - window->portRect.top );
}

/***** Randomize *****/

short Randomize( short range )
{
    long    randomNumber;

    randomNumber = Random();

    if ( randomNumber < 0 )
        randomNumber *= -1;

    return( (randomNumber * range) / kRandomUpperLimit );
}

```

### Chapter 3: ShowPICT.c

```

#define kBaseResID        128
#define kMoveToFront      (WindowPtr)-1L

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
void    WindowInit( void );
void    DrawMyPicture( void );
void    CenterPict( PicHandle picture, Rect *destRectPtr );

```

```

/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();

    DrawMyPicture();

    while ( !Button() ) ;
}

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND resource!!! */
        ExitToShell();
    }

    ShowWindow( window );
    SetPort( window );
}

```



```

/***** DrawMyPicture *****/

void DrawMyPicture( void )
{
    Rect        pictureRect;
    WindowPtr    window;
    PicHandle    picture;

    window = FrontWindow();

    pictureRect = window->portRect;

    picture = GetPicture( kBaseResID );

    if ( picture == nil )
    {
        SysBeep( 10 );    /* Couldn't load the PICT resource!!! */
        ExitToShell();
    }

    CenterPict( picture, &pictureRect );
    DrawPicture( picture, &pictureRect );
}

/***** CenterPict *****/

void CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top );
    OffsetRect( &pictRect, (windRect.right - pictRect.right)/2,
                (windRect.bottom - pictRect.bottom)/2 );
    *destRectPtr = pictRect;
}

```

### Chapter 3: FlyingLine.c

```

#define kNumLines        50    /* Try 100 or 150 */
#define kMoveToFront      (WindowPtr)-1L
#define kRandomUpperLimit 32768

```

```

#define kEmptyString      "\p"
#define kEmptyTitle       kEmptyString
#define kVisible          true
#define kNoGoAway         false
#define kNilRefCon        (long)nil

/*****/
/*  Globals  */
/*****/

Rect      gLines[ kNumLines ];
short     gDeltaTop=3, gDeltaBottom=3; /* These four are the */
short     gDeltaLeft=2, gDeltaRight=6; /* key to flying line! */
short     gOldMBarHeight;

/*****/
/*  Functions  */
/*****/

void      ToolBoxInit( void );
void      WindowInit( void );
void      LinesInit( void );
void      MainLoop( void );
void      RandomRect( Rect *rectPtr );
short     Randomize( short range );
void      RecalcLine( short i );
void      DrawLine( short i );

/***** main *****/

void      main( void )
{
    ToolBoxInit();
    WindowInit();
    LinesInit();
    MainLoop();
}

```

```
/****** ToolBoxInit *****/
```

```
void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}
```

```
/****** WindowInit *****/
```

```
void    WindowInit( void )
{
    Rect        totalRect, mBarRect;
    RgnHandle    mBarRgn;
    WindowPtr    window;

    gOldMBarHeight = MBarHeight;
    MBarHeight = 0;

    window = NewWindow( nil, &(screenBits.bounds),
        kEmptyTitle, kVisible, plainDBox, kMoveToFront,
        kNoGoAway, kNilRefCon );

    SetRect( &mBarRect, screenBits.bounds.left,
        screenBits.bounds.top,
        screenBits.bounds.right,
        screenBits.bounds.top+gOldMBarHeight );

    mBarRgn = NewRgn();
    RectRgn( mBarRgn, &mBarRect );
    UnionRgn( window->visRgn, mBarRgn, window->visRgn );
    DisposeRgn( mBarRgn );
    SetPort( window );
    FillRect( &(window->portRect), black );
                                   /* Change black to ltGray, */
    PenMode( patXor );    /* <--- and comment out this line */
}
```

```

/***** LinesInit *****/

```

```

void    LinesInit( void )
{
    short i;

    HideCursor();
    GetDateTime( (unsigned long *)(&randSeed) );
    RandomRect( &(gLines[ 0 ]) );
    DrawLine( 0 );

    for ( i=1; i<kNumLines; i++ )
    {
        gLines[ i ] = gLines[ i-1 ];
        RecalcLine( i );
        DrawLine( i );
    }
}

```

```

/***** MainLoop *****/

```

```

void    MainLoop( void )
{
    short i;

    while ( ! Button() )
    {
        DrawLine( kNumLines - 1 );
        for ( i=kNumLines-1; i>0; i-- )
            gLines[ i ] = gLines[ i-1 ];
        RecalcLine( 0 );
        DrawLine( 0 );
    }
    MBarHeight = gOldMBarHeight;
}

```

```

/***** RandomRect *****/

```

```

void    RandomRect( Rect *rectPtr )
{
    WindowPtr    window;

    window = FrontWindow();
}

```

```

rectPtr->left = Randomize( window->portRect.right
    - window->portRect.left );
rectPtr->right = Randomize( window->portRect.right
    - window->portRect.left );
rectPtr->top = Randomize( window->portRect.bottom
    - window->portRect.top );
rectPtr->bottom = Randomize( window->portRect.bottom
    - window->portRect.top );
}

/***** Randomize *****/

short    Randomize( short range )
{
    long    randomNumber;

    randomNumber = Random();

    if ( randomNumber < 0 )
        randomNumber *= -1;

    return( (randomNumber * range) / kRandomUpperLimit );
}

/***** RecalcLine *****/

void    RecalcLine( short i )
{
    WindowPtr    window;

    window = FrontWindow();

    gLines[ i ].top += gDeltaTop;
    if ( ( gLines[ i ].top < window->portRect.top ) ||
        ( gLines[ i ].top > window->portRect.bottom ) )
    {
        gDeltaTop *= -1;
        gLines[ i ].top += 2*gDeltaTop;
    }

    gLines[ i ].bottom += gDeltaBottom;
    if ( ( gLines[ i ].bottom < window->portRect.top ) ||
        ( gLines[ i ].bottom > window->portRect.bottom ) )
    {

```

```

        gDeltaBottom *= -1;
        gLines[ i ].bottom += 2*gDeltaBottom;
    }

    gLines[ i ].left += gDeltaLeft;
    if ( ( gLines[ i ].left < window->portRect.left ) ||
        ( gLines[ i ].left > window->portRect.right ) )
    {
        gDeltaLeft *= -1;
        gLines[ i ].left += 2*gDeltaLeft;
    }

    gLines[ i ].right += gDeltaRight;
    if ( ( gLines[ i ].right < window->portRect.left ) ||
        ( gLines[ i ].right > window->portRect.right ) )
    {
        gDeltaRight *= -1;
        gLines[ i ].right += 2*gDeltaRight;
    }
}

/***** DrawLine *****/

void DrawLine( short i )
{
    MoveTo( gLines[ i ].left, gLines[ i ].top );
    LineTo( gLines[ i ].right, gLines[ i ].bottom );
}

```

## Chapter 4: EventTracker.c

```

#include <AppleEvents.h>
#include <GestaltEqu.h>
#include <Values.h>

#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L
#define kSleep              MAXLONG

#define kLeftMargin         4
#define kRowStart           285
#define kFontSize           9
#define kRowHeight          (kFontSize + 2)
#define kHorizontalOffset   0

#define kGestaltMask        1L

```

```

/*****/
/*  Globals  */
/*****/

Boolean          gDone;

/*****/
/*  Functions  */
/*****/

void              ToolBoxInit( void );
void              WindowInit( void );
void              EventInit( void );
void              EventLoop( void );
void              DoEvent( EventRecord *eventPtr );
pascal OSErr      DoOpenApp( AppleEvent theAppleEvent, AppleEvent reply,
                          long refCon );
pascal OSErr      DoOpenDoc( AppleEvent theAppleEvent, AppleEvent reply,
                          long refCon );
pascal OSErr      DoPrintDoc( AppleEvent theAppleEvent, AppleEvent reply,
                          long refCon );
pascal OSErr      DoQuitApp( AppleEvent theAppleEvent, AppleEvent reply,
                          long refCon );
void              DrawEventString( Str255 eventString );
void              HandleMouseDown( EventRecord *eventPtr );

/***** main *****/

void      main( void )
{
    ToolBoxInit();
    WindowInit();
    EventInit();

    EventLoop();
}

/***** ToolBoxInit *****/

void      ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();

```

```

    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;
    Rect          windRect;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND resource!!! */
        ExitToShell();
    }

    SetPort( window );
    TextSize( kFontSize );

    ShowWindow( window );
}

/***** EventInit *****/

void    EventInit( void )
{
    OSErr      err;
    long       feature;

    err = Gestalt( gestaltAppleEventsAttr, &feature );

    if ( err != noErr )
    {
        DrawEventString( "\pProblem in calling Gestalt!" );
        return;
    }
    else
    {

```



```

        if ( !( feature & ( kGestaltMask << gestaltAppleEventsPresent) ) )
        {
            DrawEventString( "\pApple events not available!" );
            return;
        }
    }

    err = AEInstallEventHandler( kCoreEventClass, kAEOpenApplication,
                                DoOpenApp, 0L, false );
    if ( err != noErr ) DrawEventString(
        "\pkAEOpenApplication Apple event not available!" );

    err = AEInstallEventHandler( kCoreEventClass, kAEOpenDocuments,
                                DoOpenDoc, 0L, false );
    if ( err != noErr ) DrawEventString(
        "\pkAEOpenDocuments Apple event not available!" );

    err = AEInstallEventHandler( kCoreEventClass, kAEPrintDocuments,
                                DoPrintDoc, 0L, false );
    if ( err != noErr ) DrawEventString(
        "\pkAEPrintDocuments Apple event not available!" );

    err = AEInstallEventHandler( kCoreEventClass, kAEQuitApplication,
                                DoQuitApp, 0L, false );
    if ( err != noErr ) DrawEventString(
        "\pkAEQuitApplication Apple event not available!" );
}

/***** EventLoop *****/

void    EventLoop( void )
{
    EventRecord    event;

    gDone = false;
    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, kSleep, nil ) )
            DoEvent( &event );
        /*else DrawEventString( "\pnullEvent" );*/
        /*  Uncomment the previous line for a burst of flavor!  */
    }
}

```

```

/***** DoEvent */

void DoEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
        case kHighLevelEvent:
            DrawEventString( "\pHigh level event: " );
            AEProcessAppleEvent( eventPtr );
            break;
        case mouseDown:
            DrawEventString( "\pmouseDown" );
            HandleMouseDown( eventPtr );
            break;
        case mouseUp:
            DrawEventString( "\pmouseUp" );
            break;
        case keyDown:
            DrawEventString( "\pkeyDown" );
            break;
        case keyUp:
            DrawEventString( "\pkeyUp" );
            break;
        case autoKey:
            DrawEventString( "\pautoKey" );
            break;
        case updateEvt:
            DrawEventString( "\pupdateEvt" );
            BeginUpdate( (WindowPtr)eventPtr->message );
            EndUpdate( (WindowPtr)eventPtr->message );
            break;
        case diskEvt:
            DrawEventString( "\pdiskEvt" );
            break;
        case activateEvt:
            DrawEventString( "\pactivateEvt" );
            break;
        case networkEvt:
            DrawEventString( "\pnetworkEvt" );
            break;
        case driverEvt:
            DrawEventString( "\pdriverEvt" );
            break;
        case applEvt:
            DrawEventString( "\papplEvt" );
    }
}

```

```
        break;
    case app2Evt:
        DrawEventString( "\papp2Evt" );
        break;
    case app3Evt:
        DrawEventString( "\papp3Evt" );
        break;
    case osEvt:
        DrawEventString( "\posEvt: " );
        if ( ( eventPtr->message & suspendResumeMessage )
            == resumeFlag )
            DrawString( "\pResume event" );
        else
            DrawString( "\pSuspend event" );
        break;
    }
}

/***** DoOpenApp */

pascal OSErr      DoOpenApp( AppleEvent theAppleEvent, AppleEvent reply,
                           long refCon )
{
    DrawString( "\pApple event: kAEOpenApplication" );
}

/***** DoOpenDoc */

pascal OSErr      DoOpenDoc( AppleEvent theAppleEvent, AppleEvent reply,
                           long refCon )
{
    DrawString( "\pApple event: kAEOpenDocuments" );
}

/***** DoPrintDoc */

pascal OSErr      DoPrintDoc( AppleEvent theAppleEvent, AppleEvent reply,
                           long refCon )
{
    DrawString( "\pApple event: kAEPrintDocuments" );
}
```

```

/***** DoQuitApp */
pascal OSErr DoQuitApp( AppleEvent theAppleEvent, AppleEvent reply,
                        long refCon )
{
    DrawString( "\pApple event: kAEQuitApplication" );
}

/***** DrawEventString *****/
void DrawEventString( Str255 eventString )
{
    RgnHandle tempRgn;
    WindowPtr window;

    window = FrontWindow();
    tempRgn = NewRgn();
    ScrollRect( &window->portRect, kHorizontalOffset, -kRowHeight,
                tempRgn );
    DisposeRgn( tempRgn );

    MoveTo( kLeftMargin, kRowStart );
    DrawString( eventString );
}

/***** HandleMouseDown */
void HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr window;
    long thePart;

    thePart = FindWindow( eventPtr->where, &window );

    switch ( thePart )
    {
        case inSysWindow :
            SystemClick( eventPtr, window );
            break;
        case inDrag :
            DragWindow( window, eventPtr->where, &screenBits.bounds );
            break;
        case inGoAway :
    
```

```
        gDone = true;
        break;
    }
}
```

## Chapter 4: Updater.c

```
#include <Values.h>

#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

#define kScrollbarAdjust (16-1)
#define kLeaveWhereItIs   false
#define kNormalUpdates   true

#define kMinWindowHeight 50
#define kMinWindowWidth  80

/*****/
/*  Globals  */
/*****/

Boolean      gDone;

/*****/
/*  Functions  */
/*****/

void ToolBoxInit( void );
void WindowInit( void );
void EventLoop( void );
void DoEvent( EventRecord *eventPtr );
void HandleMouseDown( EventRecord *eventPtr );
void DoUpdate( EventRecord *eventPtr );
void DoActivate( WindowPtr window, Boolean becomingActive );
void DoPicture( WindowPtr window, PicHandle picture );
void CenterPict( PicHandle picture, Rect *destRectPtr );
```

```

/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();

    EventLoop();
}

/***** ToolBoxInit */

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND resource!!! */
        ExitToShell();
    }

    SetWRefCon ( window, (long)kBaseResID );
    ShowWindow( window );

    window = GetNewWindow( kBaseResID+1, nil, kMoveToFront );

    if ( window == nil )
    {

```

```
        SysBeep( 10 );    /* Couldn't load the WIND resource!!! */
        ExitToShell();
    }

    SetWRefCon ( window, (long)( kBaseResID+1 ) );
    ShowWindow( window );
}

/***** EventLoop *****/

void    EventLoop( void )
{
    EventRecord    event;

    gDone = false;
    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, MAXLONG, nil ) )
            DoEvent( &event );
    }
}

/***** DoEvent */

void    DoEvent( EventRecord *eventPtr )
{
    Boolean    becomingActive;

    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
        case updateEvt:
            DoUpdate( eventPtr );
            break;
        case activateEvt:
            becomingActive = ( (eventPtr->modifiers & activeFlag) ==
                               activeFlag );

            DoActivate( (WindowPtr)eventPtr->message, becomingActive );
            break;
    }
}
```

```

/***** HandleMouseDown */

void    HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr    window;
    short         thePart;
    GrafPtr       oldPort;
    long          windSize;
    Rect          growRect;

    thePart = FindWindow( eventPtr->where, &window );

    switch ( thePart )
    {
        case inSysWindow :
            SystemClick( eventPtr, window );
            break;
        case inContent:
            SelectWindow( window );
            break;
        case inDrag :
            DragWindow( window, eventPtr->where, &screenBits.bounds );
            break;
        case inGoAway :
            if ( TrackGoAway( window, eventPtr->where ) )
                gDone = true;
            break;
        case inGrow:
            growRect.top = kMinWindowHeight;
            growRect.left = kMinWindowWidth;
            growRect.bottom = MAXSHORT;
            growRect.right = MAXSHORT;

            windSize = GrowWindow( window, eventPtr->where, &growRect );
            if ( windSize != 0 )
            {
                GetPort( &oldPort );
                SetPort( window );
                EraseRect( &window->portRect );
                SizeWindow( window, LoWord( windSize ),
                           HiWord( windSize ), kNormalUpdates );
                InvalRect( &window->portRect );
                SetPort( oldPort );
            }
    }
}

```



```
        break;
    case inZoomIn:
    case inZoomOut:
        if ( TrackBox( window, eventPtr->where, thePart ) )
        {
            GetPort( &oldPort );
            SetPort( window );
            EraseRect( &window->portRect );
            ZoomWindow( window, thePart, kLeaveWhereItIs );
            InvalRect( &window->portRect );
            SetPort( oldPort );
        }
        break;
    }
}

/***** DoUpdate */

void DoUpdate( EventRecord *eventPtr )
{
    short        pictureID;
    PicHandle     picture;
    WindowPtr     window;

    window = (WindowPtr)eventPtr->message;

    BeginUpdate( window );
    pictureID = GetWRefCon ( window );
    picture = GetPicture( pictureID );

    if ( picture == nil )
    {
        SysBeep( 10 );    /* Couldn't load the PICT resource!!! */
        ExitToShell();
    }

    DoPicture( window, picture );
    EndUpdate( window );
}
```

```

/***** DoActivate */

void    DoActivate( WindowPtr window, Boolean becomingActive )
{
    DrawGrowIcon( window );
}

/***** DoPicture *****/

void    DoPicture( WindowPtr window, PicHandle picture )
{
    Rect        drawingClipRect, windowRect;
    GrafPtr     oldPort;
    RgnHandle    tempRgn;

    GetPort( &oldPort );
    SetPort( window );

    tempRgn = NewRgn();
    GetClip( tempRgn );
    EraseRect( &window->portRect );

    DrawGrowIcon( window );

    drawingClipRect = window->portRect;
    drawingClipRect.right -= kScrollbarAdjust;
    drawingClipRect.bottom -= kScrollbarAdjust;

    windowRect = window->portRect;
    CenterPict( picture, &windowRect );
    ClipRect( &drawingClipRect );
    DrawPicture( picture, &windowRect );

    SetClip( tempRgn );
    DisposeRgn( tempRgn );
    SetPort( oldPort );
}

/***** CenterPict *****/

void    CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

```

```

windRect = *destRectPtr;
pictRect = (**( picture )).picFrame;
OffsetRect( &pictRect, windRect.left - pictRect.left,
            windRect.top - pictRect.top);
OffsetRect( &pictRect, (windRect.right - pictRect.right)/2,
            (windRect.bottom - pictRect.bottom)/2);
*destRectPtr = pictRect;
}

```

## Chapter 4: EventTrigger.c

```

#include <AppleEvents.h>
#include <GestaltEqu.h>

#define kGestaltMask 1L

/*****
/* Functions */
*****/

void    ToolBoxInit( void );
void    EventsInit( void );
void    SendEvent( AEEEventID theAEEEventID );

/***** main *****/

void    main( void )
{
    ToolBoxInit();
    EventsInit();

    SendEvent( kAEOpenApplication );
    SendEvent( kAEOpenDocuments );
    SendEvent( kAEPrintDocuments );
    SendEvent( kAEQuitApplication );
}

/***** ToolBoxInit */

void    ToolBoxInit( void )
{
    InitGraf( &thePort );

```

```

    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** EventsInit *****/

void    EventsInit( void )
{
    long    feature;
    OSErr    err;

    err = Gestalt( gestaltAppleEventsAttr, &feature );

    if ( err != noErr )
    {
        SysBeep( 10 );    /* Error calling Gestalt!!! */
        ExitToShell();
    }

    if ( !( feature & ( kGestaltMask << gestaltAppleEventsPresent ) ) )
    {
        SysBeep( 10 );    /* AppleEvents not supported!!! */
        ExitToShell();
    }
}

/***** SendEvent *****/

void    SendEvent( AEEEventID theAEEEventID )
{
    OSErr        err;
    AEAddressDesc    address;
    OSType        appSig;
    AppleEvent    appleEvent, reply;

    appSig = 'Prmr';

    err = AECreatDesc( typeApplSignature, (Ptr)(&appSig),
                      (Size)sizeof( appSig ), &address );

```

```
err = AECreatAppleEvent( kCoreEventClass, theAEEEventID, &address,  
    kAutoGenerateReturnID, 1L, &appleEvent );  
  
err = AESend( &appleEvent, &reply, kAENoReply + kAECanInteract,  
    kAENormalPriority, kAEDefaultTimeout, nil, nil );  
}
```

## Chapter 5: WorldClock.c

```
#include <Packages.h>  
#include <GestaltEqu.h>  
  
#define kBaseResID            128  
#define kMoveToFront          (WindowPtr)-1L  
#define kUseDefaultProc       (void *)-1L  
#define kSleep                20L  
#define kLeaveWhereItIs        false  
  
#define kIncludeSeconds        true  
#define kTicksPerSecond        60  
#define kSecondsPerHour        3600L  
  
#define kAddCheckMark          true  
#define kRemoveCheckMark       false  
  
#define kPopupControlID        kBaseResID  
#define kPopupBitMask          0x0001  
  
#define kNotANormalMenu        -1  
  
#define mApple                 kBaseResID  
#define iAbout                 1  
  
#define mFile                  kBaseResID+1  
#define iQuit                  1  
  
#define mFont                  100  
  
#define mStyle                  101  
#define iPlain                 1  
#define iBold                  2  
#define iItalic                3  
#define iUnderline             4  
#define iOutline               5  
#define iShadow                6
```

```

#define kPlainStyle          0

#define kExtraPopupPixels    25

#define kClockLeft           12
#define kClockTop            25
#define kClockSize           24

#define kCurrentTimeZone     1
#define kNewYorkTimeZone     2
#define kMoscowTimeZone      3
#define kUlanBatorTimeZone   4

#define TopLeft( r )          (*(Point *) &(r).top)
#define BottomRight( r )      (*(Point *) &(r).bottom)

#define IsHighBitSet( longNum )  ( (longNum >> 23) & 1 )
#define SetHighByte( longNum )  ( longNum |= 0xFF000000 )
#define ClearHighByte( longNum ) ( longNum &= 0x00FFFFFF )

/*****/
/*  Globals  */
/*****/

Boolean      gDone, gHasPopupControl;
short        gLastFont = 1, gCurrentZoneID = kCurrentTimeZone;
Style        gCurrentStyle = kPlainStyle;
Rect         gClockRect;

/*****/
/*  Functions  */
/*****/

void ToolBoxInit( void );
void WindowInit( void );
void MenuBarInit( void );
void EventLoop( void );
void DoEvent( EventRecord *eventPtr );
void HandleNull( EventRecord *eventPtr );
void HandleMouseDown( EventRecord *eventPtr );
void SetUpZoomPosition( WindowPtr window, short zoomInOrOut );
void HandleMenuChoice( long menuChoice );
void HandleAppleChoice( short item );

```

```
void    HandleFileChoice( short item );
void    HandleFontChoice( short item );
void    HandleStyleChoice( short item );
void    DoUpdate( EventRecord *eventPtr );
long    GetZoneOffset( void );
```

```
/****** main *****/
```

```
void    main( void )
{
    ToolBoxInit();
    WindowInit();
    MenuBarInit();

    EventLoop();
}
```

```
/****** ToolBoxInit *****/
```

```
void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}
```

```
/****** WindowInit *****/
```

```
void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
    {
        SysBeep( 10 );    /* Couldn't load the WIND resource!!! */
        ExitToShell();
    }
}
```

```

    }

    SetPort( window );
    TextSize( kClockSize );

    gClockRect = window->portRect;

    ShowWindow( window );
}

/***** MenuBarInit *****/

void    MenuBarInit( void )
{
    Handle        menuBar;
    MenuHandle     menu;
    ControlHandle  control;
    OSErr          myErr;
    long           feature;

    menuBar = GetNewMBar( kBaseResID );
    SetMenuBar( menuBar );

    menu = GetMHandle( mApple );
    AddResMenu( menu, 'DRVR' );

    menu = GetMenu( mFont );
    InsertMenu( menu, kNotANormalMenu );
    AddResMenu( menu, 'FONT' );

    menu = GetMenu( mStyle );
    InsertMenu( menu, kNotANormalMenu );
    CheckItem( menu, iPlain, true );

    DrawMenuBar();

    HandleFontChoice( gLastFont );

    myErr = Gestalt( gestaltPopupAttr, &feature );
    gHasPopupControl = ((myErr == noErr) &&
        ((feature & kPopupBitMask) == 1));

    if ( gHasPopupControl )
        control = GetNewControl( kPopupControlID, FrontWindow() );
}

```



```

/***** EventLoop *****/

void    EventLoop( void )
{
    EventRecord    event;

    gDone = false;

    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, kSleep, nil ) )
            DoEvent( &event );
        else
            HandleNull( &event );
    }
}

/***** DoEvent *****/

void    DoEvent( EventRecord *eventPtr )
{
    char    theChar;

    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
        case keyDown:
        case autoKey:
            theChar = eventPtr->message & charCodeMask;
            if ( (eventPtr->modifiers & cmdKey) != 0 )
                HandleMenuChoice( MenuKey( theChar ) );
            break;
        case updateEvt:
            DoUpdate( eventPtr );
            break;
    }
}
```

```

/***** HandleNull *****/

void HandleNull( EventRecord *eventPtr )
{
    static long    lastTime = 0;

    if ( (eventPtr->when / kTicksPerSecond) > lastTime )
    {
        InvalRect( &gClockRect );
        lastTime = eventPtr->when / kTicksPerSecond;
    }
}

/***** HandleMouseDown *****/

void HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr      whichWindow;
    GrafPtr        oldPort;
    short          thePart;
    long           menuChoice;
    ControlHandle   control;
    short          ignored;

    thePart = FindWindow( eventPtr->where, &whichWindow );
    switch ( thePart )
    {
        case inMenuBar:
            menuChoice = MenuSelect( eventPtr->where );
            HandleMenuChoice( menuChoice );
            break;
        case inSysWindow:
            SystemClick( eventPtr, whichWindow );
            break;
        case inContent:
            SetPort( whichWindow );
            GlobalToLocal( &eventPtr->where );

            if ( FindControl( eventPtr->where, whichWindow, &control ) )
            {
                ignored = TrackControl( control, eventPtr->where,
                                         kUseDefaultProc );
                gCurrentZoneID = GetCtlValue( control );
            }
    }
}

```

```

        break;
    case inDrag:
        DragWindow( whichWindow, eventPtr->where, &screenBits.bounds );
        break;
    case inZoomIn:
    case inZoomOut:
        if ( TrackBox( whichWindow, eventPtr->where, thePart ) )
        {
            SetUpZoomPosition( whichWindow, thePart );
            ZoomWindow( whichWindow, thePart, kLeaveWhereItIs );
        }
        break;
    }
}

```

```

/***** SetupZoomPosition *****/

```

```

void      SetUpZoomPosition( WindowPtr window, short zoomInOrOut )
{
    WindowPeek      wPeek;
    WStateData      *wStatePtr;
    Rect            windowRect;
    Boolean          isBig;
    short           deltaPixels;

    wPeek = (WindowPeek) window;
    wStatePtr = (WStateData *) *(wPeek->dataHandle);

    windowRect = window->portRect;
    LocalToGlobal( &TopLeft( windowRect ) );
    LocalToGlobal( &BottomRight( windowRect ) );

    wStatePtr->stdState = windowRect;
    wStatePtr->userState = wStatePtr->stdState;

    if ( gHasPopupControl )
    {
        isBig = (windowRect.bottom - windowRect.top) >
            (gClockRect.bottom - gClockRect.top);

        if ( isBig )
            deltaPixels = -kExtraPopupPixels;
        else
            deltaPixels = kExtraPopupPixels;
    }
}

```

```

        if ( zoomInOrOut == inZoomIn )
            wStatePtr->userState.bottom += deltaPixels;
        else
            wStatePtr->stdState.bottom += deltaPixels;
    }
    else
        SysBeep( 20 );
}

/***** HandleMenuChoice *****/

void    HandleMenuChoice( long menuChoice )
{
    short    menu;
    short    item;

    if ( menuChoice != 0 )
    {
        menu = HiWord( menuChoice );
        item = LoWord( menuChoice );

        switch ( menu )
        {
            case mApple:
                HandleAppleChoice( item );
                break;
            case mFile:
                HandleFileChoice( item );
                break;
            case mFont:
                HandleFontChoice( item );
                break;
            case mStyle:
                HandleStyleChoice( item );
                break;
        }
        HiliteMenu( 0 );
    }
}

/***** HandleAppleChoice *****/

void    HandleAppleChoice( short item )
{

```

```
MenuHandle    appleMenu;
Str255        accName;
short         accNumber;

switch ( item )
{
    case iAbout: /* We'll put up an about box next chapter.*/
        SysBeep( 20 );
        break;
    default:
        appleMenu = GetMHandle( mApple );
        GetItem( appleMenu, item, accName );
        accNumber = OpenDeskAcc( accName );
        break;
}
}

/***** HandleFileChoice *****/

void    HandleFileChoice( short item )
{
    switch ( item )
    {
        case iQuit :
            gDone = true;
            break;
    }
}

/***** HandleFontChoice *****/

void    HandleFontChoice( short item )
{
    short         fontNumber;
    Str255        fontName;
    MenuHandle    menuHandle;

    menuHandle = GetMHandle( mFont );

    CheckItem( menuHandle, gLastFont, kRemoveCheckMark );
    CheckItem( menuHandle, item, kAddCheckMark );

    gLastFont = item;
```

```
    GetItem( menuHandle, item, fontName );
    GetFNum( fontName, &fontNumber );

    TextFont( fontNumber );
}

/***** HandleStyleChoice *****/

void    HandleStyleChoice( short item )
{
    MenuHandle menuHandle;

    switch( item )
    {
        case iPlain:
            gCurrentStyle = kPlainStyle;
            break;
        case iBold:
            if ( gCurrentStyle & bold )
                gCurrentStyle -= bold;
            else
                gCurrentStyle |= bold;
            break;
        case iItalic:
            if ( gCurrentStyle & italic )
                gCurrentStyle -= italic;
            else
                gCurrentStyle |= italic;
            break;
        case iUnderline:
            if ( gCurrentStyle & underline )
                gCurrentStyle -= underline;
            else
                gCurrentStyle |= underline;
            break;
        case iOutline:
            if ( gCurrentStyle & outline )
                gCurrentStyle -= outline;
            else
                gCurrentStyle |= outline;
            break;
        case iShadow:
            if ( gCurrentStyle & shadow )
                gCurrentStyle -= shadow;
```

```
        else
            gCurrentStyle |= shadow;
        break;
    }

    menuHandle = GetMHandle( mStyle );

    CheckItem( menuHandle, iPlain, gCurrentStyle == kPlainStyle );
    CheckItem( menuHandle, iBold, gCurrentStyle & bold );
    CheckItem( menuHandle, iItalic, gCurrentStyle & italic );
    CheckItem( menuHandle, iUnderline, gCurrentStyle & underline );
    CheckItem( menuHandle, iOutline, gCurrentStyle & outline );
    CheckItem( menuHandle, iShadow, gCurrentStyle & shadow );

    TextFace( gCurrentStyle );
}

/***** DoUpdate *****/

void    DoUpdate( EventRecord *eventPtr )
{
    WindowPtr      window;
    Str255          timeString;
    unsigned long   curTimeInSecs;

    window = (WindowPtr)eventPtr->message;

    BeginUpdate( window );

    GetDateTime ( &curTimeInSecs );
    curTimeInSecs += GetZoneOffset();

    IUTimeString( (long)curTimeInSecs, kIncludeSeconds,
                  timeString );

    EraseRect( &gClockRect );
    MoveTo( kClockLeft, kClockTop );
    DrawString( timeString );

    DrawControls( window );

    EndUpdate( window );
}
```

```

/***** GetZoneOffset *****/

long GetZoneOffset( void )
{
    MachineLocation  loc;
    long             delta, defaultZoneOffset;

    ReadLocation( &loc );
    defaultZoneOffset = ClearHighByte( loc.gmtFlags.gmtDelta );

    if ( IsHighBitSet( defaultZoneOffset ) )
        SetHighByte( defaultZoneOffset );

    switch ( gCurrentZoneID )
    {
        case kCurrentTimeZone :
            delta = defaultZoneOffset;
            break;
        case kNewYorkTimeZone :
            delta = -5L * kSecondsPerHour ;
            break;
        case kMoscowTimeZone :
            delta = 3L * kSecondsPerHour;
            break;
        case kUlanBatorTimeZone :
            delta = 8L * kSecondsPerHour;
            break;
    }
    delta -= defaultZoneOffset;

    return delta;
}

```

## Chapter 6: Reminder.c

```

#include <Notification.h>
#include <Processes.h>
#include <Aliases.h>

#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L
#define kSleep              3600L
#define kLeaveWhereItIs      false
#define kUseDefaultProc      (void *) -1L

```



```
#define kNotANormalMenu      -1

#define mApple                kBaseResID
#define iAbout                1

#define mFile                 kBaseResID+1
#define iSetReminder          1
#define iCancelReminder       2
#define iQuit                 4

#define mHours                100
#define mMinutes              101
#define mAMorPM               102
#define mReminders            103

#define kDialogResID          kBaseResID+1

#define iHoursPopup           4
#define iMinutesPopup         5
#define iAMorPMPopup          6

#define iMessageText          8

#define iSoundCheckBox         9
#define iRotateCheckBox        10
#define iLaunchCheckBox        11

#define iAppNameText          12

#define kOn                    1
#define kOff                    0

#define kMarkApp               1

#define kAM                    1
#define kPM                     2

#define kMinTextPosition       0
#define kMaxTextPosition       32767

#define kDisableButton         255
#define kEnableButton           0
```

```

typedef struct
{
    QElem        queue;
    NMRec        notify;
    FSSpec       file;
    short        hour;
    short        minute;
    Boolean      launch;
    Str255       alert;
    Str255       menuString;
    short        menuItem;
    Boolean      dispose;
    Boolean      wasPosted;
}   ReminderRec, *ReminderPtr;

/*****/
/*  Functions  */
/*****/

void        ToolBoxInit( void );
void        MenuBarInit( void );
void        EventLoop( void );
void        DoEvent( EventRecord *eventPtr );
void        HandleNull( void );
void        HandleMouseDown( EventRecord *eventPtr );
void        HandleMenuChoice( long menuChoice );
void        HandleAppleChoice( short item );
void        HandleFileChoice( short item );

ReminderPtr HandleDialog( void );

void        GetFileName( StandardFileReply *replyPtr );

pascal void LaunchResponse( NMRecPtr notifyPtr );
pascal void NormalResponse( NMRecPtr notifyPtr );

void        CopyDialogToReminder( DialogPtr dialog,
                                ReminderPtr reminder );

ReminderPtr GetFirstReminder( void );
ReminderPtr GetNextReminder( ReminderPtr reminder );
ReminderPtr GetReminderFromNotification( NMRecPtr notifyPtr );

```

```

ReminderPtr    FindReminderOnMenu( short menuItem );
ReminderPtr    FindReminderToPost( short hour, short minute );
ReminderPtr    FindReminderToDispose( void );

void           SetupReminderMenu( void );
short          CountRemindersOnMenu( void );
void           RenumTrailingReminders( ReminderPtr reminder );
void           InsertReminderIntoMenu( ReminderPtr reminder );
void           ScheduleReminder( ReminderPtr reminder );
void           PostReminder( ReminderPtr reminder );
void           DeleteReminderFromMenu( ReminderPtr reminder );
void           DeleteReminder( ReminderPtr reminder );
ReminderPtr    DisposeReminder( ReminderPtr reminder );

void           ConcatString( Str255 str1, Str255 str2);

/* see tech note 304 */
pascal OSErr SetDialogDefaultItem(DialogPtr theDialog, short newItem)
    = { 0x303C, 0x0304, 0xAA68 };
pascal OSErr SetDialogCancelItem(DialogPtr theDialog, short newItem)
    = { 0x303C, 0x0305, 0xAA68 };
pascal OSErr SetDialogTracksCursor(DialogPtr theDialog, Boolean tracks)
    = { 0x303C, 0x0306, 0xAA68 };

/*****
/*  Globals  */
*****/

Boolean        gDone;
QHDr           gReminderQueue;

/***** main *****/

void           main( void )
{
    ToolBoxInit();
    MenuBarInit();

    EventLoop();
}

```

```
/****** ToolBoxInit */
```

```
void    ToolBoxInit( void )
{
    InitGraf( &qd.thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}
```

```
/****** MenuBarInit */
```

```
void    MenuBarInit( void )
{
    Handle          menuBar;
    MenuHandle      menu;

    menuBar = GetNewMBar( kBaseResID );
    if ( menuBar == nil )
    {
        SysBeep( 20 );
        ExitToShell();
    }

    SetMenuBar( menuBar );

    menu = GetMenu( mReminders );
    InsertMenu( menu, kNotANormalMenu );

    menu = GetMHandle( mApple );
    AddResMenu( menu, 'DRVr' );

    DrawMenuBar();
}
```

```
/****** EventLoop */
```

```
void    EventLoop( void )
{
    EventRecord      event;
```

```
gDone = false;

while ( gDone == false )
{
    if ( WaitNextEvent( everyEvent, &event, GetCaretTime(), nil ) )
        DoEvent( &event );
    else
        HandleNull();
}
}

/***** DoEvent */

void    DoEvent( EventRecord *eventPtr )
{
    char    theChar;

    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
        case keyDown:
        case autoKey:
            theChar = eventPtr->message & charCodeMask;
            if ( (eventPtr->modifiers & cmdKey) != 0 )
                HandleMenuChoice( MenuKey( theChar ) );
            break;
    }
}

/***** HandleNull *****/

void    HandleNull( void )
{
    unsigned long    time;
    DateTimeRec      dateTime;
    ReminderPtr      theReminder;

    GetDateTime( &time );
    Secs2Date( time, &dateTime );
}
```

```

theReminder = FindReminderToPost( dateTime.hour, dateTime.minute );
while ( theReminder )
{
    PostReminder( theReminder );
    DeleteReminderFromMenu( theReminder );
    theReminder = FindReminderToPost ( dateTime.hour, dateTime.minute );
}

theReminder = FindReminderToDispose();
while ( theReminder )
{
    DisposeReminder( theReminder );
    theReminder = FindReminderToDispose ();
}
}

/***** HandleMouseDown *****/

void    HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr    window;
    short         thePart;
    long          menuChoice;

    thePart = FindWindow( eventPtr->where, &window );
    switch ( thePart )
    {
        case inMenuBar:
            SetupReminderMenu();
            menuChoice = MenuSelect( eventPtr->where );
            HandleMenuChoice( menuChoice );
            break;
        case inSysWindow:
            SystemClick( eventPtr, window );
            break;
    }
}

/***** SetupReminderMenu *****/

void    SetupReminderMenu( void )
{
    MenuHandle    fileMenu;
    short         items;

```

```
fileMenu = GetMenu( mFile );
items = CountRemindersOnMenu();
if ( items ) EnableItem( fileMenu, iCancelReminder);
else DisableItem( fileMenu, iCancelReminder);
}

/***** HandleMenuChoice *****/

void    HandleMenuChoice( long menuChoice )
{
    short    menu;
    short    item;
    ReminderPtr  reminder;

    if ( menuChoice != 0 )
    {
        menu = HiWord( menuChoice );
        item = LoWord( menuChoice );

        switch ( menu )
        {
            case mApple:
                HandleAppleChoice( item );
                break;
            case mFile:
                HandleFileChoice( item );
                break;
            case mReminders:
                reminder = FindReminderOnMenu( item );
                if ( reminder )
                    DeleteReminder( reminder );
                break;
        }
        HiliteMenu( 0 );
    }
}

/***** HandleAppleChoice *****/

void    HandleAppleChoice( short item )
{
    MenuHandle    appleMenu;
    Str255        accName;
    short         accNumber;
```

```
switch ( item )
{
    case iAbout:
        NoteAlert( kBaseResID , nil );
        break;
    default:
        appleMenu = GetMHandle( mApple );
        GetItem( appleMenu, item, accName );
        accNumber = OpenDeskAcc( accName );
        break;
}
}

/***** HandleFileChoice *****/

void    HandleFileChoice( short item )
{
    ReminderPtr  reminder;

    switch ( item )
    {
        case iSetReminder:
            reminder = HandleDialog();
            if ( reminder )
                ScheduleReminder( reminder );
            break;
        case iQuit :
            gDone = true;
            break;
    }
}

/***** GetFirstReminder *****/

ReminderPtr  GetFirstReminder( void )
{
    return( (ReminderPtr)gReminderQueue.qHead );
}
```



```

/***** GetNextReminder *****/

ReminderPtr  GetNextReminder( ReminderPtr reminder )
{
    return( (ReminderPtr)reminder->queue.qLink );
}

/***** FindReminderOnMenu *****/

ReminderPtr  FindReminderOnMenu( short menuItem )
{
    ReminderPtr  theReminder;

    theReminder = GetFirstReminder();
    while ( theReminder )
    {
        if ( theReminder->menuItem == menuItem )
            break;
        theReminder = GetNextReminder( theReminder );
    }
    return( theReminder );
}

/***** FindReminderToPost *****/

ReminderPtr  FindReminderToPost( short hour, short minute )
{
    ReminderPtr  theReminder;

    theReminder = GetFirstReminder();
    while ( theReminder )
    {
        if ( (!theReminder->wasPosted)
            && (theReminder->hour <= hour)
            && (theReminder->minute <= minute) )

            break;
        theReminder = GetNextReminder (theReminder);
    }
    return( theReminder );
}

```

```

/***** FindReminderToDispose *****/

```

```

ReminderPtr FindReminderToDispose( void )
{
    ReminderPtr theReminder;

    theReminder = GetFirstReminder ();
    while ( theReminder )
    {
        if ( theReminder->dispose )
            break;
        theReminder = GetNextReminder (theReminder);
    }
    return( theReminder );
}

```

```

/***** InsertReminderIntoMenu *****/

```

```

void InsertReminderIntoMenu( ReminderPtr reminder )
{
    short itemBefore;
    MenuHandle reminderMenu;

    reminderMenu = GetMenu( mReminders );

    itemBefore = CountRemindersOnMenu();

    InsMenuItem( reminderMenu, reminder->menuString, itemBefore );

    reminder->menuItem = itemBefore + 1;
}

```

```

/***** CountRemindersOnMenu *****/

```

```

short CountRemindersOnMenu( void )
{
    MenuHandle reminderMenu;

    reminderMenu = GetMenu( mReminders );

    return( CountMItems( reminderMenu ) );
}

```

```

/***** DeleteReminderFromMenu *****/

void    DeleteReminderFromMenu( ReminderPtr reminder )
{
    MenuHandle reminderMenu;

    reminderMenu = GetMenu( mReminders );
    RenumberTrailingReminders( reminder );
    DelMenuItem( reminderMenu, reminder->menuItem );
    reminder->menuItem = 0;
}

/***** RenumberTrailingReminders *****/

void    RenumberTrailingReminders( ReminderPtr reminder )
{
    short    count;

    count = reminder->menuItem;
    reminder = GetNextReminder( reminder );
    while ( reminder )
    {
        if ( reminder->menuItem != 0)
            reminder->menuItem = count++;
        reminder = GetNextReminder( reminder );
    }
}

/***** ScheduleReminder *****/

void    ScheduleReminder( ReminderPtr reminder )
{
    Enqueue( &reminder->queue, &gReminderQueue );
    InsertReminderIntoMenu( reminder );
}

/***** PostReminder *****/

void    PostReminder( ReminderPtr reminder )
{
    reminder->notify.nmRefCon = (long)reminder;
    reminder->wasPosted = true;
    NMInstall( &reminder->notify );
}

```

```

/***** DeleteReminder *****/

```

```

void    DeleteReminder( ReminderPtr reminder )
{
    if ( reminder->menuItem )
        DeleteReminderFromMenu( reminder );

    reminder->dispose = true;
}

```

```

/***** DisposeReminder *****/

```

```

ReminderPtr    DisposeReminder( ReminderPtr reminder )
{
    ReminderPtr    next;

    if (reminder->menuItem)
        DeleteReminderFromMenu( reminder );
    next = (ReminderPtr)reminder->queue.qLink;
    Dequeue( &reminder->queue, &gReminderQueue );
    DisposePtr( (Ptr)reminder );
    return( next );
}

```

```

/***** GetFileName *****/

```

```

void    GetFileName( StandardFileReply *replyPtr )
{
    SFTYPELIST    typeList;
    short          numTypes;

    typeList[ 0 ] = 'APPL';
    numTypes = 1;

    StandardGetFile( nil, numTypes, typeList, replyPtr );
}

```

```

/***** HandleDialog *****/

```

```

ReminderPtr HandleDialog( void )
{
    DialogPtr      dialog;

```

```
Boolean          dialogDone = false;
short            itemHit, itemType;
Handle           textItemHandle;
Handle           itemHandle;
Handle           okItemHandle;
Handle           launchItemHandle;
Rect             itemRect;
Str255           itemText;
StandardFileReply reply;
ReminderPtr      reminder;

dialog = GetNewDialog( kDialogResID, nil, kMoveToFront );

ShowWindow( dialog );
SetPort( dialog );

reminder = (ReminderPtr)NewPtr( sizeof ( ReminderRec ) );
reminder->menuItem = 0;
reminder->dispose = false;
reminder->wasPosted = false;

SetDialogDefaultItem( dialog, ok );
SetDialogCancelItem( dialog, cancel );
SetDialogTracksCursor( dialog, true );

GetDItem( dialog, iMessageText, &itemType,
          &textItemHandle, &itemRect );
GetDItem( dialog, ok, &itemType, &okItemHandle, &itemRect );
GetDItem( dialog, iLaunchCheckBox, &itemType,
          &launchItemHandle, &itemRect );
SetIText( dialog, iMessageText, kMinTextPosition, kMaxTextPosition );

while ( ! dialogDone )
{
    GetIText( textItemHandle, itemText );

    if ( itemText[ 0 ] == 0 &&
        !GetCtlValue( (ControlHandle)launchItemHandle ) )
        HiliteControl( (ControlHandle)okItemHandle, kDisableButton );
    else
        HiliteControl( (ControlHandle)okItemHandle, kEnableButton );

    ModalDialog( nil, &itemHit );
}
```

```

switch ( itemHit )
{
    case ok:
    case cancel:
        dialogDone = true;
        break;
    case iSoundCheckBox:
    case iRotateCheckBox:
        GetDItem( dialog, itemHit, &itemType, &itemHandle,
                  &itemRect );
        SetCtlValue( (ControlHandle)itemHandle,
                     ! GetCtlValue( (ControlHandle)itemHandle ) );
        break;
    case iLaunchCheckBox:
    case iAppNameText:
        if ( ! GetCtlValue( (ControlHandle)launchItemHandle ) )
        {
            GetFileName( &reply );
            if ( reply.sfGood )
            {
                SetCtlValue( (ControlHandle)launchItemHandle,
                             kOn );
                reminder->file = reply.sfFile;
                GetDItem( dialog, iAppNameText, &itemType,
                          &itemHandle, &itemRect );
                SetIText( itemHandle, reminder->file.name );
            }
        }
        else
        {
            SetCtlValue( (ControlHandle)launchItemHandle, kOff );
            GetDItem( dialog, iAppNameText, &itemType,
                      &itemHandle, &itemRect );
            SetIText( itemHandle, "\p<Not Selected>" );
        }
        break;
}

}

if ( itemHit == cancel )
{
    DisposePtr( (Ptr)reminder );
    reminder = nil;
} else
    CopyDialogToReminder( dialog, reminder );

```

```

    DisposDialog( dialog );

    return( reminder );
}

/***** CopyDialogToReminder */

void      CopyDialogToReminder( DialogPtr dialog, ReminderPtr reminder)
{
    short      itemType;
    Rect        itemRect;
    Handle      itemHandle;
    Str255      string;
    MenuHandle   menu;
    short      val;
    long        tmp;

    GetDItem( dialog, iMessageText, &itemType, &itemHandle, &itemRect );
    GetIText( itemHandle, reminder->alert );
    reminder->notify.nmStr = (StringPtr)&reminder->alert;

    GetDItem( dialog, iSoundCheckBox, &itemType,
               &itemHandle, &itemRect );
    if ( GetCtlValue( (ControlHandle)itemHandle ) )
        reminder->notify.nmSound = (Handle)-1L;
    else
        reminder->notify.nmSound = nil;

    GetDItem( dialog, iRotateCheckBox, &itemType,
               &itemHandle, &itemRect );
    if( GetCtlValue( (ControlHandle)itemHandle ) )
        reminder->notify.nmIcon = GetResource( 'SICN', kBaseResID );
    else
        reminder->notify.nmIcon = nil;

    GetDItem( dialog, iLaunchCheckBox, &itemType,
               &itemHandle, &itemRect );
    if( reminder->launch = GetCtlValue( (ControlHandle)itemHandle ) )
        reminder->notify.nmResp = &LaunchResponse;
    else
        reminder->notify.nmResp = &NormalResponse;

    GetDItem( dialog, iHoursPopup, &itemType, &itemHandle, &itemRect );
    val = GetCtlValue( (ControlHandle)itemHandle );
    NumToString( (long) val, string );
    StringToNum ( string, &tmp );
}

```

```

    reminder->hour = tmp;

    reminder->menuString[0] = 0;
    ConcatString( reminder->menuString, string );
    ConcatString( reminder->menuString, "\p:" );

    GetDItem( dialog, iMinutesPopup, &itemType, &itemHandle, &itemRect );
    val = GetCtlValue( (ControlHandle)itemHandle );
    menu = GetMHandle( mMinutes );
    GetItem( menu, val, string );
    StringToNum ( string, &tmp );
    reminder->minute = tmp;

    ConcatString( reminder->menuString, string );
    ConcatString( reminder->menuString, "\p " );

    GetDItem( dialog, iAMorPMPopup, &itemType, &itemHandle, &itemRect );
    val = GetCtlValue( (ControlHandle)itemHandle );

    if( val == kPM )
        reminder->hour += 12;

    menu = GetMenu ( mAMorPM );
    GetItem( menu, val, string );
    ConcatString( reminder->menuString, string );

    reminder->notify.qType = nmType;
    reminder->notify.nmMark = kMarkApp;
}

/***** ConcatString *****/

void    ConcatString( Str255 str1, Str255 str2)
{
    short i;

    for (i=str1[0];i<str2[0]+str1[0];i++)
    {
        str1[i+1]=str2[i-str1[0]+1];
    }
    str1[0]=i;
}

```



```
/****** NormalResponse *****/
```

```
pascal void NormalResponse( NMRecPtr notifyPtr )
{
    ReminderPtr reminder;
    OSErr err;

    reminder = GetReminderFromNotification( notifyPtr );
    err = NMRemove( notifyPtr );
    reminder->dispose = true;
}
```

```
/****** LaunchResponse *****/
```

```
pascal void LaunchResponse( NMRecPtr notifyPtr )
{
    LaunchParamBlockRec launchParams;
    OSErr err;
    FSSpec fileSpec;
    ReminderPtr reminder;
    Boolean isFolder;
    Boolean wasAlias;

    reminder = GetReminderFromNotification( notifyPtr );

    fileSpec = reminder->file;

    err = ResolveAliasFile( &fileSpec, true, &isFolder, &wasAlias );

    launchParams.launchBlockID = extendedBlock;
    launchParams.launchEPBLength = extendedBlockLen;
    launchParams.launchFileFlags = 0;
    launchParams.launchControlFlags = launchContinue + launchNoFileFlags;
    launchParams.launchAppSpec = &fileSpec;

    launchParams.launchAppParameters = nil;

    if ( LaunchApplication( &launchParams ) ) SysBeep( 20 );

    err = NMRemove( notifyPtr );

    reminder->dispose = true;
}
```

```

/***** GetReminderFromNotification *****/

```

```

ReminderPtr  GetReminderFromNotification( NMRecPtr notifyPtr )
{
    return (ReminderPtr) notifyPtr->nmRefCon;
}

```

## Chapter 7: ResWriter.c

```

#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L

```

```

#define iText           4

```

```

#define kDisableButton  255
#define kEnableButton   0

```

```

#define kWriteTextOut   true
#define kDontWriteTextOut false

```

```

#define kMinTextPosition 0
#define kMaxTextPosition 32767

```

```

/*****/
/*  Functions  */
/*****/

```

```

void      ToolBoxInit( void );
Boolean   DoTextDialog( StringHandle oldTextHandle );

```

```

pascal OSErr SetDialogDefaultItem( DialogPtr theDialog,
                                   short newItem ) = { 0x303C, 0x0304, 0xAA68 };
pascal OSErr SetDialogCancelItem( DialogPtr theDialog,
                                   short newItem ) = { 0x303C, 0x0305, 0xAA68 };
pascal OSErr SetDialogTracksCursor( DialogPtr theDialog,
                                   Boolean tracks ) = { 0x303C, 0x0306, 0xAA68 };

```

```

/***** main *****/

```

```

void      main( void )
{
    StringHandle textHandle;

    ToolBoxInit();

```

```
textHandle = GetString( kBaseResID );

if ( textHandle == nil )
{
    SysBeep( 20 );
    ExitToShell();
}

if ( DoTextDialog( textHandle ) == kWriteTextOut )
{
    ChangedResource( (Handle)textHandle );
    WriteResource( (Handle)textHandle );
}
}

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** DoTextDialog *****/

Boolean DoTextDialog( StringHandle textHandle )
{
    DialogPtr    dialog;
    Boolean      done;
    short        itemHit, itemType;
    Handle       OKItemHandle, textItemHandle;
    Rect         itemRect;
    Str255       itemText;

    dialog = GetNewDialog( kBaseResID, nil, kMoveToFront );

    GetDItem( dialog, ok, &itemType, &OKItemHandle, &itemRect );
    GetDItem( dialog, iText, &itemType, &textItemHandle, &itemRect );
}
```

```
HLock( (Handle)textHandle );
SetIText( textItemHandle, *textHandle );
HUnlock( (Handle)textHandle );

SelIText( dialog, iTText, kMinTextPosition, kMaxTextPosition );

ShowWindow( dialog );
SetPort( dialog );

SetDialogDefaultItem( dialog, ok );
SetDialogCancelItem( dialog, cancel );
SetDialogTracksCursor( dialog, true );

done = false;
while ( ! done )
{
    GetIText( textItemHandle, itemText );

    if ( itemText[ 0 ] == 0 )
        HiliteControl( (ControlHandle)OKItemHandle, kDisableButton );
    else
        HiliteControl( (ControlHandle)OKItemHandle, kEnableButton );

    ModalDialog( nil, &itemHit );

    done = ( (itemHit == ok) || (itemHit == cancel) );
}

if ( itemHit == ok )
{
    GetIText( textItemHandle, itemText );
    SetHandleSize( (Handle)textHandle, (Size)(itemText[ 0 ] + 1) );

    HLock( (Handle)textHandle );
    GetIText( textItemHandle, *textHandle );
    HUnlock( (Handle)textHandle );

    DisposDialog( dialog );

    return( kWriteTextOut );
}
else
{
    DisposDialog( dialog );
}
```

```

        return( kDontWriteTextOut );
    }
}

```

## Chapter 7: Pager.c

```

#include <Values.h>

#define kBaseResID            128
#define kMoveToFront         (WindowPtr)-1L
#define kScrollBarWidth      16
#define kNilActionProc       nil
#define kSleep                MAXLONG

#define kVisible              true
#define kStartValue           1
#define kMinValue             1
#define kNilRefCon            0L
#define kEmptyTitle           "\p"

#define kEmptyString          "\p"
#define kNilFilterProc        nil

#define kErrorAlertID         kBaseResID

/*****/
/*  Globals  */
/*****/

Boolean      gDone;

/*****/
/*  Functions  */
/*****/

void      ToolBoxInit( void );
void      WindowInit( void );
void      SetUpScrollBar( WindowPtr window );
pascal void ScrollProc( ControlHandle theControl, short partCode );
void      EventLoop( void );
void      DoEvent( EventRecord *eventPtr );
void      HandleMouseDown( EventRecord *eventPtr );
void      UpdateWindow( WindowPtr window );

```

```
void    CenterPict( PicHandle picture, Rect *destRectPtr );
void    DoError( Str255 errorString, Boolean fatal );
```

```
/****** main *****/
```

```
void    main( void )
{
    ToolBoxInit();
    WindowInit();

    EventLoop();
}
```

```
/****** ToolBoxInit *****/
```

```
void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}
```

```
/****** WindowInit *****/
```

```
void    WindowInit( void )
{
    WindowPtr    window;

    if ( ( window = GetNewWindow( kBaseResID, nil,
                                kMoveToFront ) ) == nil )
        DoError( "\pCan't Load WIND resource!", true );

    SetUpScrollBar( window );

    ShowWindow( window );
    SetPort( window );
}
```

```
/****** SetupScrollBar *****/
```

```
void SetupScrollBar( WindowPtr window )
{
    Rect          vScrollRect;
    short          numPictures;
    ControlHandle  scrollBarH;

    if ( ( numPictures = CountResources( 'PICT' ) ) <= 0 )
        DoError( "\pNo PICT resources were found!", true );

    vScrollRect = window->portRect;
    vScrollRect.top -= 1;
    vScrollRect.bottom += 1;
    vScrollRect.left = vScrollRect.right - kScrollBarWidth + 1;
    vScrollRect.right += 1;

    scrollBarH = NewControl( window, &vScrollRect,
        kEmptyTitle, kVisible, kStartValue, kMinValue,
        numPictures, scrollBarProc, kNilRefCon );
}
```

```
/****** ScrollProc *****/
```

```
pascal void ScrollProc( ControlHandle theControl, short partCode )
{
    short          curCtlValue, maxCtlValue, minCtlValue;
    WindowPtr      window;

    maxCtlValue = GetCtlMax( theControl );
    curCtlValue = GetCtlValue( theControl );
    minCtlValue = GetCtlMin( theControl );

    window = (**theControl).ctrlOwner;

    switch ( partCode )
    {
        case inPageDown:
        case inDownButton:
            if ( curCtlValue < maxCtlValue )
            {
                curCtlValue += 1;
                SetCtlValue( theControl, curCtlValue );
                UpdateWindow( window );
            }
    }
}
```

```

        }
        break;
    case inPageUp:
    case inUpButton:
        if ( curCtlValue > minCtlValue )
        {
            curCtlValue -= 1;
            SetCtlValue( theControl, curCtlValue );
            UpdateWindow( window );
        }
    }
}

/***** EventLoop *****/

void    EventLoop( void )
{
    EventRecord    event;

    gDone = false;

    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, kSleep, nil ) )
            DoEvent( &event );
    }
}

/***** DoEvent *****/

void    DoEvent( EventRecord *eventPtr )
{
    WindowPtr    window;

    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
        case updateEvt:
            window = (WindowPtr)eventPtr->message;

            BeginUpdate( window );

```



```

        DrawControls( window );
        UpdateWindow( window );
        EndUpdate( window );
        break;
    }
}

/***** HandleMouseDown *****/

void    HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr    window;
    short        thePart;
    Point        thePoint;
    ControlHandle theControl;

    thePart = FindWindow( eventPtr->where, &window );
    switch ( thePart )
    {
        case inSysWindow :
            SystemClick( eventPtr, window );
            break;
        case inDrag :
            DragWindow( window, eventPtr->where, &screenBits.bounds );
            break;
        case inContent:
            thePoint = eventPtr->where;
            GlobalToLocal( &thePoint );

            thePart = FindControl( thePoint, window, &theControl );

            if ( theControl == ((WindowPeek)window)->controlList )
            {
                if ( thePart == inThumb )
                {
                    thePart = TrackControl( theControl,
                                            thePoint, kNilActionProc );
                    InvalRect( &(window->portRect) );
                }
                else
                    thePart = TrackControl( theControl, thePoint,
                                            &ScrollProc );
            }
            break;
    }
}

```

```

        case inGoAway :
            gDone = true;
            break;
    }
}

/***** UpdateWindow *****/

void    UpdateWindow( WindowPtr window )
{
    PicHandle    currentPicture;
    Rect          windowRect;
    RgnHandle     tempRgn;

    tempRgn = NewRgn();
    GetClip( tempRgn );

    windowRect = window->portRect;
    windowRect.right -= kScrollBarWidth;
    EraseRect( &windowRect );

    ClipRect( &windowRect );

    currentPicture = (PicHandle)GetIndResource( 'PICT',
        GetCtlValue( ((WindowPeek)window)->controlList ) );

    if ( currentPicture == nil )
        DoError( "\pCan't Load PICT resource!", true );

    CenterPict( currentPicture, &windowRect );
    DrawPicture( currentPicture, &windowRect );

    SetClip( tempRgn );
    DisposeRgn( tempRgn );
}

/***** CenterPict *****/

void    CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect    windRect, pictRect;

    windRect = *destRectPtr;

```

```

    pictRect = (**( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
               windRect.top - pictRect.top);
    OffsetRect( &pictRect, (windRect.right - pictRect.right)/2,
               (windRect.bottom - pictRect.bottom)/2);
    *destRectPtr = pictRect;
}

/***** DoError *****/

void    DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString, kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}

```

## Chapter 7: ShowClip.c

```

#define kBaseResID            128
#define kMoveToFront          (WindowPtr)-1L

#define kEmptyString          "\p"
#define kNilFilterProc        nil

#define kErrorAlertID         kBaseResID

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
void    WindowInit( void );
void    MainLoop( void );
void    CenterPict( PicHandle picture, Rect *destRectPtr );
void    DoError( Str255 errorString, Boolean fatal );

```

```

/***** main *****/

void    main( void )
{
    ToolBoxInit();
    WindowInit();
    MainLoop();
}

/***** ToolBoxInit *****/

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** WindowInit *****/

void    WindowInit( void )
{
    WindowPtr    window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
        DoError( "\pCan't load the WIND resource!", true );

    ShowWindow( window );
    SetPort( window );
}

/***** MainLoop *****/

void    MainLoop( void )
{
    Rect    pictureRect;
```

```

Handle      clipHandle;
long        length, offset;
WindowPtr   window;

clipHandle = NewHandle( 0 );
window = FrontWindow();

if ( ( length = GetScrap( clipHandle, 'TEXT', &offset ) ) < 0 )
{
    if ( GetScrap( clipHandle, 'PICT', &offset ) < 0 )
        DoError( "\pThere's no PICT and no text in the scrap...",
                  true );
    else
    {
        pictureRect = window->portRect;
        CenterPict( (PicHandle)clipHandle, &pictureRect );
        DrawPicture( (PicHandle)clipHandle, &pictureRect );
    }
}
else
{
    HLock( clipHandle );
    TextBox( *clipHandle, length, &(window->portRect), teJustLeft );
    HUnlock( clipHandle );
}

while ( !Button() ) ;
}

/***** CenterPict *****/

void      CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect   windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top    - pictRect.top);
    OffsetRect( &pictRect, (windRect.right - pictRect.right)/2,
                (windRect.bottom - pictRect.bottom)/2);
    *destRectPtr = pictRect;
}

```

```

/***** DoError *****/

void    DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString, kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}

```

## Chapter 7: SoundMaker.c

```

#include <Sound.h>
#include <SoundInput.h>
#include <GestaltEqu.h>

#define kBaseResID            128

#define kNilSoundChannel      nil
#define kSynchronous          false

#define kEmptyString          "\p"
#define kNilFilterProc        nil

#define kErrorAlertID         kBaseResID

/*****/
/*  Functions  */
/*****/

void    ToolBoxInit( void );
Handle  RecordSound( void );
void    PlaySound( Handle soundHandle );
void    DoError( Str255 errorString, Boolean fatal );

/*****/
***** main *****/

void    main( void )
{
    Handle    soundHandle;
    long      feature;
}

```

```
OSErr      err;

MaxApplZone();
ToolBoxInit();

err = Gestalt( gestaltSoundAttr, &feature );

if ( err != noErr )
    DoError( "\pError returned by Gestalt!", true );

if ( feature & (1 << gestaltHasSoundInputDevice) )
{
    soundHandle = RecordSound();
    PlaySound( soundHandle );
    DisposHandle( soundHandle );
}
else
    DoError( "\pSound input device not available!!!", true );
}

/***** ToolboxInit *****/

void      ToolboxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** RecordSound *****/

Handle    RecordSound( void )
{
    OSErr      err;
    Point      upperLeft;
    Handle      soundHandle;

    SetPt( &upperLeft, 50, 50 );
```

```

    soundHandle = nil;

    err = SndRecord( nil, upperLeft, siBestQuality, &soundHandle );

    if ( err == userCanceledErr )
        DoError( "\pRecording canceled...", true );

    if ( err != 0 )
        DoError( "\pError returned by SndRecord()...", true );

    return( soundHandle );
}

/***** PlaySound *****/

void    PlaySound( Handle soundHandle )
{
    OSErr    err;

    err = SndPlay( kNilSoundChannel, soundHandle, kSynchronous );

    if ( err != noErr )
        DoError( "\pError returned by SndPlay()...", true );
}

/***** DoError *****/

void    DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString, kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}

```



**Chapter 7: OpenPICT.c**

```
#include <GestaltEqu.h>

#define kBaseResID          128
#define kMoveToFront        (WindowPtr)-1L

#define kEmptyString        "\p"
#define kNilFilterProc      nil

#define kErrorAlertID       kBaseResID

#define kPICTHeaderSize     512

/*****
/*  Functions  */
*****/

void          ToolBoxInit( void );
void          GetFileName( StandardFileReply *replyPtr );
PicHandle     LoadPICTFile( StandardFileReply *replyPtr );
void          WindowInit( void );
void          DrawMyPicture( PicHandle picture );
void          CenterPict( PicHandle picture, Rect *destRectPtr );
void          DoError( Str255 errorString, Boolean fatal );

/***** main *****/

void  main( void )
{
    PicHandle     picture;
    StandardFileReply  reply;

    ToolBoxInit();

    GetFileName( &reply );

    if ( reply.sfGood )
    {
        picture = LoadPICTFile( &reply );

        if ( picture != nil )
```

```

    {
        WindowInit();
        DrawMyPicture( picture );

        while ( ! Button() ) ;
    }
}

```

/\*\*\*\*\*\* ToolBoxInit \*\*\*\*\*/

```

void    ToolBoxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

```

/\*\*\*\*\*\* GetFileName \*\*\*\*\*/

```

void    GetFileName( StandardFileReply *replyPtr )
{
    SFTYPELIST    typeList;
    short          numTypes;
    long           feature;
    OSERR          err;

    err = Gestalt( gestaltStandardFileAttr, &feature );

    if ( err != noErr )
        DoError( "\pError returned by Gestalt!", true );

    if ( feature & (1 << gestaltStandardFile58) )
    {
        typeList[ 0 ] = 'PICT';
        numTypes = 1;

        StandardGetFile( kNilFilterProc, numTypes, typeList, replyPtr );
    }
}

```

```
else
{
    DoError( "\pThe new Standard File routines \
are not supported by this OS!", true );
}
}

/***** LoadPICTFile *****/

PicHandle    LoadPICTFile( StandardFileReply *replyPtr )
{
    short      srcFile;
    PicHandle   picture;
    char        pictHeader[ kPICTHeaderSize ];
    long        pictSize, headerSize;
    long        feature;
    OSErr       err;

    err = Gestalt( gestaltFSAttr, &feature );

    if ( err != noErr )
        DoError( "\pError returned by Gestalt!", true );

    if ( feature & (1 << gestaltHasFSSpecCalls) )
    {
        if ( FSpOpenDF( &(replyPtr->sfFile), fsRdPerm, &srcFile )
            != noErr )
        {
            DoError( "\pCan't open file...", false );
            return( nil );
        }

        if ( GetEOF( srcFile, &pictSize ) != noErr )
        {
            DoError( "\pError returned by GetEOF()...", false );
            return( nil );
        }

        headerSize = kPICTHeaderSize;

        if ( FSRead( srcFile, &headerSize, pictHeader ) != noErr )
        {
            DoError( "\pError reading file header...", false );
            return( nil );
        }
    }
}
```

```

    }

    pictSize -= kPICTHeaderSize;

    if ( ( picture = (PicHandle)NewHandle( pictSize ) ) == nil )
    {
        DoError( "\pNot enough memory to read picture...", false );
        return( nil );
    }

    HLock( (Handle)picture );

    if ( FSRead( srcFile, &pictSize, *picture ) != noErr )
    {
        DoError( "\pError returned by FSRead()...", false );
        return( nil );
    }

    HUnlock( (Handle)picture );
    FSClose( srcFile );

    return( picture );
}
else
{
    DoError( "\pThe new FSSpec File Manager routines \
are not supported by this OS!", true );
}
}

/***** WindowInit *****/

void WindowInit( void )
{
    WindowPtr window;

    window = GetNewWindow( kBaseResID, nil, kMoveToFront );

    if ( window == nil )
        DoError( "\pCan't load WIND resource...", true );

    ShowWindow( window );
    SetPort( window );
}

```

```
/****** DrawMyPicture *****/
```

```
void DrawMyPicture( PicHandle picture )
{
    Rect      pictureRect;
    WindowPtr window;

    window = FrontWindow();

    pictureRect = window->portRect;

    CenterPict( picture, &pictureRect );
    DrawPicture( picture, &pictureRect );
}
```

```
/****** CenterPict *****/
```

```
void CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top );
    OffsetRect( &pictRect, (windRect.right - pictRect.right)/2,
                (windRect.bottom - pictRect.bottom)/2 );
    *destRectPtr = pictRect;
}
```

```
/****** DoError *****/
```

```
void DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString, kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}
```

## Chapter 7: PrintPICT.c

```
#include <PrintTraps.h>

#define kBaseResID            128

#define kDontScaleOutput      nil

#define kEmptyString          "\p"
#define kNilFilterProc        nil

#define kErrorAlertID         kBaseResID

/*****/
/*  Functions  */
/*****/

void          ToolBoxInit( void );
PicHandle     LoadPICT( void );
THPrint       PrintInit( void );
Boolean       DoDialogs( THPrint printRecordH );
void          PrintPicture( PicHandle picture, THPrint printRecordH );
void          CenterPict( PicHandle picture, Rect *destRectPtr );
void          DoError( Str255 errorString, Boolean fatal );

/*****/
***** main *****/

void          main( void )
{
    PicHandle   picture;
    THPrint     printRecordH;

    ToolBoxInit();

    picture = LoadPICT();
    printRecordH = PrintInit();

    if ( DoDialogs( printRecordH ) )
        PrintPicture( picture, printRecordH );
}
```

```

/***** ToolboxInit *****/

void    ToolboxInit( void )
{
    InitGraf( &thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( nil );
    InitCursor();
}

/***** LoadPICT *****/

PicHandle    LoadPICT( void )
{
    PicHandle    picture;

    picture = GetPicture( kBaseResID );

    if ( picture == nil )
        DoError( "\pCan't load PICT resource...", true );

    return( picture );
}

/***** PrintInit *****/

THPrint    PrintInit( void )
{
    THPrint    printRecordH;

    printRecordH = (THPrint)NewHandle( sizeof( TPrint ) );

    if ( printRecordH == nil )
        DoError( "\pNot enough memory to allocate print record", true );

    PrOpen();
    PrintDefault( printRecordH );

    return( printRecordH );
}
```

```

/***** DoDialogs *****/

Boolean DoDialogs( THPrint  printRecordH )
{
    Boolean confirmed = true;

    confirmed = PrStlDialog( printRecordH );

    if ( confirmed )
        confirmed = PrJobDialog( printRecordH );

    return( confirmed );
}

/***** PrintPicture *****/

void PrintPicture( PicHandle picture, THPrint printRecordH )
{
    TPrPort      printPort;
    Rect         pictureRect;
    TPrStatus    printStatus;

    printPort = PrOpenDoc( printRecordH, nil, nil );

    PrOpenPage( printPort, kDontScaleOutput );

    if ( PrError() != noErr )
        DoError( "\pError returned by PrOpenPage()...", true );

    pictureRect = (**printRecordH).prInfo.rPage;

    CenterPict( picture, &pictureRect );
    DrawPicture( picture, &pictureRect );

    PrClosePage( printPort );
    PrCloseDoc( printPort );

    if ( (**printRecordH).prJob.bJDocLoop == bSpoolLoop )
        PrPicFile( printRecordH, nil, nil, nil, &printStatus );

    PrClose();
    DisposHandle( (Handle)printRecordH );
}

```



```
/****** CenterPict *****/

void      CenterPict( PicHandle picture, Rect *destRectPtr )
{
    Rect  windRect, pictRect;

    windRect = *destRectPtr;
    pictRect = (*( picture )).picFrame;
    OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top - pictRect.top);
    OffsetRect( &pictRect, (windRect.right - pictRect.right)/2,
                (windRect.bottom - pictRect.bottom)/2);
    *destRectPtr = pictRect;
}

/****** DoError *****/

void      DoError( Str255 errorString, Boolean fatal )
{
    ParamText( errorString, kEmptyString, kEmptyString, kEmptyString );

    StopAlert( kErrorAlertID, kNilFilterProc );

    if ( fatal )
        ExitToShell();
}
```

## Appendix C

---

# THINK C Command Summary

*This appendix summarizes some  
of the basic operations of THINK C,  
Version 5.*

THINK C Is a simple but powerful programming environment. This appendix provides an overview of its operations.

## The Project Menu

THINK C keeps track of all aspects of your current program in a **project file**. All object code is stored in the project file, as well as information describing the compilation state and interdependence of all files that make up the project. To create a project, select **New Project...** from the **Project** menu. To open an existing project, use **Open Project**. If you have a project open, and you want to look at another project, use **Close Project**. If you choose **Close & Compact**, the project file will be compressed. This makes it smaller, but means that it takes longer to open. **Set Project Type...** brings up a dialog box, which allows you to specify the type of project you'd like to build (Figure C.1). There are four types of projects: applications, desk accessories, device drivers, and code resources.

● Application  
○ Desk Accessory  
○ Device Driver  
○ Code Resource

File Type   
Creator

Partition (K)   
SIZE Flags

☐ Far CODE  
☐ Far DATA  
☐ Separate STRS

Figure C.1 The Set Project Type... dialog box.

## Applications

In most cases, you'll use THINK C to build a standard, double-clickable application. Use the **Set Project Type...** dialog to enter the application's **File Type** and **Creator**. The default **File Type** is **APPL** and the default **Creator** is **????**. For more on application file and creator types, go back to Chapter 8.

The **Far CODE** check box tells your application to reference all code by using 32-bit addresses instead of 16-bit addresses. This allows you a larger jump table (256K instead of the normal 32K). **Far CODE** causes your application to grow by about 6% due to the larger addresses.

Similarly, the **Far DATA** check box causes program data references to use 32-bit addresses instead of 16-bit addresses. With **Far DATA** unchecked, your application is limited to 32K of global data. With **Far DATA** checked, you are limited to 32K of global data per file in your project. **Far DATA** causes your application to grow by about 8% due to the larger addresses.

**Separate STRS** asks THINK C to store string literals in a separate STRS resource. This option is not particularly useful and is available primarily for compatibility with older versions of THINK C.

Specify your application's preferred memory partition in the **Partition (K)** field. Specify the partition in multiples of 1024 bytes.

Finally, the **Set Project Type...** dialog allows you to completely specify the **SIZE** resource. The **SIZE** resource is detailed in Chapter 8.

## Desk Accessories and Device Drivers

Although THINK C still supports them, System 7 has made desk accessories obsolete. If you're thinking of writing a desk accessory, Apple recommends that you write a small application instead.

If you are building a driver, your **Set Project Type...** dialog will look like the one shown in Figure C.2. Normally, drivers are limited to a single segment. No longer! The **Multi-Segment** check box allows you to build a driver of up to 31 segments.

The **Name** field contains the name of the desk accessory or device driver resource. THINK C will add a null byte to the beginning of your desk accessory (a convention). It will also place a period before the device driver name if you don't put one there.

Drivers always have a **Type** of **DRVR**. Enter the resource ID you'd like your driver saved as in the **ID** field.

## Code Resource

THINK C allows you to build standalone CODE resources. This comes in really handy for building INITs, WDEFs, and cdevs. The code

The dialog box for setting project type for a Desk Accessory. It features four radio buttons on the left: Application, Desk Accessory, Device Driver (selected), and Code Resource. To the right are two text fields for File Type and Creator, both containing '????', and a checkbox for Multi-Segment. Below these is a Name text field. At the bottom left is a Type dropdown menu showing 'DRVR', and at the bottom right is an ID text field. At the very bottom are OK and Cancel buttons.

**Figure C.2** The Set Project Type... dialog box for a Desk Accessory.

resource Set Project Type... dialog has the same basic fields as the driver dialog (Figure C.3). If the **Custom Header** check box is unchecked, THINK C builds a standard 16-byte header for your CODE resource. The header places the address of your resource into register A0 and branches to your `main()` function. Check the **Custom Header** check box if you plan on adding your own custom header to the CODE resource.

The **Attrs** field allows you to select the standard resource attributes for your code resource. You can use the pop-up menu

The dialog box for setting project type for a CODE resource. It features four radio buttons on the left: Application, Desk Accessory, Device Driver, and Code Resource (selected). To the right are two text fields for File Type and Creator, both containing '????', and a checkbox for Multi-Segment. Below these is a Name text field. At the bottom left is a Type text field. At the bottom right is an ID text field. Below the ID field is an Attrs field with a pop-up menu icon and the value '20'. At the very bottom are OK and Cancel buttons.

Attrs pop-up menu:

- ✓ Purgeable
- Locked
- System Heap
- Preloaded
- Protected

**Figure C.3** The CODE resource Set Project Type... dialog.

(shown to the right of the dialog box in Figure C.3) to set these attributes or you can type a value directly in the text edit field. You can also use ResEdit to set these flags.

## More on the Project Menu

Selecting **Remove Objects** from the **Project** menu removes the object code for all files in your project. Doing this can significantly reduce the size of your project and ensure that you'll have to reload or recompile all project files the next time you run your project. **Remove Objects** if you plan on filing your project away for a while, or to ensure that you are using the latest version of all files and libraries (such as MacTraps). **Bring Up to Date** compiles source code and loads library files that haven't been compiled or loaded yet. **Check Link** checks the link-worthiness of your project without running it.

**Build Library...** takes the current project and saves it as a binary library so it can be used by other projects. **Build Application...** (or Desk Accessory, Device Driver, or CODE Resource) saves the project as a standalone application (or desk accessory, device driver, or CODE resource), depending on the project type chosen.

If you set the **Use Debugger** flag, the debugger will be launched automatically the next time you run your application. The final menu item, **Run**, runs your application as a separate entity.

## The File Menu

Most of the options in the **File** menu are self-explanatory (Figure C.4). To create a new file, select **New**. To open an existing file, choose **Open**. You can also open a file by double-clicking its name in the Project window. If a file name is highlighted in an edit window, **Open Selection** will open the file. **Close** will close your file; you are prompted to save or discard your changes if any have been made.

**Save** will save the current file you're working on. **Save As...** will save your current file under a new name and change the name in the Project window. **Save A Copy As...** will save your current file under a different name and use the original file. **Revert** will return the current file to the saved version of that file. **Page Setup...** and **Print...** perform their usual functions. **Transfer...** allows you to go directly to another program without going back to the Finder. **Quit** allows you to leave THINK C; you are prompted to save changes in any open files.

File	
New	⌘N
Open...	⌘O
Open Selection	⌘D
Close	⌘W
-----	
Save	⌘S
Save As...	
Save A Copy As...	
Revert	
-----	
Page Setup...	
Print...	⌘P
-----	
Transfer...	
Quit	⌘Q

Figure C.4 THINK C's File menu.

---

## The Edit Menu

---

The **Edit** Menu provides options for working on your current file (Figure C.5). The **Undo**, **Cut**, **Copy**, **Paste**, **Clear**, and **Select All** menu items are the standard text editing options available on most Macintosh applications. **Set Tabs & Font...** allows you to select the tab size (usually, one tab every four character positions) as well as the font type and size for any open source code file.

**Shift Left** and **Shift Right** will move selected text one tab to the right or left. **Balance** will highlight the code balanced by the nearest **()**, **[]**, or **{ }** before and after the cursor position. **Options...** brings up a dialog box that allows you to set the default options for six different areas of THINK C (Figure C.6).

The popup at the top of the **Options...** dialog box allows you to select one of **Preferences**, **Language Settings**, **Compiler Settings**, **Code Optimization**, **Debugging**, and **Prefix** (Figure C.7). Each of these settings brings up its own set of buttons and check boxes. The **Options...** dialog has a built-in help facility. To find out what any specific item does, click on it and a complete description appears in the help area toward the bottom of the dialog box.

Edit	
Undo Paste	⌘Z
Cut	⌘H
Copy	⌘C
Paste	⌘V
Clear	
Select All	⌘A
Set Tabs & Font...	
Shift Left	⌘[
Shift Right	⌘]
Balance	⌘B
Options...	⌘;

Figure C.5 THINK C's Edit menu.

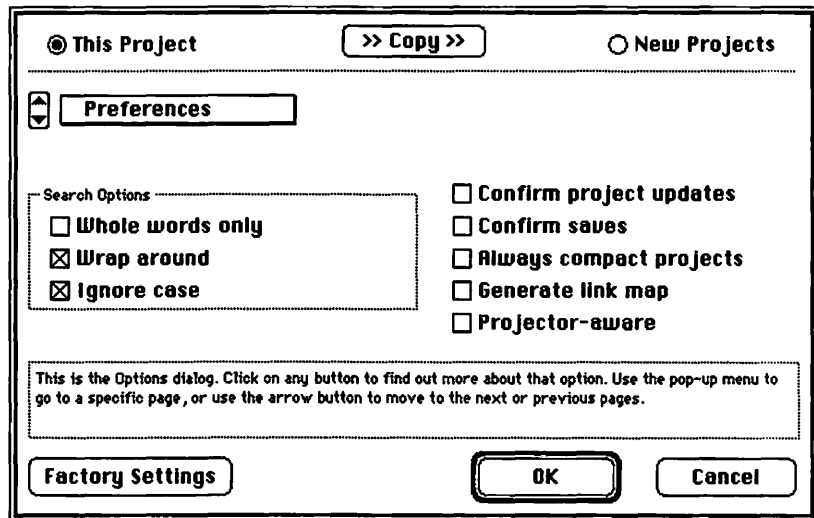
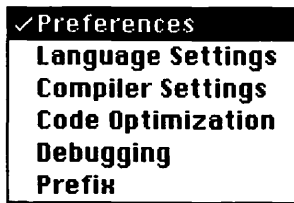


Figure C.6 THINK C's Options... dialog box showing settings for Preferences.





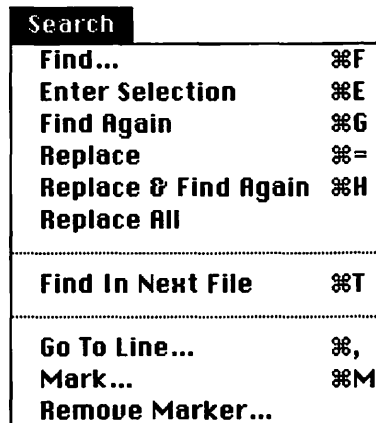
**Figure C.7** The Options... pop-up menu.

The **Factory Settings** button at the bottom of the dialog box resets that screen's settings to the way they were when you first installed THINK C. The two radio buttons at the top of the window, **This Project** and **New Projects**, allow you to set options for the current project only or set options that will carry over to all new projects. The **Copy** button allows you to copy settings between these two states.

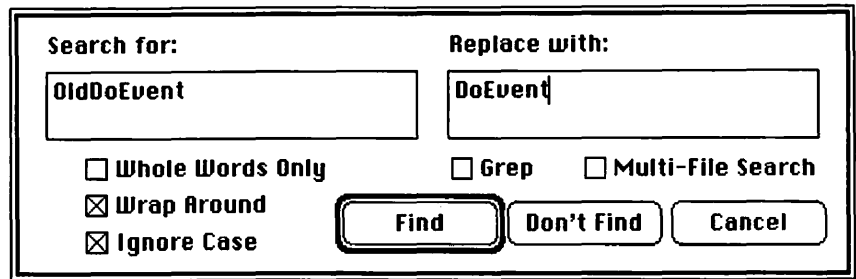
## The Search Menu

The **Search** menu (Figure C.8) offers a number of options that allow you to find and change text in your files.

**Find...** brings up a dialog box that allows you to specify a search string and, optionally, a replacement string (Figure C.9). **Whole Words Only** restricts search matches to whole words only. **Wrap Around** tells **Find...** to continue searching from the top of the file once



**Figure C.8** THINK C's Search menu.



**Figure C.9** The Find... dialog box.

the bottom is reached. **Ignore Case** tells **Find...** to treat upper- and lowercase letters equally. If **Grep** is checked, a utility similar to the Grep utility in UNIX is run. If **Multi-File Search** is checked, a dialog box is put up to allow you to select which files to search.

**Enter Selection** copies any currently highlighted text into the **Search for:** field in the **Find...** dialog box. **Find Again** searches for the next occurrence of the **Search for:** text without bringing up the dialog box again. **Replace** replaces the highlighted string with the text in the **Find...** dialog's **Replace with:** field; if there is no string in the **Replace with:** field, the highlighted text is deleted. **Replace and Find Again** will replace the highlighted text and search for the next occurrence of the **Search for:** string. **Replace All** replaces all occurrences of the sought string. **Find In Next File** is used in conjunction with the **Multi-File Search** check box; it continues the search with the next file on the list.

## The Source Menu

The **Source** menu (Figure C.10) deals with the project source code files. **Add** adds the front-most source code file to the project. If **Add** is dimmed, the current window has not been saved, or does not have the .c suffix. **Remove** removes a file, selected in the project window, from the project. **Get Info** provides a dialog that displays information about the current file, such as number of lines of code and data and string resources used. **Debug** sends the current source code file to the debugger's source code window. This option is available only when the debugger is running.

**Check Syntax** compiles a file to check syntax without adding object code to the project. **Preprocess** creates a new window showing the source code as it exists in memory after the first compile pass and

Source	
Add	
Add...	
Remove	
Get Info	
Debug	⌘I
<hr/>	
Check Syntax	⌘Y
Preprocess	
Disassemble	
<hr/>	
Precompile...	
Compile	⌘K
Load Library	
Make...	⌘\
<hr/>	
Browser	⌘J

Figure C.10 THINK C's **Source** menu.

all `#includes` and `#defines` have been applied. **Disassemble** translates a source code file into assembly language, placing the output in a new window.

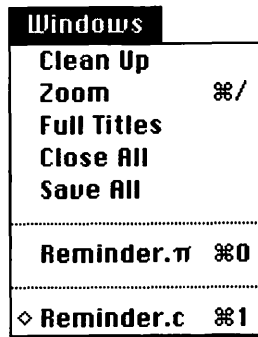
**Precompile...** rebuilds the selected precompiled header (such as `<MacHeaders>`). Although you have to have a project open for this option to be available, the precompiled header is changed for all projects, not just for the one currently open. **Compile** compiles the current source code file, saving the object code in the project file. **Load Library** loads the specified library's object code into the project. **Make...** allows you to specify dependencies between your project files instead of depending on THINK C's recompilation rules.

---

## The Windows Menu

---

The **Windows** menu (Figure C.11) controls all of THINK C's windows. **Clean Up** resizes and stacks currently opened windows. **Zoom** resizes the current window to fill the screen; if selected again, the window returns to the previous size. **Full Titles** puts the full path of each file in the window's title. **Close All** closes all edit windows. **Save All** saves all edit windows. The remaining items correspond to the project window (⌘O) and any other open windows. Selecting a window on the list brings it to the front.



**Figure C.11** THINK C's **Windows** menu.

## Changes to THINK C

Perhaps the most noticeable change from earlier versions is THINK C's tighter typechecking. A common complaint is that source code that compiles under THINK C 4 won't compile under THINK C 5, no matter how the options are set up. This is true. Here's why: THINK C 4 played fast and loose with typechecking. For example, this line of code:

```
window = GetNewWindow( 128, nil, -1L );
```

compiles just fine under THINK C 4 but yields the error last argument to function 'GetNewWindow' does not match prototype under THINK C 5. If you run into this type of error, you are trying to pass a value of one type in a parameter of another type. The solution? Use typecasting when mixing types. For example, this line:

```
window = GetNewWindow( 128, nil, (WindowPtr)-1L );
```

compiles without a hitch. You might say that THINK C forces you to write cleaner, more maintainable code.

Another major change is THINK C's commitment to and support of the current ANSI C standard. By selecting the appropriate THINK C options, you can force your code to adhere exactly to the ANSI standard.

This appendix is meant to provide only an overview of THINK C. For detailed information about THINK C, read the *THINK C User Manual* and *Standard Libraries Reference*.

## Appendix D







---

# The Debugger Command Summary

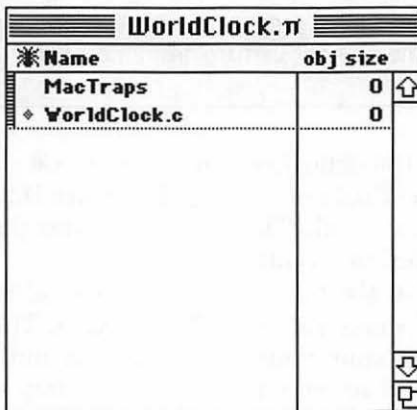
*This appendix uses Chapter 5's  
WorldClock program as the basis of a  
guided tour through the THINK C  
source level debugger. Fire up your  
Mac and follow along.*

THINK C's SOURCE-LEVEL debugger allows you to watch your application execute, routine by routine and line by line. To use the debugger, you'll need at least two megabytes of memory and you must be running under either System 7 or MultiFinder.

This programmer's tour of the debugger uses Chapter 3's Mondrian program as an example. If you'd like to follow along, open up your WorldClock folder and double-click on WorldClock.π. When THINK C starts up, select **Use Debugger** from the **Project** menu. A check mark should appear next to the **Use Debugger** item, and the project window should look like the one shown in Figure D.1.

Notice the  just under the left edge of the title bar, and the  just to the left of the source code file name WorldClock.c. The  tells you that the debugger is on. The  next to a file name tells you that debugging information is available for that file. Click on the  to turn it off and on again. You'll want the  next to any file you plan on debugging.

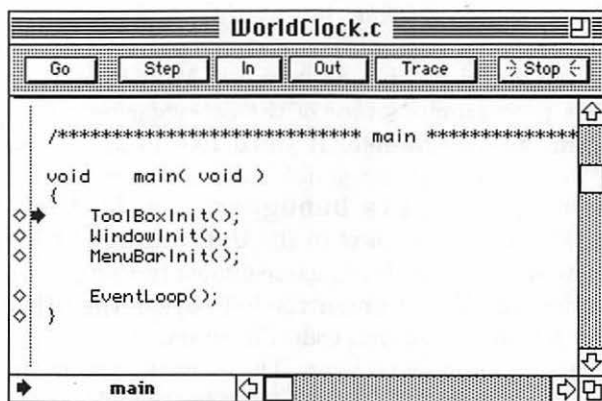
When you're ready to debug your software, select **Run** from the **Project** menu. Click **Yes** to bring the project up to date. If the program compiled successfully, the two debugger windows will appear.



**Figure D.1** Mondrian's project window with the debugger turned on.

## The Debugger's Source Window

The window on the left (in this case, labeled **WorldClock.π**) is the debugger's **source code window**. Using this window, you can scroll through your project's source code. Figure D.2 shows the source code for WorldClock.c.



**Figure D.2** The debugger's source code window.



If your project has more than one source code file, the source code window starts off with the file containing the function `main()`. To switch source code files, highlight a source code file name in the project window, then select **Debug** from the **Source** menu.

Once you enter the debugger, a dark arrow (➡) will appear to the left of the first line of code in `main()`. In Figure D.2, the arrow points to the `ToolBoxInit()` call. This arrow indicates the line of code that the debugger is about to execute.

The six buttons at the top of the source code window enable you to step through your source code in different ways. The **Go** button will start executing your program, continuing until a breakpoint is encountered (we'll get to those in a minute) or until the program exits. The **Step** button executes the line of code the dark arrow is pointing to, moving the arrow to the next executable statement. The **In** button executes code until a function call is encountered. The arrow stops at the first line of code inside that function. The **Out** button executes code until the current function exits, stopping at the first line of code outside the current function.

The **Trace** button is very similar to the **Step** button. A single statement is executed but **Trace** will fall into a function if it encounters one. **Step** will never fall into a function—it stays at the same level until it hits the end of the current function.

The diamonds (◆) to the left of the source code are used to set **breakpoints**. If you set a breakpoint next to a line of code, and then

press the  button, the debugger will execute until it hits that breakpoint. To set up a breakpoint, just click on the appropriate diamond. To clear a breakpoint, click on the diamond again. To set up a temporary breakpoint, hold down the command or option key when you click on a statement's diamond. The debugger will automatically start executing until it hits the breakpoint. Notice that you didn't have to hit the  button, and that the breakpoint was cleared once the break occurred.



One really nice feature of the debugger is that it automatically saves your debugging environment between runs. That means that you can set a bunch of breakpoints, open a whole series of data windows (we'll get to those in a minute), and when you quit the debugger, the breakpoints and data windows will be preserved exactly as you left them.

The  button halts execution, if at all possible. For the most part, your program must contain an event loop for the  button to work.

There is a debugger option in THINK C's **Options...** dialog box that allows you to turn "session saving" on and off. If you have this option turned on and you don't want the debugger to run with the last session intact, hold down the option key when you launch the debugger. If you have the option turned off and you want to save your current session, select **Save** from the debugger's **File** menu.

Back in the debugger, click on the  button. When WorldClock's time window appears, click on the debugger's source code window to bring it to the front, then click on the  button. Chances are you landed right next to a call to `WaitNextEvent()`.

To get to the debugger if the debugger windows are obscured, select **THINK C Debugger 5.0** from System 7's active application menu on the extreme right end of the menu bar.




The field in the lower left-hand corner of the source code window indicates the name of the currently executing function. If you click on this field, a pop-up menu will appear, listing the names of all the functions called to get from `main()` to where you are now. Selecting one of the function names from this menu jumps to that function in the source code window.

The button actions can also be selected from the debugger's **Debug** menu. The first six items correspond to the six buttons at the top of the source code window. Remember the **Debug** menu's command-key equivalents. They really come in handy.

The **Debug** menu item **Go Until Here** is the same as setting a temporary breakpoint. If you click on a line of source code and select **Go Until Here**, the debugger will run until that line of code is reached. **Skip To Here** allows you to skip execution of portions of your code. Be careful that whatever you are testing doesn't depend on the code you are skipping.

You can  or  continuously through your program if you click on these buttons with either the command or the option key depressed. To cancel the action, click on the  button or type `⌘-⌥-` (Command-Shift-period).

**Monitor** invokes the currently installed monitor (low-level debuggers like Macsbug or TMON). **ExitToShell** halts execution of your program and quits to THINK C.

Finally, check out the  menu's **Shortcuts...** item. A dialog box appears, allowing you to step through a list of really cool debugger shortcuts. Once you've got the debugger down, take the time to step through all of the shortcuts.

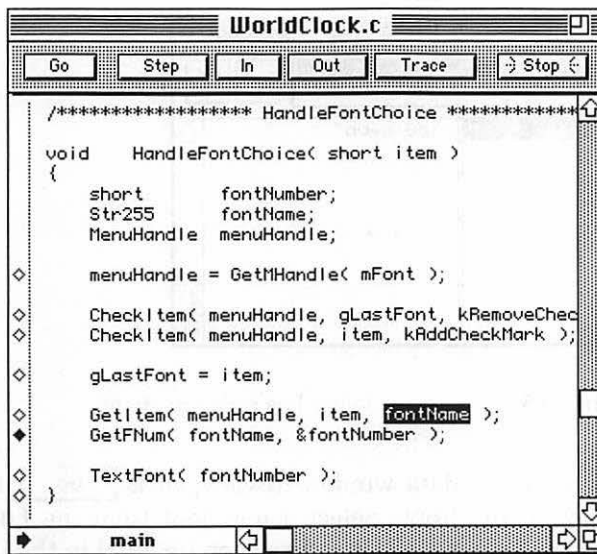
---

## The Debugger's Data Window

---

Along with the source code window, the debugger also opens up a **data window**. The data window lets you find out the values of your variables as the program runs. To use it, type a C expression in the data entry area just under the title bar, then press the ☒ button, or hit the enter or return key. The expression will appear in the scrolling list on the left side of the window, and the value of the expression will appear on the right side. Here's an example.

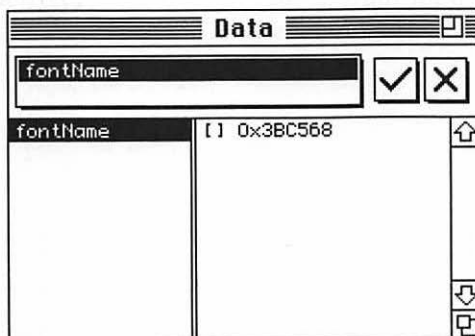
In the source code window, set a breakpoint in the routine `HandleFontChoice()`, just after the call of `GetItem()`. This call to `GetItem()` retrieves the name of the font just selected from the **Font** menu. The font's name is stored in the variable `fontName`. Figure D.3 shows our setup.



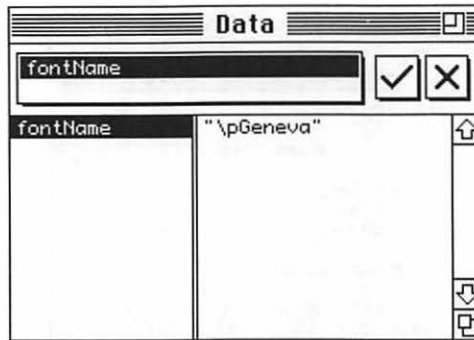
**Figure D.3** HandleFontChoice() with a single breakpoint set.

Now, click the **Go** button to jump to the breakpoint. Use the cursor to highlight the variable name `fontName`, as we did in Figure D.3. Go to the **Edit** menu and select **Copy To Data**. The variable `fontName` will appear on the left side of the data window and, after a brief delay, its value will appear on the right side (Figure D.4).

Notice that `fontName`'s value appears in hex. If you pull down the **Data** menu, you'll see that the **Address** item is checked. Since we want to look at the data as a Pascal string, go back to the **Data** menu and select **Pascal String**. The data will change from hex to pascal string format, as shown in Figure D.5.



**Figure D.4** The data window with the variable `fontName` copied to it.



**Figure D.5** `fontName` viewed as a Pascal string.

Keeping the data window in view, click **Go** to bring the clock window to the front. Select a new font from the **Font** menu. Notice that `fontName`'s new value has been updated in the Data window.

To clear a variable from the data window, click on the variable's name on the left side of the data window and select **Clear** from the **Edit** menu. To expand a struct, double-click on its name in the data window. The debugger will open up a new window, allowing you to track all the fields in the struct.

In the **Data** menu, you can also **Show Context**, which displays the context for an expression selected in the data window in the source window. If you select a line in the source window, you can then **Set Context** to change the context to the line highlighted in the source window. If you want to make sure that an expression doesn't change its value, select **Lock** in the **Data** menu with the desired variable highlighted.

This appendix is meant to provide an overview of the Debugger. For more detailed information, read Symantec's *THINK C User Manual* and *Standard Libraries Reference*.

## Appendix E

---

# Debugging Techniques

*One of the most frustrating experiences in programming is running up against a really tough bug. In this appendix, we'll discuss some techniques for hunting down bugs, and some others for avoiding them in the first place.*

## Compilation Errors

THE FIRST BUGS you're likely to encounter will pop up during compilation, when you've typed in your code and select **Run** from the **Project** menu. The first sign that something's amiss is the appearance of a bug alert, like the one shown in Figure E.1.



**Figure E.1** One of the more frustrating bug alerts.

This alert crops up a lot and seems frustratingly uninformative. Syntax errors are usually indicative of a misspelled keyword or bad programming grammar. For example, if you misspell `#define` or type something like:

```
EventRecord = theEvent;
```

instead of:

```
EventRecord      theEvent;
```

you'll end up with a syntax error. This happens frequently. Once you've clicked away the bug alert, carefully review the line of code with the blinking cursor in the left-hand column. If you still can't find the bug, check the previous line. Is there a semicolon at the end of the line? Is there supposed to be one?

Missing semicolons can cause several different types of bugs. For example, in Chapter 4's `EventTracker.c`, we took the semicolon away from the end of two different lines and got two different errors—an invalid declaration error and a missing semicolon error. C compilers can be tricky.

Another popular error message is the `xxx has not been declared` alert. Sometimes this is the result of a missing declaration, but often it's the result of a misspelled variable name. Remember, in C, upper and lower case are crucially different when it comes to identifiers. The variables `myPicture` and `MyPicture` are completely different. Check your case.

Another indicator of a typing error is the illegal character error message. Usually, these are pretty straightforward. For example, this line of code:

```
myVar $ = 27;
```

generates the message illegal character '\$'. That seems clear enough. This line of code, however:

```
myVar . = 27;
```

generates the classic syntax error message. Hmmm. So much for consistency.

## Indirect Compiler Errors

An indirect compiler error seems to point to one source but was caused by another. The classic example of an indirect compiler error is caused by a missing `#include` file. For example, Chapter 5's WorldClock program depends on the `#include` file `<GestaltEqu.h>`. This file is not one of the standard `#includes` automatically included by THINK C. If you leave out this `#include`, you'll get the error message `gestaltPopupAttr` has not been declared, and THINK C highlights the line:

```
myErr = Gestalt( gestaltPopupAttr, &feature );
```

This error seems pretty easy to figure out. Since you know that `gestaltPopupAttr` is not local to WorldClock, you suspect that you left out a `#include` file. If you scroll through the files in the Apple `#includes` folder, you'll spot the file associated with `Gestalt()` pretty quickly.

A more difficult item to track down is a variable or constant that doesn't necessarily reflect the `#include` file name in its name. For example, what file would you find `MAXLONG` defined in? As it turns out, `MAXLONG` is defined in `<Values.h>`. But how would you find that out?

For starters, check out the indexes in the *Inside Macintosh X-Ref (Revised Edition)*. If you can't find your reference in there, try using THINK Reference, or one of the other *Inside Macintosh* cross-reference tools. As a last resort, open up the files in the Apple `#includes` folder, one at a time, and use the THINK C search utilities.

Another indirect compiler error stems from not closing your comment blocks. For example:

```
/* my 1st comment    *  
int i;  
/* my 2nd comment    */  
i = 10;
```

This code will lead to an 'i' was not declared error. The declaration of `i` was swallowed up by the `my 1st comment` comment block, which was never closed.

## Linker Errors

If you call a procedure or function in your program that was never declared, you'll get a link failed error and a **Link Errors** window will appear, listing the routines that were called but that the linker couldn't locate. This error is often the result of a misspelled procedure name. For example:

```
sysBeeep( 20 );
```

The compiler will accept this line because it will assume that you've written a routine called `sysBeeep()` that will be provided at link time.

---

## Improving Your Debugging Technique

---

Once your program compiles, your next step is to get the bugs out. One of the best ways to debug a Mac program is to use a debugger like the THINK C debugger described in Appendix D, The Debugger by Steve Jasik, or the TMON debugger from ICOM Simulations. Debuggers are real lifesavers.

No matter which debugging tool you use, there are some things you can do to improve your debugging technique.

## Being a Good Detective

When your program crashes or exhibits some unusual behavior, you have to be a detective. Did the system error occur just before your dialog box was scheduled to appear? Did those wavy lines start appearing immediately after you clicked on the **OK** button?

The key to being a good detective is having a good surveillance technique. Try to establish a definite pattern in your program's misbehavior. Can you pinpoint exactly where in your code things started to go awry? These clues will help you home in on the offending code.

If you can't tell by observation exactly when things went sour, don't give up. You can always use the binary method of bug control.

## The Binary Method

The key to the binary method lies in establishing good boundary conditions for the bug. First, you'll need to establish a **lower limit**, a place in your code at which you feel fairly certain the bug has not yet occurred. It's best if the lower limit is as close to the actual bug as possible, but make sure the bug has not yet happened.

Next, establish an **upper limit** in your code, a point by which you're certain the bug has occurred (because the system has crashed, or the screen has turned green, or whatever).

To use the binary method, split the difference between the upper and lower limits. If the bug still has not occurred, split the difference again. Now, if the bug has occurred, you have a new upper limit. By repeating this procedure, you'll eventually locate the exact line of source code where the bug occurs. .

There are several different ways to split the difference between two lines of source code. If you're using a debugger, you can set a breakpoint halfway between the lines of code representing the upper and lower limits. Did you hit the breakpoint without encountering the bug? If so, set a new breakpoint, halfway between this one and the upper limit.

If you don't have a debugger, use a ROM call like `SysBeep()` to give you a clue. Did you hear the beep before the bug occurred? If so, put a new `SysBeep()` halfway between the old one and the upper limit. The nice thing about using `SysBeep()` is that it is reasonably nonintrusive, unlike putting up a new window and drawing some debugging information in it, which tends to interfere with your program's basic algorithm.

---

## Recommended Reading

---

In closing, we'd like to recommend some good reading material: your *THINK C User Manual*! The *User Manual* is a treasure trove of valuable tips for writing and debugging Mac programs. The more you know about the Macintosh and the THINK C development environment, the better you'll be at debugging your programs.



## Appendix F

---

---

# Building HyperCard XCMDs

*The introduction of HyperCard back in August 1987 caused quite a stir in the Macintosh world. A complete programming environment in its own right, HyperCard became even richer with the addition of XCMDs and XFCNs, the standalone code resources that allow you to access the raw power of THINK C from inside HyperCard.*

HYPERCARD comes with its own powerful programming language, **HyperTalk**. The designers of HyperTalk thoughtfully provided a mechanism for adding extensions to the HyperTalk command set. These extensions are code resources of type 'XCMD' and 'XFCN'.

XCMDs (pronounced as X-commands) take a parameter block as input from HyperCard, perform some calculations, put the results back into the parameter block, and return to the calling script. XFCNs (pronounced as X-functions) take the same parameter block as input, perform the same types of calculations, but return the results as a C or Pascal function would.

We've written an XCMD called `xFindApplication` that takes a four-character signature as input and returns `true` if an application with that signature is currently running, and returns `false` otherwise. A typical call of `xFindApplication` looks like this:

```
xFindApplication KAHL
Put the result into card field 1
```

We also created an XFCN called `fFindApplication` that performs the same service. A typical call of `fFindApplication` looks like this:

```
Put fFindApplication( KAHL ) into card field 1
```

The source code for `xFindApplication` and `fFindApplication` is identical. One is saved as an XCMD and the other as an XFCN. We've included the source code and project files, as well as a HyperCard test stack, on the *Mac Primer* source code disk (use the coupon on the last page).

---

## The `xFindApplication` XCMD

---

Create a new folder in your Development folder called `xFindApp`. Launch THINK C and create a project called `xFindApp.π` in the `xFindApp` folder. Add `MacTraps` to the project. Next, add `HyperXLib` to the project. You'll find `HyperXLib` in the same folder as `MacTraps`.

Select **New** from the **File** menu and type in the following source code:

```
#include "HyperXCmd.h"
#include "Processes.h"
```

```

/***** main *****/

pascal void main( XCmdPtr paramPtr )
{
    OSErr                err;
    ProcessSerialNumber   process;
    ProcessInfoRec        info;
    Boolean               appRunning=false;
    Str255                string;

    if ( paramPtr->params[0] == nil )
        SysBeep( 20 );
    else
    {
        process.highLongOfPSN = 0;
        process.lowLongOfPSN = kNoProcess;

        info.processInfoLength = sizeof
            ( ProcessInfoRec );

        info.processName = (StringPtr) NewPtr( 32 );
        info.processAppSpec = nil;

        while ( GetNextProcess( &process ) == noErr
            && !appRunning )
        {
            if ( GetProcessInformation( &process,
                &info ) == noErr )
                appRunning = info.processSignature ==
                    *(long *)*(paramPtr->params[0]) ;
        }
        BoolToStr( paramPtr, appRunning, string );
        paramPtr->returnValue = PasToZero
            ( paramPtr, string );
    }
}

```

Save the file as xFindApp.c and **Add** it to the project.

## Building the XCMD

Select **Set Project Type...** from the **Project** menu and click on the **Code Resource** radio button. Next, change the settings to match those found in Figure F.1. Make sure you change the resource ID to 128 and the resource type to XCMD. When you're done, click the **OK** button.

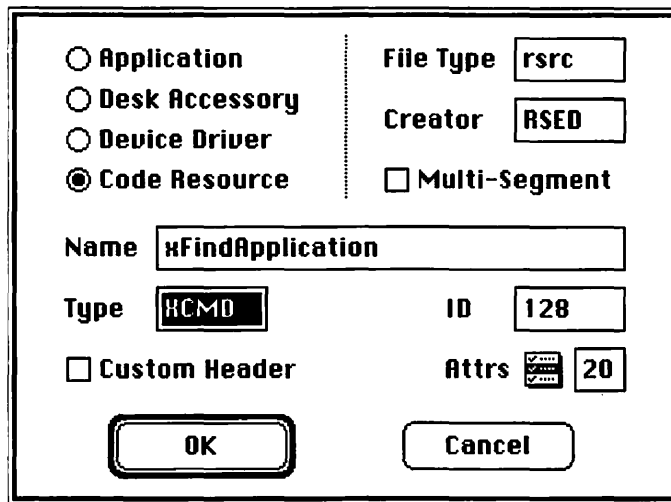


Figure F.1 The Set Project Type... dialog for the XCMD.

To create a file containing the XCMD resource, select **Build Code Resource...** from the **Project** menu. Click **Yes** when asked to update the project. Once the project compiles, a dialog will appear, asking you to name your new code resource file (Figure F.2). Save the code resource under the name `xFindApplication`. Finally, quit THINK C.

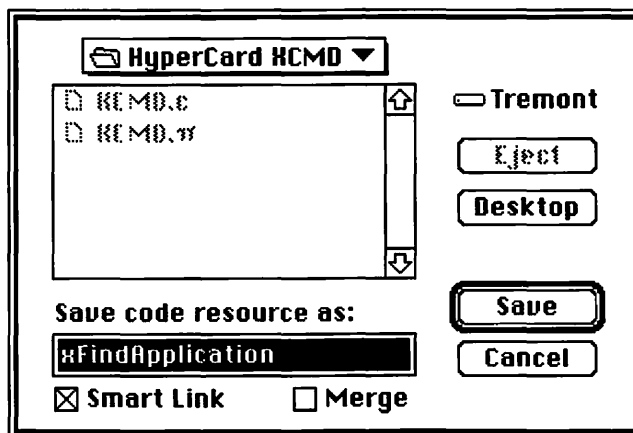
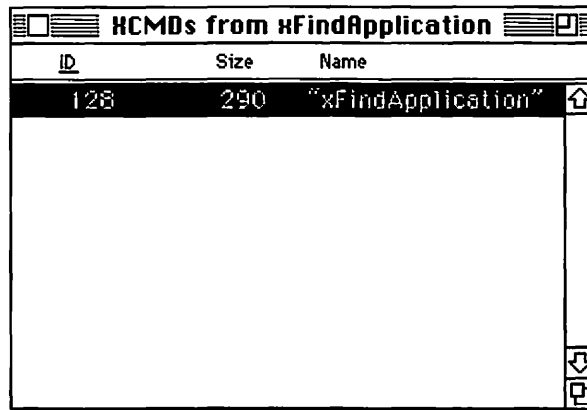


Figure F.2 The Build Code Resource... dialog.

## Adding Your XCMD to a Stack

Use HyperCard to create a stack to test your new XCMD. Once the stack is saved, quit HyperCard and launch ResEdit. Inside ResEdit, open the file containing your XCMD. When the main window appears, double-click on the XCMD icon. A window similar to the one shown in Figure F.3 should appear. Select the XCMD, then copy it to the clipboard. Next, open your test stack from within ResEdit and paste the XCMD. Quit ResEdit, saving your changes.



**Figure F.3** A ResEdit window, listing all XCMDs in the file xFindApplication.

You can also copy the XCMD directly into HyperCard, but work with a copy to stay on the safe side. XCMDs within the HyperCard application are accessible from all HyperCard stacks. An XCMD in a specific stack is available only within that stack.

Test out your stack by creating a field with an ID of 1 and a test button within your test stack. Enter this script in the button's script editing window:

```
on mouseUp
  xFindApplication KAHL
  put the result into card field 1
end mouseUp
```

Try out your new button. If THINK C is running when you press the button, the word true should appear in card field 1. If THINK C is not running, the word false should appear.

Let's take a look at the source code for this XCMD.

## Walking Through the Source Code

Whenever you write an XCMD or XFCN, you'll start by including the file `HyperXCmd.h`. `HyperXCmd.h` gives you access to the XCMD utilities found in the library `HyperXLib`. The include file `Processes.h` gives you access to the routines used by the Process Manager.

```
#include "HyperXCmd.h"
#include "Processes.h"
```

All XCMDs and XFCNs are declared this way. The single parameter is a pointer to a HyperCard parameter block. The parameter block contains the data passed into and returned from the XCMD.

```
/****** main *****/

pascal void main( XCmdPtr paramPtr )
{
    OSErr          err;
    ProcessSerialNumber  process;
    ProcessInfoRec  info;
    Boolean         appRunning = false;
    Str255          string;
```

At this point, the input parameter is hidden in the `params` field of the parameter block. `params` is an array of handles, one handle per parameter. Since we're expecting one parameter, we'll check the first handle to see if it has any data in it. If the handle is `nil`, we'll beep once. Otherwise, we'll go on with the Process Manager code.

```
    if ( paramPtr->params[0] == nil )
        SysBeep( 20 );
    else
    {
```

Calling `SysBeep()` from within an XCMD isn't particularly helpful. We did it here for illustrative purposes.

The next chunk of code is specific to the Process Manager. We build a process descriptor, passing it to the routine `GetNextProcess()`. `GetNextProcess()` steps through each running process, checking to see if the process's signature matches the signature passed in `param[ 0 ]`. If the signatures match, `appRunning` is set to `true` and the loop exits. The loop also exits once all processes have been checked.

```
process.highLongOfPSN = 0;
process.lowLongOfPSN = kNoProcess;

info.processInfoLength = sizeof
    ( ProcessInfoRec );

info.processName = (StringPtr) NewPtr( 32 );
info.processAppSpec = nil;

while ( GetNextProcess( &process ) == noErr
        && !appRunning )
{
    if ( GetProcessInformation( &process,
        &info ) == noErr )
        appRunning = info.processSignature ==
            *(long *)*(paramPtr->params[0]) ;
}
```

The routines `BoolToStr()` and `PasToZero()` are part of the HyperXLib library. `BoolToStr()` converts a Boolean value to a string format and `PasToZero()` converts a pascal string to a zero-terminated C string.

```
BoolToStr( paramPtr, appRunning, string );
paramPtr->returnValue = PasToZero( paramPtr,
    string );
}
```



## Getting More Information

---

If you plan on developing XCMDs, get the HyperCard Development Kit from Claris. You can reach Claris at (800) 325-2747 in the United States and at (800) 668-8948 in Canada. The Development Kit will fill you in on all the routines that are part of the HyperXLib library and will keep you up to date with the latest version of HyperCard.

To learn more about the Process Manager routines used in this appendix, read the Process Manager chapter in *Inside Macintosh*, Volume VI.



**Appendix G**

**Bibliography**

- Apple Computer, Inc. *Inside Macintosh*, Volume I. Reading, MA: Addison-Wesley, 1985. \$24.95.
- Apple Computer, Inc. *Inside Macintosh*, Volume II. Reading, MA: Addison-Wesley, 1985. \$24.95.
- Apple Computer, Inc. *Inside Macintosh*, Volume III. Reading, MA: Addison-Wesley, 1985. \$19.95.
- Apple Computer, Inc. *Inside Macintosh*, Volume IV. Reading, MA: Addison-Wesley, 1986. \$24.95.
- Apple Computer, Inc. *Inside Macintosh*, Volume V. Reading, MA: Addison-Wesley, 1988. \$26.95.
- Apple Computer, Inc. *Inside Macintosh*, Volume VI. Reading, MA: Addison-Wesley, 1991. \$39.95.
- Apple Computer, Inc. *Inside Macintosh X-Ref*, 2nd edition. Reading, MA: Addison-Wesley, 1988. \$9.95.
- Apple Computer, Inc. *Programmer's Introduction to the Macintosh Family*. Reading, MA: Addison-Wesley, 1988. \$22.95 (HC).
- Apple Computer, Inc. *Technical Introduction to the Macintosh Family*. Reading, MA: Addison-Wesley, 1987. \$19.95.
- Kernighan, Brian W., and Ritchie, Dennis M. *The C Programming Language*, 2nd edition. Englewood Cliffs, NJ: Prentice-Hall, 1988. \$29.95.
- Knaster, Scott. *How to Write Macintosh Software*, 3rd edition. Reading, MA: Addison-Wesley, 1992. \$28.95.
- Knaster, Scott. *Macintosh Programming Secrets*, 2nd edition. Reading, MA: Addison-Wesley, 1992. \$28.95.
- Mark, Dave. *Learn C on the Macintosh*. Reading, MA: Addison-Wesley, 1991. \$34.95.
- Mark, Dave. *Macintosh C Programming Primer*, Volume II. Reading, MA: Addison-Wesley, 1990. \$24.95.
- Mark, Dave, and Reed, Cartwright. *Macintosh Pascal Programming Primer*, Volume I. Reading, MA: Addison-Wesley, 1991. \$24.95.
- Smith, David E., ed. *The Best of MacTutor, The Macintosh Programming Journal*, Volume 1. Placentia, CA, 1985. \$24.95.
- Smith, David E., ed. *The Complete MacTutor, The Macintosh Programming Journal*, Volume 2. Placentia, CA, 1986. \$24.95.

## **Appendix H**

---

---

# **New Inside Macintosh Series**

*Apple Computer, Inc.  
Addison-Wesley Publishing Company*

# New Inside Macintosh Series

## Apple Computer, Inc.

### Addison-Wesley Publishing Company

---

#### **Books for All Programmers**

##### *Inside Macintosh: Overview*

Introduces New Inside Macintosh and provides an overview of Macintosh programming fundamentals. Topics include the Macintosh interface, compatibility guidelines, the event loop, resources, programming languages and tools, and international software considerations. Fall 1992.

#### **Toolbox Titles**

##### *Inside Macintosh: Macintosh Toolbox Essentials*

Describes how to implement essential user interface components in Macintosh applications. Covers the Control, Dialog, Event, Menu, and Window Managers, and a discussion of the Finder interface. Fall 1992.

##### *Inside Macintosh: More Macintosh Toolbox*

Covers the Help, List, Resource, Scrap, and Sound Managers, the Control Panel, and sound input. Winter 1992.

#### **Operating System Titles**

##### *Inside Macintosh: Files*

Covers all aspects of file handling. Detailed information on the File Manager, Alias Manager, and disk initialization. Fall 1992.

##### *Inside Macintosh: Processes*

Covers procedural functions such as starting up and shutting down, deferred tasks, and interrupts. Fall 1992.

##### *Inside Macintosh: Memory*

Covers all aspects of memory, including the Memory Manager, Virtual Memory Manager, and memory management utilities. Fall 1992.

##### *Inside Macintosh: Operating System Utilities*

Covers date and time, error handling, PRAM, and Toolbox utilities. Winter 1992.

##### *Inside Macintosh: Text*

Covers Font, Script, Dictionary, and Text Services Managers, QuickDraw Text, TextEdit, Keyboard Resources, and an extensive discussion of International Resources. Winter 1992.

*Inside Macintosh: Imaging*

Covers 32-bit QuickDraw, working with color, and picture utilities. Fall 1992.

*Inside Macintosh: Interapplication Communication*

Discusses in-depth collaborative computing, plus chapters describing the AppleEvent, and Edition Managers, and the PPC Toolbox. Winter 1992.

## **Special Topics**

### **QuickTime Titles**

*Inside Macintosh: QuickTime Movie Toolbox*

Describes all the QuickTime Toolbox utilities. Spring 1993.

*Inside Macintosh: QuickTime Components*

Describes how to use QuickTime components such as clock components, image compressors, movie controller, sequence grabbers, and video digitizers. Spring 1993.

### **Other**

*Inside Macintosh: Networking*

Describes how to write software that uses AppleTalk networking protocols. Spring 1993.

*Inside Macintosh: Communications*

Covers the Data Access Manager and Communications Toolbox. Spring 1993.

*Inside Macintosh: Devices*

Covers the Device, SCSI, Power, Component, Serial, and Slot Managers, writing a device driver, and the Apple Desktop Bus. Spring 1993.

---

---

# Index

- About boxes, 205, 212  
About Reminder... menu item, 328-329, 345  
Accessories, menus for, 240, 606  
activateEvt events, 131-133, 153, 181  
Active pop-up states, 207  
Add command, 612  
Add Files dialog box, 45-46  
AddResMenu() routine, 240-241  
AddResource() routine, 422  
Addresses with debugger, 621  
AECCountItems() routine, 485  
AECreatAppleEvent() routine, 190-191, 198-199  
AECreatDesc() routine, 190-191, 198  
AEGGetAttributeDesc() routine, 484  
AEGGetAttributePtr() routine, 484, 486-487  
AEGGetNthPtr() routine, 485-486  
AEGGetParamDesc() routine, 484  
AEInstallEventHandler() routine, 139-140, 142, 481, 483  
AEProcessAppleEvent() routine, 139, 141, 160  
AESend() routine, 190, 192, 198-199  
ALRT resources and alerts, 261, 267, 275-277  
    for OpenPICT, 430  
    for Pager, 383-384  
    for PrintPICT, 445  
    for Reminder, 303-305  
    for ShowClip, 407  
    for SoundMaker, 416  
altDBoxProc windows, 58-59  
America Online, 499, 501  
Ampersands (&) for parameters, 33-34  
AM/PM menu, 291-292, 295, 330, 358  
ANSI C standard, 614  
ANSI library, 46-47  
APPL file type, 462-463, 606  
Apple Event Registry, 199, 497  
Apple events, 138-142  
    and Finder, 480-481  
    handlers for, 139-142, 481  
    responding to, 482-489  
    sending. *See* EventTrigger program  
Apple menu, 205  
    for Reminder, 284, 286-287, 328-329  
    for WorldClock, 209, 212, 233  
Apple Programmers and Developers Association (APDA), 37, 496  
AppleLink, 497-498  
Applications  
    adding menus to, 208-209  
    Apple events in, 138-142  
    closing, 138, 158, 163  
    dialogs in, 268-275  
    exiting, 489  
    fonts for, 84  
    memory for, 283, 421  
    opening, 138, 158, 162  
    Process Manager for, 281-283  
    Project menu for, 606  
    signatures for, 460-461, 480  
    standalone. *See* Standalone applications  
    structure of, 68-69, 133-142  
    waiting for events in, 135-138  
    writing, 11-12  
appxEvt events, 132-133  
Archive files, 25-28  
Arcs, 64-65  
arrow global variable, 81  
Arrows with menu items, 205  
Artwork. *See* ShowPICT program  
Associate program, 497-498  
Attr suffix for selector codes, 159  
Attributes  
    Apple event, 484  
    for code resources, 607-608  
Auto-positioning dialogs, 305, 307, 368-369  
AutoExtractor window, 26  
autoKey events, 131, 154, 243  
Background  
    events for, 133, 257, 472  
    launching applications in, 283, 473  
    running WorldClock in, 257-258  
    sleep parameter for, 137, 393  
Balance command, 609  
Balloon help, 478-480, 500  
BalloonWriter program, 479  
Becoming active, 181  
BeginUpdate() routine, 161, 165, 182-183  
behind parameter for  
    GetNewWindow(), 60-61  
*Best of MacTutor, The*, 501  
Big Long Window Technique, 52-53  
Binary debugging method, 627-628  
Blanks in source code, 43  
BNDL resources, 459-460  
    editing, 461-464, 466-467  
    for stationery pads, 473  
Bold text style, 66, 85-86  
Boolean data type, 30  
BoolToStr() routine, 636  
Borders, 153, 177  
BottomRight() macro, 236, 248  
Breakpoints, 618-620, 628  
Bring Up to Date command, 608  
Build Application... command, 468, 608  
Build Code Resource... command, 633  
Build Library... command, 608  
Button() routine, 30-31

- Button tool, 296
- Buttons, 264-265, 270, 272, 296
  - creating, 296-299
  - position of, 277
- Buttons, cancel
  - creating, 298-299
  - position of, 277
- C Libraries folder, 45-46
- C strings, 32
- Call-back functions, 401
- Calling sequences, 30-31, 33
- Case-sensitivity
  - errors from, 625
  - Pascal versus C, 31
  - for resources, 38
- Caution alerts, 275-277
- CautionAlert() routine, 276-277
- CD-ROM drives for developers, 497
- cdevs, 606
- CenterPict() routine
  - for OpenPict, 435, 443
  - for Pager, 391, 403
  - for PrintPict, 448-449, 454-455
  - for ShowClip, 410, 415
  - for ShowPict, 108, 111-112
  - for Updater, 176, 189-190
- ChangedResource() routine, 375
- Check boxes, 263-264
  - creating, 300-302
  - initializing, 271
- Check Link command, 608
- Check marks, 204, 216, 233, 253
- Check Syntax command, 612
- CheckItem() routine, 251, 253
- CIS Navigator, 499
- Classes
  - of events, 142
  - libraries for, 10, 27
- Clean Up command, 613
- Clear command (Debugger), 622
- Clear command (Edit), 609
- ClearHighByte() macro, 237
- Clipboard. *See* Scrap Manager; ShowClip program
- Clipping
  - drawings, 188-189
  - lines, 64
  - in Pager, 401-403
- ClipRect() routine, 189, 402
- Close All command, 613
- Close & Compact command, 605
- Close boxes, 56
  - clicks in, 166, 185
  - in frames, 153, 177
- Close command, 608
- Close Project command, 605
- CloseDown() routine, 489
- Closing
  - applications, 138, 158, 163
  - files, 429, 608
  - projects, 605
  - windows, 56, 166, 185
- CMaster customizer, 501
- CNTL resources, 208-209
  - for Reminder, 293-295
  - for WorldClock, 218-219, 241
- Code and CODE resources, 40, 42. *See also* Source code
  - disassemblers for, 500
  - Project menu for, 606-607
- Code Optimization options, 609
- Color
  - global variable for, 93
  - for icons, 465-466
  - for Mondrian, 102-103
  - predefined, 103
- Color QuickDraw, 103
- Command-based interfaces, 5
- Command-key equivalents, 204-205, 214
- Comments, errors from, 626-627
- Compatibility, 7, 19
- Compile command and compilation, 10, 31, 613
  - errors in, debugging, 625-627
  - of Hello, World, 44
  - resources for, 40
- Compiler Settings options, 609
- Compressed files, 25-28
- CompuServe Information Service, 498-499, 501
- ConcatString() routine, 326, 358-359
- Condensed text style, 85-86
- Content regions, 97, 185
- Control Manager, 218, 263
- Controls, 218, 262-266
  - dimmed, 273
  - drawing, 255
  - initializing, 269-271
  - for pop-up menus, 299-300
  - scroll bars, 379-382
- Conventions in C, 35-36
- Coordinate systems, 51-56, 62, 248
- Copy button, 611
- Copy command, 406, 609
- Copy To Data command, 621
- CopyDialogToReminder() routine, 324-326, 355-358
- Copying
  - to desk scrap, 406
  - icons, 465-466
  - settings, 611
- Core event classes, 142
- Count suffix for selector codes, 159
- Count1Resources() routine, 396
- CountRemindersOnMenu() routine, 319, 343, 348-349
- CountResources() routine, 396
- CreateWindow() routine, 482-483
- Creator IDs, 461
- Cross-reference tools, 626
- CToPstr() routine, 32
- Curly braces ({}), 35
- Current windows, 63
- Cursors
  - for dialogs, 272
  - initializing, 80
  - shape of, 81
  - tracking, 137, 160
- Cut command, 406, 609
- Data forks, 40, 500
- Data links, 480
- Data menu, 621-622
- Data types, 32, 34
  - Pascal versus C, 30
  - typecasting, 61, 614
- Data window for debugger, 620-622
- dBoxProc windows, 58-59
- Debug command, 612
- Debug menu, 620
- Debugging, The, 627
- Debugging, 9-10, 500
  - compilation errors, 625-627
  - data window for, 620-622
  - and naming standards, 36
  - options for, 609
  - source window for, 617-620
  - techniques in, 627-628
- Declarations, errors in, 625
- Default items in dialogs, 272, 306, 353, 374, 377
- #define statements
  - for dialog boxes, 269
  - for EventTracker, 144, 154
  - for EventTrigger, 193, 197
  - for FlyingLine, 114, 119
  - for Hello2, 76, 79
  - literals in, 31
  - for Mondrian, 89, 93
  - names for, 36, 236
  - for OpenPict, 430-431, 437
  - for Pager, 385-386, 393-394
  - for PrintPict, 445, 450
  - for Reminder, 309-310, 335-336
  - for ResWriter, 369, 373-374
  - for ShowClip, 407-408, 411-412
  - for ShowPict, 106, 109
  - for SoundMaker, 417, 420-421
  - for Updater, 170, 177
  - for WorldClock, 221-222, 235-236
- DeleteReminder() routine, 321, 344, 350-351
- DeleteReminderFromMenu() routine, 320, 342, 349-350
- Dequeue() routine, 338, 342, 351
- Descriptors, 190-191, 484-485, 487
- Desk accessories, menus for, 240, 606
- Desk scrap. *See* Scrap Manager; ShowClip program
- Developer CD Series, 497
- Developer Programs Hotline, 498
- Developer Technical Support, 461
- Development environments, 499
- Device drivers, Project menu for, 606

- Dialog Manager, initializing, 80
- Dialogs, 261-262
  - adding, to programs, 268-275
  - alerts, 261, 267, 275-277
  - controls for, 262-266
  - filters with, 277, 422
  - item lists for, 262
  - memory for, 356
  - modal, 267-268
  - modeless, 267-268
  - position of, 305, 307, 368-369
  - for Reminder, 293-295, 352-358
  - resources for, 268-269
  - visibility of, 272
  - windows for, 58-59
- DialogSelect() routine, 268
- Dials, 265-266
- Diamonds
  - for breakpoints, 618
  - for notifications, 278-280
- Dimmed controls, 273, 275
- Dimmed menu items, 204
- Directories
  - C Libraries, 45-46
  - in HFS, 428
  - for source code, 25
- Disabled dialog items, 273, 275
- Disabled menu items, 204
- Disassemble command, 613
- Disassemblers, 600
- Disk for desk scrap, 405
- diskEvt events, 131, 154
- DisposDialog() routine, 356, 379
- DisposePtr() routine, 351
- DisposeReminder() routine, 321, 342, 351
- DisposHandle() routine, 422
- DITL resources, 262, 268-270, 276
  - for OpenPICT, 430
  - for Pager, 383-384
  - for PrintPICT, 445
  - for Reminder, 295-303, 352
  - for ResWriter, 367-368
  - for ShowClip, 407
  - for SoundMaker, 416
- DLOG resources, 39, 262, 268-269, 272
  - for Reminder, 305-307, 352
  - for ResWriter, 368-369, 376
- DoActivate() routine, 175, 184
- DoAError() routine, 484-485
- Document windows, 57
- documentProc windows, 57, 59
- Documents
  - opening, 138, 158, 162
  - printing, 138, 158, 163
- DoDialogs() routine, 447, 451, 453
- DoError() routine
  - for OpenPICT, 430, 435, 438, 440, 443
  - for Pager, 392, 394-396, 403-404
  - for PrintPICT, 449, 452, 454-455
  - for ShowClip, 410, 415
  - for SoundMaker, 416, 419, 422-424
- DoEvent() routine, 134-135, 137-138, 141
  - for EventTracker, 148-149, 158, 160-162
  - for Pager, 389, 399
  - for Reminder, 313-314, 341
  - for Updater, 173, 180
  - for WorldClock, 225, 242-243
- DoOpenApp() routine, 149, 158, 162, 482-484
- DoOpenDoc() routine, 149, 158, 162, 482-487
- DoPicture() routine, 175-176, 182, 188
- DoPrintDoc() routine, 150, 158, 163, 482, 487-488
- DoQuitApp() routine, 150, 158, 163, 482, 489
- DoTextDialog() routine, 371-372, 374, 376-379
- DoUpdate() routine
  - for Updater, 175, 181-182
  - for WorldClock, 231-232, 239, 244, 254-255
- Drag regions, 56-57
  - clicks in, 166, 185
  - in frames, 153, 177
- Dragging windows, 246
- DragWindow() routine, 166, 246
- DrawControls() routine, 255, 381, 399
- DrawEventString() routine, 150, 157-158, 161-163
- DrawGrowIcon() routine, 184, 189
- Drawings. *See* QuickDraw
- DrawLine() routine, 124, 126
- DrawMenuBar() routine, 241
- DrawMyPicture() routine
  - for OpenPICT, 435, 442-443
  - for ShowPICT, 107, 109-110
- DrawPicture() routine, 111
  - for PrintPICT, 454
  - for Updater, 189
- DrawRandomRect() routine, 91, 96, 99-100, 102-103
- DrawString() routine, 32, 83, 162, 165
- driverEvt events, 132
- Drop shadows, 207-208
- DRVr resources and type, 240, 606
- Edit menu, 609-611
  - with Debugger, 621-622
  - with Reminder, 284, 288-289, 330
  - with WorldClock, 209, 212, 214, 233
- Editable text fields, 273-275, 300-301
- Editing. *See also* ResEdit
  - BNDL resources, 461-464, 466-467
  - data forks, 600
- Edition Manager, 480
- Ellipses (...), 261
- Enabled dialog items, 273
- Enabled menus, 212, 214
- EndUpdate() routine, 161, 182-183
- Enqueue() routine, 338
- Enter Selection option, 612
- EraseRect() routine, 189
- Errors, 261, 267, 275-277. *See also* Debugging
- Event Manager, 129-133
- EventInit() routine, 140, 146-147, 155, 157-159
- EventLoop() routine, 134-135, 141
  - for EventTracker, 147-148, 155, 158-160
  - for Pager, 389, 398
  - for Reminder, 313, 340
  - for Updater, 172, 178, 180
  - for WorldClock, 225, 242
- EventRecord structure, 129, 136
- Events, 11, 14, 129
  - Apple. *See* Apple events;  
EventTrigger program
  - classes of, 142
  - high-level and low-level, 139, 153, 473
  - IDs for, 142, 484
  - masks for, 135-136
  - and program structure, 133-134
  - types of, 130-133
  - waiting for, 135-138
- EventsInit() routine, 194, 198
- EventTracker program, 14
  - building, 167-168
  - resources for, 143-144
  - running, 151-154
  - source code for, 144-151, 154-165, 534-541
- EventTrigger program, 15-16, 190-192
  - running, 195-196
  - source code for, 193-199, 481-482, 547-549
- everyEvent constant, 135
- ExitToShell command, 620
- ExitToShell() routine, 82
- Expressions with debugger, 620
- Extended text style, 85-86
- extern statements, 31-32
- Factory Settings button, 611
- Families of icons, 464-467
- Far CODE check box, 606
- Far DATA check box, 606
- feature parameter for Gestalt(), 157, 159
- FIFO (First In, First Out) queues, 130
- File Manager, 427-429
- File menu, 608-609
  - with Reminder, 284, 288, 328-329
  - with WorldClock, 209, 212, 214, 233
- File reference numbers, 429
- File system specification, 429, 439, 486
- Files, 424. *See also* OpenPICT program
- archive, 25-28
- closing, 429, 608
- File Manager for, 427-429
- names for, 426, 428
- opening, 429, 608
- project, 10, 25



- size of, 429, 440-441
- Standard File Package for, 424-427
- types of, 462-463, 474, 606
- version information for, 476-477
- FillRect()** routine, 122-123
- Filters**
  - with dialogs, 277, 422
  - with file lists, 425-426
  - with sending events, 192
- Find Again** option, 612
- Find...** command, 611
- FindControl()** routine, 245-246, 381, 400
- Finder**
  - and Apple events, 480-481
  - flags with, 282-283, 474-475
  - information with, 475-476
  - resources for, 459-460, 463, 471-478
  - testing database of, 470-471
- FindReminderOnMenu()** routine, 318, 344, 346-347
- FindReminderToDispose()** routine, 318-319, 347-348
- FindReminderToPost()** routine, 318, 341-342, 347
- FindWindow()** routine
  - for EventTracker, 165
  - for scroll bars, 381
  - for Updater, 184
  - for WorldClock, 245
- FIInfo** structure, 475-476
- Flags**
  - Finder, 282-283, 474-475
  - SIZE, 152, 328
- Flat** file format, 427-428
- FlyingLine** program, 13-14, 113
  - running, 118
  - source code for, 114-125, 529-534
- Folders**
  - C Libraries, 45-46
  - in HFS, 428
  - for source code, 25
- Font Manager**, 80, 87
- Font** menu, 212, 215-216, 233, 240-241, 251-252
- Fonts**, 65-66
  - for applications, 84
  - for code, 20
  - for Hello2, 83-85, 87
  - for menu text, 203
  - for projects, 42
  - for ShowClip, 414
  - TrueType, 87
- ForeColor()** routine, 96
- Forks**, 40, 393, 500
- Formatted** text on desk scrap, 414
- FrameArc()** routine, 101
- FrameRoundRect()** routine, 99
- Frames** for windows, 153, 177
- FREF** resource, 463, 466, 473
- FrontWindow()** routine
  - for Mondrian, 96
  - for ShowClip, 413
  - for ShowPICT, 110
  - for WorldClock, 241
- FSClose()** routine, 429
- FSGetFile()** routine, 429
- FSpOpenDF()** routine, 429, 440
- FSRead()** routine, 429, 440-441
- FSSpec**, 429, 439, 486
- Full Titles** command, 613
- Functions**, 30
  - for EventTracker, 145, 155
  - for EventTrigger, 193, 197
  - for FlyingLine, 114, 120
  - for Hello2, 76
  - for Mondrian, 90, 94
  - names for, 36
  - for OpenPICT, 431, 437
  - for Pager, 386, 394
  - for PrintPICT, 446, 451
  - prototypes for, 79
  - for Reminder, 310-311, 337-338
  - for ResWriter, 370, 374
  - for ShowClip, 408, 412
  - for ShowPICT, 106, 109
  - for SoundMaker, 417, 421
  - for Updater, 171, 178
  - for WorldClock, 222, 237
- General Control Panel**, 256
- Gestalt()** routine
  - for EventTracker, 154, 157
  - for EventTrigger, 196, 198
  - for OpenPICT, 436, 438
  - selector codes for, 159
  - for SoundMaker, 420-421
  - for WorldClock, 235, 241
- Get Info** command, 469, 612
- GetIndResource()** routine, 396
- GetClip()** routine, 188, 401-402
- GetCtlValue()** routine, 246
- GetDateTime()** routine, 95, 254
- GetDItem()** routine, 269-271
  - for Reminder, 353, 356-357
  - for ResWriter, 377
- GetEOF()** routine, 429, 440
- GetFileName()** routine
  - for OpenPICT, 432, 437-439
  - for Reminder, 321-322, 351-352, 354
- GetFirstReminder()** routine, 317, 346
- GetFNum()** routine
  - calling sequence for, 33
  - for fonts, 84
  - for WorldClock, 252
- GetFontName()** routine, 34
- GetIndResource()** routine, 396, 402
- GetItem()** routine, 251-252
- GetIText()** routine, 273-275, 353, 379
- GetMenu()** routine, 240, 343
- GetMHandle()** routine, 240
- GetNewControl()** routine, 241
- GetNewDialog()** routine, 267, 269
  - for Reminder, 352
  - for ResWriter, 376-377
- GetNewMBar()** routine, 239
- GetNewWindow()** routine, 7
  - calling sequence for, 30-31
  - for FlyingLine, 122
  - for Hello2, 73, 82
  - for Mondrian, 95
  - for WIND resources, 60-61
- GetNextEvent()** routine, 137
- GetNextProcess()** routine, 636
- GetNextReminder()** routine, 317, 346
- GetPicture()** routine, 111, 113
  - for PrintPICT, 452
  - for Updater, 182
- GetReminderFromNotification()** routine, 327, 360
- GetResource()** routine, 274
- GetScrap()** routine, 404-406, 413-414
- GetString()** routine, 274, 375
- GetWRefCon()** routine, 179, 182
- GetZoneOffset()** routine, 232-233, 254-257
- gFillColor** global variable, 93, 96, 102
- Global** coordinate system, 52-54, 248
- Global** variables, 81, 121
  - for EventTracker, 145, 155
  - for FlyingLine, 114, 120
  - for Mondrian, 89, 93-94
  - naming, 36
  - for Pager, 386, 394
  - for Reminder, 312, 338
  - for Updater, 170, 178
  - for WorldClock, 222
- Glossary**, 503-520
- Go-away** boxes, 56
  - clicks in, 166, 185
  - in frames, 153, 177
- Go** button with debugger, 618-619
- Go Until Here** command, 620
- GotRequiredParams()** routine, 485
- GrafPorts**, 63
  - global variables for, 81
  - pointers to, 247
  - for printing, 444-445, 454
  - saving, 188
  - scrolling, 163-165
- Graphical User Interfaces (GUI)**, 3-5
- Grep** check box, 612
- Grid**, 51-53
- Grow** boxes, 57, 59, 184
- GrowWindow()** routine, 185-187
- HandleAppleChoice()** routine
  - for Reminder, 316-317, 344-345
  - for WorldClock, 229, 250-251
- HandleDialog()** routine, 322-324, 345, 352-356
- HandleFileChoice()** routine
  - for Reminder, 317, 344-346
  - for WorldClock, 229, 251
- HandleFontChoice()** routine, 229-230, 241, 251-252
- HandleKeyDown()** routine, 137

- HandleMenuChoice() routine
  - for Reminder, 315-316, 344
  - for WorldClock, 228-229, 245, 249-250
- HandleMouseDown() routine, 137, 139
  - for EventTracker, 150-151, 160, 165-166
  - for Pager, 390, 399-401
  - for Reminder, 315, 341-343
  - for Updater, 173-174, 184-190
  - for WorldClock, 226-227, 243-247
- HandleNull() routine
  - for Reminder, 314, 341-342
  - for WorldClock, 226, 242, 244
- Handles, 113, 248, 379
  - for desk scrap, 405, 413
  - for memory, 112, 378-379
  - for menu data, 240
  - for print records, 453
  - for regions, 164
- HandleStyleChoice() routine, 230-231, 252-254
- Hard drive requirements, 20, 25
- Hardware, information on, 157
- Headers
  - for CODE resources, 607
  - for PICT files, 440-441
- Headers & Libs.sea archive, 26
- Hello, World program
  - creating, 40-42
  - running, 43-48
  - source code for, 43, 523
- Hello2 program, 69
  - project file for, 75
  - resources for, 70-75
  - running, 78
  - signature for, 461
  - source code for, 76-77, 79-83, 523-524
  - variants of, 83-88
- HELLO resources, 466
- Help and Help Manager, 478-480
  - editing, 500
  - for Options command, 609
  - resources for, 479
- HideWindow() routine, 61
- Hierarchical File System (HFS), 428
- Hierarchical menus, 203, 206, 215, 240
- High-level events, 139, 153, 473
- Highlighted menu items, 204
- HiliteControl() routine, 273, 353, 373
- HiliteMenu() routine, 250
- HiWord() routine, 249
- HLock() routine, 377, 414
- Hours menu, 289-290, 293, 330
- How to Write Macintosh Software, Third Edition* (Knaster), 497
- HyperCard, 10, 18
  - buttons in, 264
  - forks for, 40
  - XCMDs for, 631-637
- HyperTalk, 631
- HyperXCmd.h file, 635
- HyperXLib file, 635
- Icon edit panel, 464-465
- Icons
  - color for, 465-466
  - in dialog boxes, 266
  - families of, 464-467
  - masks for, 465
  - with menu items, 204
  - for notifications, 278-281, 331
  - resources for, 307-308, 333, 356, 464
  - for standalone applications, 463-471
  - with stationery pads, 473
  - testing, 467-468
- ICxx icon resources, 464
- Identifiers, errors with, 625
- Ignore Case option, 612
- Illegal characters, 626
- In button with debugger, 618
- Inactive pop-up states, 207
- #include statements, 31
  - errors from, 626
  - for EventTracker, 144, 154
  - for EventTrigger, 193, 196-197
  - for OpenPICT, 430, 436
  - for Pager, 385, 393
  - for PrintPICT, 445, 450
  - for Reminder, 309, 335
  - for SoundMaker, 417, 420
  - for Updater, 170, 177
  - for WorldClock, 221, 235
- inContent part code, 185
- Indentation style, 35
- Indirect compiler errors, 626-627
- inDrag part code, 166, 185
- InfoScrap() routine, 404-405
- InfoWorld magazine, 493
- inGoAway part code, 166, 185
- InitCursor() routine, 80
- InitDialogs() routine, 80
- InitFonts() routine, 80
- InitGraf() routine, 80
- InitMenus() routine, 80-81
- INITs, 606
- InitWindows() routine, 80-81
- InsertMenu() routine, 240
- InsertReminderIntoMenu() routine, 319, 348
- Inside Macintosh*, 8, 20, 29, 493-496, 645-646
- Inside Macintosh X-Ref*, 626
- Installing Think C, 25-29
- inSysWindow part code, 166, 184-185, 343
- Intelligence At Large, 501
- Interfaces, 3-5
- International Resources, 254
- International Utilities Package, 255
- InvalidRect() routine
  - for Updater, 186-188
  - for WorldClock, 244
- InvertArc() routine, 101
- InvertRoundRect() routine, 100
- IsDialogEvent() routine, 268
- IsHighBitSet() macro, 237
- isStationery flag, 474
- Italic text style, 66, 85-86
- IUTimeString() routine, 235, 254
- Job dialog box, 261, 450, 453
- kAEOpenApplication event, 138, 142, 153, 158, 480-483
- kAEOpenDocuments event, 138, 142, 158, 461, 471, 480-487
- kAEPrintDocuments event, 138, 142, 158, 191, 481-482, 487-488
- kAEQuitApplication event, 138, 142, 158, 481-482, 489
- kCoreEventClass constant, 142, 191
- Keyboard, repeat rate for, 131, 154, 243
- keyDown events, 130, 243
- KeyRepThresh global variable, 131
- KeyThresh global variable, 131
- keyUp events, 131
- kHighLevelEvent constant, 139
- Labels for pop-up menus, 218-219
- Language Settings options, 609
- LaunchApplication() routine, 281-283
- Launching applications with Reminder, 334
- LaunchParamBlockRec structure, 282
- LaunchResponse() routine, 327, 357, 359
- Learn C on the Macintosh* (Mark), 8
- Length byte in Pascal strings, 32
- Libraries, 10, 27, 45-47
- Lines, drawing, 63-64
- LinesInit() routine, 116, 120, 123
- LineTo() routine, 63-64
- Link Errors window, 44-45
- "Link failed" dialog box, 44
- Linking
  - data, 480
  - object files, 44-45, 627
- Literal parameters, 31
- Load Library command, 613
- LoadPICT() routine, 447, 451
- LoadPICTFile() routine, 433-434, 437-442
- LoadScrap() routine, 404-405
- Local coordinate system, 53-55, 62, 248
- LocalToGlobal() routine, 248
- Lock command, 622
- Low-level events, 139
- Lower limits in debugging, 628
- LoWord() routine, 249
- Mac Programming 101* series, 501
- Macintosh Developer Technical Support (MacDTS), 498
- Macintosh File System (MFS), 427-428
- Macintosh Inside Out* series, 497

- Macintosh interface, 4-5
- Macintosh Programmer's Workshop (MPW), 9, 499
- Macintosh Revealed*, 8
- Macintosh Technical Notes*, 20, 496-498
- Macintosh Toolbox. *See* Toolbox
- MacNosy debugger, 500
- MacsBug debugger, 500
- MacTraps file, 77
- MacTutor* magazine, 493, 501
- MacUser* magazine, 493
- MacWorld* magazine, 493
- Main event loops, 135
- main() routine, 68, 134-135
  - for EventTracker, 145, 155
  - for EventTrigger, 193-194, 197
  - for FlyingLine, 114, 120
  - for Hello2, 76, 79-80
  - for Mondrian, 90, 94
  - for OpenPICT, 431, 437-438
  - for Pager, 386, 394-395
  - for PrintPICT, 446, 451
  - for Reminder, 312, 338-339
  - for ResWriter, 370, 375
  - for ShowClip, 408, 412
  - for ShowPICT, 106, 109
  - for SoundMaker, 417-418, 421-422
  - for Updater, 171, 178
  - for WorldClock, 223, 238
  - for XCMD, 632, 635
- Main text, 19
- MainLoop() routine
  - for FlyingLine, 116, 123-124
  - for Mondrian, 91, 95-96, 102-103
  - for ShowClip, 409-410, 413-415
- Make... command, 613
- Managers, 6
- Map Control Panel, 234-235
- Masks
  - for events, 135-136
  - for icons, 465
- MaxApplZone() routine, 421
- MAXLONG constant, 137, 393
- MBAR resources, 208
  - for Reminder, 285-286
  - for WorldClock, 210-211, 239
- MBarHeight global variable, 121
- Memory
  - for applications, 283, 421
  - for desk scrap, 404-406
  - for dialogs, 356
  - for editing resources, 37
  - handles for, 112, 378-379
  - for PICT files and resources, 182, 441
  - for regions, 188
  - requirements for, 20
  - for windows, 60, 74, 82
- Menu bars, 203-204
  - drawing, 81
  - rectangles with, 122
- Menu lists, 239
- Menu Manager, 80-81
- MENU resources, 38, 208
  - for Reminder, 286-293
  - for WorldClock, 212-218
- MenuBarInit() routine
  - for Reminder, 312-313, 339-340
  - for WorldClock, 224, 239-242
- MenuKey() routine, 243
- Menus. *See also* WorldClock program
  - adding, to programs, 208-209
  - components of, 203-205
  - hierarchical, 203, 206, 215, 240
  - items on, 204
  - pop-up. *See* Pop-up menus
  - THINK C, 605-614
  - titles for, 204, 208, 218-219
- MenuSelect() routine, 243, 245
- message field for WaitNextEvent(), 136
- Message string resources, 477
- Minutes menu, 290-291, 294, 330
- Modal dialogs, 267-268
- ModalDialog() routine, 267-269, 272-273, 353
  - for ResWriter, 378
  - for SoundMaker, 422
- Modeless dialogs, 267-268
- Modes
  - file access, 429
  - text, 66-67
- modifiers field for WaitNextEvent(), 136
- Mondrian program, 13, 64-65
  - resources for, 88-89
  - running, 92-93
  - source code for, 89-98, 524-527
  - variants of, 98-103
- Monitor command, 620
- mouseDown events, 130, 137, 139
  - for EventTracker, 150-151, 154, 160, 165-166
  - for Pager, 390, 399-401
  - for Reminder, 315, 341-343
  - for scroll bars, 381
  - and SIZE resource, 473
  - for Updater, 173-174, 184-190
  - for WorldClock, 226-227, 243-247
- mouseMoved events, 133, 137
- mouseRgn parameter for
  - WaitNextEvent(), 137
- mouseUp events, 130
  - with EventTracker, 154
  - and SIZE resource, 473
- Movable modal dialog boxes, 268
- MoveTo() routine, 62, 84, 165
- Moving windows, 55
- Multi-File Search option, 612
- Multi-Segment check box, 606
- MultiFinder-Aware flag, 472
- Name Project dialog box, 41
- Name string resources, 477-478
- Names
  - for accessories and device drivers, 606
  - for #define statements, 236
  - for files, 426, 428
  - for projects, 41, 70
  - for resources, 39, 42, 74
  - for source code, 10, 42
  - for variables and functions, 36
- networkEvt events, 132
- New command, 608
- New Project... command, 605
- New Projects button, 611
- NewControl() routine, 379-380, 393, 396
- NewHandle() routine, 413, 444
- NewRgn() routine, 188
- NewWindow() routine, 119, 122, 268
- NMInstall() routine, 279-280, 350
- NMRec structure, 280-281
- NMRemove() routine, 280, 360
- noGrowDocProc windows, 57-59
- NormalResponse() routine, 326, 357, 359
- Note alerts, 275-277
- NoteAlert() routine, 276-277, 345
- Notification Manager, 278-281
- Null events, 130, 160
  - background, 257, 472
  - masking, 135
  - for WorldClock, 242
- Object code
  - linking, 44
  - in project files, 605
  - removing, 608
  - resources for, 42
- OK buttons, 264, 272
  - creating, 296-298
  - position of, 277
- Open command, 425, 608
- Open Project command, 40-41, 605
- Open Selection command, 608
- OpenDeskAcc() routine, 251, 345
- Opening
  - applications, 138, 158, 162
  - documents, 138, 158, 162
  - files, 429, 608
  - projects, 40-41, 605
- OpenPICT program, 429
  - resources for, 430
  - running, 436
  - source code for, 430-443, 593-597
- Options... command, 609-610, 619
- osEvt events, 133
  - with EventTracker, 154
  - flags for, 152
- Out button with debugger, 618
- Ovals, 64-65, 88-103
- P-RAM, 256
- Page Setup... command, 263, 449, 453, 608

- Page setup information, 444
  - Pager program
    - resources for, 382-385, 402
    - running, 392-393
    - source code for, 385-404, 581-587
  - Pages, printing, 444, 454
  - PaintArc() routine, 102
  - PaintOval() routine, 96
  - PaintRect() routine, 98, 123
  - PaintRoundRect() routine, 98
  - Parameter RAM, 256
  - Parameters
    - for Apple events, 484-485, 487
    - for functions, 31
    - passing, 33-34
    - typecasting and typechecking, 61, 614
  - ParamText() routine, 274, 394, 403
  - Parentheses for functions, 31
  - Part codes, 165-166
  - Partition (K) field, 606
  - Partners program, 497
  - Pascal
    - case-sensitivity in, 31
    - data types in, 30
    - strings in, 32, 621
  - Pascal String option with debugger, 621
  - Paste command, 609
  - PasToZero() routine, 636
  - Patterns
    - fill, 123
    - for pens, 62
  - PCWeek magazine, 493
  - PenMode() routine, 62, 64, 122
  - PenPat() routine, 62
  - Pens, drawing, 62, 100
  - PenSize() routine, 62
  - Periodicals, 493
  - PICT files and format. *See also*
    - OpenPICT program; PrintPICT program
  - for desk scrap, 404, 406, 411, 414
  - in dialog boxes, 266
  - icons for, 466-467, 469-470
  - with QuickDraw, 67
- PICT resources, 67
- displaying, 103-113
  - memory for, 182
  - for Pager, 385, 402
  - for PrintPICT, 445
  - for ShowPICT, 105
  - for Updater, 168-170
- Pixels, 52
- plainDBox windows, 58-59
- PlaySound() routine, 419-420, 422-424
- Pointers, 112, 247
- Points data type, 34
- Pop-up menus, 203, 206-208
- for Reminder, 293-295, 330, 358
  - for WorldClock, 212, 215-219, 234
- Pop-up rectangles, 206
- portRect field, 97
- Position
  - of buttons, 277
  - of dialogs, 305, 307, 368-369
  - of scroll bars, 380
  - of windows, 71-72
- PostReminder() routine, 321, 342, 350
- PrCloseDoc() routine, 444
- PrClosePage() routine, 444
- Precompile command, 613
- Preferences options, 609, 611
- Prefix options, 609
- Preprocess command, 612
- PrError() routine, 454
- Previewing
  - dialogs, 304, 306
  - windows, 72-73
- Print... command, 608
- Print Job dialog box, 261, 450, 453
- Print jobs, 444
- Print records, 444, 453
- PrintDefault() routine, 453
- printf() routine, 45
- Printing documents, 138, 158, 163, 482, 487-488
- Printing Manager, 443-444. *See also*
  - PrintPICT program
- PrintInit() routine, 447, 451-453
- PrintMonitor application, 279
- PrintPICT program, 443-444
  - resources for, 445
  - running, 449-450
  - source code for, 445-455, 598-601
- PrintPicture() routine, 448, 450-451, 454-455
- Priorities
  - with sending events, 192
  - of windows, 60-61
- PrJobDialog() routine, 444, 453
- PROCEDURES, 30
- Process Manager, 281-283
- Processes.h file, 635
- ProcID: field, 73
- Programming for System 7*, 497
- Programs, structure of, 68-69, 133-142. *See also* Applications
- Project files, 10
  - for Hello2, 75
  - for old versions, 25
- Project menu, 605-608
- Projects
  - creating, 40-42
  - running, 43-48
  - source code for, 43
- ProOpen() routine, 444, 453
- ProOpenDoc() routine, 444, 454
- ProOpenPage() routine, 444, 450, 454
- prototypes, function, 79
- PrPicFile() routine, 444, 454
- PrStdDialog() routine, 444, 453
- PtoCstr() routine, 32
- PtToAngle() routine, 34
- Pull-down menus, 203
- Purgeable windows, 74
- PutScrap() routine, 404-406
- Queues
  - event, 129-130
  - notification, 279-281
  - reminder, 338, 346-347, 360
- QuickDraw, 13, 20
  - coordinate system for, 51-56
  - for dialogs, 271
  - for FlyingLine, 113-125
  - for Hello2, 69-88
  - initializing, 80
  - for Mondrian, 88-103
  - for ShowPICT, 103-113
  - Toolbox for, 62-69
  - windows for, 56-61
- Quit command, 608
- Radio buttons, 265, 270
- Random numbers, 81, 95
- Random() routine, 98
- Randomize() routine
  - for FlyingLine, 117, 125
  - for Mondrian, 92, 97-98
- RandomRect() routine
  - for FlyingLine, 116-117, 124
  - for Mondrian, 92, 97
- randSeed global variable, 81, 95
- rDocProc windows, 57-59
- Reading, opening files for, 429
- ReadLocation() routine, 255
- RecalcLine() routine, 117-118, 125-126
- Recording sound. *See* SoundMaker
  - program
- RecordSound() routine, 418-419, 422-423
- Rect structure, 33
- Rectangles
  - drawing, 54
  - with menu bars, 122
  - pop-up, 206
  - rounded-corner, 64-65
  - zoom state, 247
- Redrawing windows, 165
- Reference constants, 179, 182, 219
- Regions, 67-68
  - content, 97, 185
  - drag, 56-57, 153, 177
  - handles to, 164
  - memory for, 188
  - and mouseMoved events, 137
  - for redrawing windows, 165
  - update, 183, 186-187
- Registering signatures, 461
- Reminder program, 15, 17, 284
  - ALRT resources for, 303-305
  - CNTL resources for, 293-295
  - DITL resources for, 295-303, 352
  - DLOG resources for, 305-307, 352
  - MBAR resources for, 285-286

- MENU resources for, 286-293
- running, 328-334
- SICN resources for, 307-308, 333, 356
- source code for, 309-327, 335-360, 560-578
- ReminderRec structure, 336
- Reminders menu, 292-293
- Remove command, 612
- Remove Objects command, 608
- RenumberTrailingReminders() routine, 320, 349-350
- Repeat rates, keyboard, 131, 154, 243
- Replacing text, 612
- reply parameter with DoOpenDoc(), 483
- Required events. *See* Apple events; EventTrigger program
- ResEdit, 7-8
  - for Finder information, 476
  - obtaining, 37
  - with Toolbox, 36-37
  - for WIND resources, 59-60
- ResEdit Complete*, 497
- ResEdit Info window, 37
- ResError() routine, 376
- ResolveAliasFile() routine, 360
- Resource forks, 40, 393
- Resource IDs, 39, 59-61, 73
  - for menus, 213-214, 236
  - numbering of, 93
- Resource Utilities.sea archive, 28
- Resourcecerer resource editor, 480, 500
- Resources, 4, 7-8, 38-40
  - attributes for, 607-608
  - for dialogs, 268-269
  - for EventTracker, 143-144
  - files for, 78
  - Finder, 459-460, 463, 471-478
  - for Hello2, 70-75
  - help, 479
  - for icons, 307-308, 333, 356, 464
  - for menus, 208-209
  - for Mondrian, 88-89
  - names for, 39, 42, 74
  - for OpenPICT, 430
  - for Pager, 382-385, 402
  - for PrintPICT, 445
  - for Reminder, 285-307
  - for ResWriter, 367-369, 374-377
  - for ShowClip, 407
  - for ShowPICT, 104-105
  - for SoundMaker, 416
  - for Updater, 168-170
  - for versions, 476-477
  - for WorldClock, 210-220
  - writing out. *See* ResWriter program
- Response procedures, 279, 281
- ResWriter program, 365-366
  - resources for, 367-369
  - running, 372-373
  - source code for, 369-379, 578-581
- Revert command, 608
- RgnHandle, 164
- ROM (read-only memory) for Toolbox, 36
- Rounded-corner rectangles, 64-65
- Routines, names for, 36
- Run command, 608, 617
- Running
  - EventTracker, 151-154
  - EventTrigger, 195-196
  - FlyingLine, 118
  - Hello, World, 43-48
  - Hello2, 78
  - Mondrian, 92-93
  - OpenPICT, 436
  - Pager, 392-393
  - PrintPICT, 449-450
  - Reminder, 328-334
  - ResWriter, 372-373
  - ShowClip, 411
  - ShowPICT, 108
  - SoundMaker, 419-420
  - Updater, 176-177
  - WorldClock, 233-235
- Sample applications, 11-12
- Save command, 425, 608
- Save A Copy As... command, 608
- Save All command, 613
- Save As... command, 425, 608
- Saving
  - debugging environment, 619
  - files, 425-426, 608
  - GrafPorts, 188
  - projects, 41
  - windows, 613
- Scaling fonts, 87
- ScheduleReminder() routine, 320, 345
- Scrap Manager, 404-406. *See also* ShowClip program
- ScrapStuff structure, 405
- Screen saver program, 113-125
- screenBits global variable, 81
- Scripts, 255
- Scroll bars, 57. *See also* Pager program
  - creating, 379-382
  - as dials, 265-266
- ScrollProc() routine, 388-389, 397-398, 401
- ScrollRect() routine, 163-165
- scrxxx text modes, 66-67
- Search menu, 611-612
- Seeds for random numbers, 81, 95
- Segmentation, code, 40
- Select All command, 609
- Selected menu items, 204
- Selecting files, 425
- Selector codes, 159
- SelectWindow() routine, 185
- Self-extracting archives, 25-28
- Semicolons (;), 625
- SendEvent() routine, 195, 198
- Sending Apple events. *See* EventTrigger program
- Separate STRS check box, 606
- Separator lines in menus, 212-213
- Serial numbers for applications, 283
- Set Context command, 622
- Set Project Type... command, 151-152, 472, 605-607, 633
- Set Tabs & Font... command, 609
- SetClip() routine, 188
- SetCtlValue() routine, 269-271, 355
- SetDialogCancelItem() routine, 272, 353, 374, 377
- SetDialogDefaultItem() routine, 272, 306, 353, 374, 377
- SetDialogTracksCursor() routine, 272, 353, 374-375, 377
- SetHandleSize() routine, 378
- SetHighByte() macro, 237
- SetIText() routine, 273-275
  - for Reminder, 353, 355
  - for ResWriter, 374, 377
- SetMenuBar() routine, 239, 340
- SetPort() routine, 63
  - for dialogs, 271
  - for FlyingLine, 122
  - for Hello2, 83-84
  - for Pager, 395
  - for printing, 444
- SetRect() routine, 122
- SetupReminderMenu() routine, 315, 343
- SetUpScrollBar() routine, 387-388, 393, 395-396
- SetUpZoomPosition() routine, 227-228, 246-249
- SetWRefCon() routine, 179
- Shadow text style, 85-86
- Shapes
  - of cursors, 81
  - drawing, 64-65
- Shift Left command, 609
- Shift Right command, 609
- Shortcuts... command, 620
- Show Context command, 622
- ShowClip program, 406
  - resources for, 407
  - running, 411
  - source code for, 407-415, 587-590
- ShowPICT program, 13-14, 103. *See also* Updater program
  - resources for, 104-105
  - running, 108
  - source code for, 106-113, 527-529
- ShowWindow() routine, 61
  - for dialogs, 272
  - for EventTracker, 156
  - for Hello2, 83
  - for Pager, 395
  - for Updater, 180
- SICN resources, 281, 307-308, 333, 356
- Signatures, 191, 460-461, 480

- Simple controls, 218
- Single-stepping, 618, 620
- Size
  - of buttons, 296
  - of files, 429, 440-441
  - of menu text, 203
  - of object code, 42
  - of pens, 62, 100
  - of resources, 378
  - of text, 66, 86-88, 203
  - of windows, 177, 185-188
  - of WorldClock, 257
- Size boxes, 57, 59, 184
- SIZE flags, 152, 328
- SIZE resource, 151-152, 472-473, 606
- SizeWindow() routine, 186
- Skip To Here command, 620
- sleep parameter for WaitNextEvent(), 137, 393
- SndPlay() routine, 420, 423
- SndRecord() routine, 416, 422-423
- Software
  - development tools for, 499-501
  - information on, 157
- Sound with notifications, 278-279, 281, 331
- SoundMaker program
  - resources for, 416
  - running, 419-420
  - source code for, 417-424, 590-592
- Source code
  - editor for, 10
  - for EventTracker, 144-151, 154-165, 534-541
  - for EventTrigger, 193-199, 481-482, 547-549
  - for FlyingLine, 114-125, 529-534
  - folder for, 25
  - font for, 20
  - for Hello, World, 43, 523
  - for Hello2, 76-77, 79-83, 523-524
  - for Mondrian, 89-98, 524-527
  - names for, 42
  - for old versions, 25
  - for OpenPICT, 430-443, 593-597
  - for Pager, 385-404, 581-587
  - for PrintPICT, 445-455, 598-601
  - for Reminder, 309-327, 335-360, 560-578
  - for ResWriter, 369-379, 578-581
  - for ShowClip, 407-415, 587-590
  - for ShowPICT, 106-113, 527-529
  - for SoundMaker, 417-424, 590-592
  - sources of, 501
  - for Updater, 170-184, 541-547
  - for WorldClock, 220-233, 235-257, 549-560
  - for XCMDs, 633, 635-636
- Source code window for debugger, 617-620
- Source menu, 612-613
- Spaces in source code, 43
- Spacing of text, 66
- Special menu, 209, 212, 214, 216, 233
- Spooling, 444
- Squares, 64-65
- Stacks, adding XCMDs to, 634-635
- Stages, alert, 275
- Standalone applications, 10, 83
  - BNDL resources for, 461-464, 466-467
  - EventTracker, 167-168
  - Finder resources for, 459-460
  - icons for, 463-471
  - signatures for, 460-461
- Standard File Package, 424-427
- Standard Get File dialog box, 425
- Standard Put File dialog box, 426
- Standard window state, 247
- StandardFileReply structure, 426-427
- StandardGetFile() routine, 351-352, 354, 425, 427, 429, 439
- StandardPutFile() routine, 426-427
- Standards in C, 35-36
- Static text fields, 273-274, 297, 300-301
- Stationery pads, 473-475
- Step button with debugger, 618, 620
- Stop alerts, 275-277
- Stop button with debugger, 619
- StopAlert() routine, 276-277, 394, 403
- 'STR' resources, 38, 274
  - for ResWriter, 365-367, 374-375, 377
  - types of, 477-478
- STR# resources, 274
- Str255 data type, 32, 34
- Strings, 34
  - C versus Pascal, 32
  - concatenating, 358-359
- Style menu, 212, 215-217, 233-234, 240-241, 252-254
- Styles, text, 66, 85-86
- Submenus, 206, 215, 240
- Subscribing to editions, 480
- SuperCard, 10
- Suspend & Resume Events flag, 472
- Syntax errors, 625-627
- SysBeep() routine, 82, 246
  - for debugging, 628
  - in XCMD, 636
- System 7, 6, 19, 21
- System files, resource forks in, 393
- System fonts, 84
  - loading, 80
  - for menu text, 203
- System global variables, 121. *See also* Global variables
- System windows, clicks in, 166, 184-185, 245, 343
- SystemClick() routine, 166, 245, 343
- SystemTask() routine, 137
- Table suffix for selector codes, 159
- Tabs in source code, 43
- Target addresses, 190, 192
- TCL 1.1 Demos.sea archive, 27
- Tech blocks, 19
- Tech Notes*, 20, 496-498
- Technical references, 496-497
- Technical support, 461, 498-499
- TEInit() routine, 80
- Templates, stationery pads as, 473-475
- Text
  - in dialogs, 273-274
  - drawing, 65-67, 85-88
  - for menus, 203
- TEXT files and format
  - for desk scrap, 404, 406, 411, 414
  - icons for, 466-467, 469-470
- TextBox() routine, 414
- TextEdit, initializing, 80
- TextFace() routine, 85-86
- TextFont() routine, 84, 252
- TextSize() routine, 86-88
- theAEventClass parameter for AEInstallEventHandler(), 142
- theAppleEvent parameter for DoOpenDoc(), 483
- thePort global variable, 81
- THINK C, 8-11
  - changes to, 614
  - debugger in, 617-622
  - Edit menu in, 609-611
  - File menu in, 608-609
  - installing, 25-29
  - Project menu in, 605-608
  - Search menu in, 611-612
  - Source menu in, 612-613
  - Toolbox access with, 29-40
  - Windows menu in, 613-614
- THINK C Debugger 5.0 file, 28
- THINK C User Manual*, 628
- THINK C Utilities.sea archive, 28
- THINK C 5.0 Demos.sea archive, 26
- THINK C 5.0 file, 28
- THINK C 5.0 Utilities file, 36-37
- THINK Class Library 1.1.sea archive, 27
- 32-bit mode and SIZE resource, 473
- This Project button, 611
- Thumb, 57, 398
- Time, displaying. *See* WorldClock program
- Time outs with sending events, 192
- Time Zone menu, 217-218
- Titles
  - for menus, 204, 208, 218-219
  - for windows, 72, 613
- TMON debugger, 500, 627
- Toolbox, 3-4, 6, 20-21, 29
  - calling sequences for, 30-31
  - #include, #define, and extern statements with, 31-32
  - parameter passing with, 33-34
  - for QuickDraw, 62-69
  - ResEdit with, 36-37

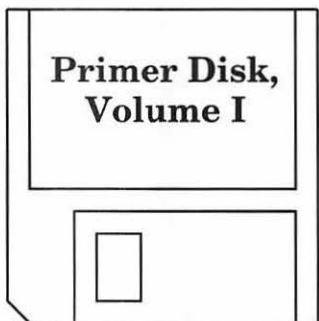
- resources with, 38-40
- standards for, 35-36
- strings with, 32
- ToolBoxInit() routine, 76, 79-81
- TopLeft() macro, 236, 248
- TPrint records, 452
- Trace button with debugger, 618, 620
- TrackBox() routine, 187, 246
- TrackControl() routine, 246, 381, 393, 397, 400-401
- Transfer... command, 608
- Trap addresses, 495
- Triangles in menus, 206, 208
- TrueType fonts, 87
- Two-pass compiling, 31
- Type suffix for selector codes, 159
- Typecasting, 61, 614
- Typechecking, 614
- Typefaces, 65-66
- Types
  - of events, 130-133
  - of files, 462-463, 474, 606
  - of projects, 605-607
  - of resources, 38-39
  - of windows, 57-59
- Underline text style, 66, 85-86
- Undo command, 609
- UnionRect() routine, 33
- UnloadScrap() routine, 404-405
- Update regions, 183, 186-187
- updateEvt events, 131, 153, 161, 177
- Updater program, 15-16
  - mouseDown events in, 173-174, 184-190
  - resources for, 168-170
  - running, 176-177
  - source code for, 170-184, 541-547
- UpdateWindow() routine, 391, 397, 399, 401-403
- Upper limits in debugging, 628
- Use Debugger command, 608, 617
- User-defined events, 132-133
- User groups, 501
- User items in dialog boxes, 266-267, 271
- User window state, 247
- Variables
  - global *See* Global variables
  - monitoring, 621-622
  - names for, 36
  - vers resource, 476-477
  - Version suffix for selector codes, 159
  - Versions, resource for, 476-477
  - Visibility of windows, 61
  - Volumes, 427
- WaitNextEvent() routine, 135-138
  - for EventTracker, 158
  - for modeless dialogs, 268
  - for Pager, 393
  - for Reminder, 340
  - for Updater, 180
  - for WorldClock, 242, 244
- Warnings, 19, 261, 267, 275-277
- WDEFs, 606
- what field for WaitNextEvent(), 136
- when field for WaitNextEvent(), 136
- where field for WaitNextEvent(), 136
- white global variable, 81
- White space, 43
- Whole Words Only option, 611
- Width
  - of pens, 100
  - of windows, 72
- 'WIND' Characteristics dialog box, 72-73
- WIND resources, 38-39, 59
  - for EventTracker, 143-144
  - for Hello2, 70-74
  - for Mondrian, 88-89
  - for OpenPICT, 430
  - for Pager, 382-383
  - problems with, 82
  - for ShowClip, 407
  - for ShowPICT, 104-105
  - for Updater, 168-169
  - for WorldClock, 220
- Window Manager, 13, 56, 80-81, 177
- WindowInit() routine
  - for EventTracker, 146, 156
  - for FlyingLine, 115, 119, 121-123
  - for Hello2, 76-77, 79, 82-88
  - for Mondrian, 90-91, 95, 100
  - for OpenPICT, 434, 442
  - for Pager, 387, 395
  - for ShowClip, 409, 413
  - for ShowPICT, 107, 109-110
  - for Updater, 171-172, 179-180
- for WorldClock, 223-224, 238-239
- Windows
  - activating, 131-133
  - closing, 56, 166, 185
  - creating, 122, 482-483
  - current, 63
  - dragging, 246
  - frames for, 153, 177
  - on grid, 52-53
  - memory for, 60, 74, 82
  - moving, 55
  - part codes for, 165-166
  - parts of, 56-57
  - position of, 71-72
  - redrawing, 165
  - saving, 613
  - setting up, 59-61
  - size of, 177, 185-188
  - types of, 57-59
  - update regions in, 183, 186-187
  - visibility of, 61
- Windows menu, 613-614
- WorldClock program, 15, 17, 209
  - CTRL resources for, 218-219, 241
  - MBAR resources for, 210-211, 239
  - MENU resources for, 212-218
  - running, 233-235
  - source code for, 220-233, 235-257, 549-560
  - variants of, 257-258
  - WIND resources for, 220
- Wrap Around option, 611
- WriteLocation() routine, 255
- WriteResource() routine, 274, 375
- Writing applications, 11-12
- Writing out resources. *See* ResWriter program
- wStorage parameter for
  - GetNewWindow(), 60
- XCMDs, 18, 631-637
- XFCNs, 631
- xFindApplication XCMD, 631-636
- ZeroScrap() routine, 404-406
- Zoom boxes, 57, 59, 73, 153, 177
- Zoom command, 613
- Zoom state rectangles, 247
- ZoomWindow() routine, 246





# Macintosh Programming Primer, Volume I: The Disk!

If you'd like to receive a complete set of source code, projects, and resources from Volume I of the Macintosh C Primer (Second Edition):



- 1) Fill out the coupon. Print clearly.
- 2) Attach a check for \$30. Make the check out to **Intelligence At Large**. (Make sure that the check is in **U.S. dollars**, drawn on a U.S. or Canadian bank. If you'd like the disk shipped outside the United States, please add \$5.
- 3) Send the check and the coupon to:

**Intelligence At Large, C1 Primer Disk**  
**3508 Market Street**  
**Philadelphia, PA 19104**

To order by phone, call **Intelligence At Large** at 215-387-6002.  
Have your Visa or MasterCard number ready.

Here's my \$30! Send me the Primer Disk quick!!!  
Mail the disk to:

Name \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

(Please allow three weeks for delivery)



Dave Mark Cartwright Reed

## Macintosh C Programming Primer

### Volume I, Second Edition

#### PRAISE FOR THE FIRST EDITION:

"One of the easiest-reading Mac programming books ever written. . . a first-rate guide to learning THINK C." — *MacWorld*

"We recommend *Macintosh C Programming Primer*." — *MacUser*

"The book wastes no time with peripheral issues but goes straight to the matter of writing real Mac applications. . . and making the best use of the Mac Toolbox and resources." — *MacWEEK*

The **Macintosh C Programming Primer, Volume I** is the bestselling tutorial on the art of Macintosh programming. Programmers new to the Macintosh®, but with some previous programming experience, will learn how to write Macintosh software using the powerful Toolbox, resources, and the Macintosh interface. The authors present the concepts involved in building an application—starting with the most basic and progressing to the more complex aspects of event-driven programming—and show you how to enter, compile, and run the programs you have created. The concepts are accompanied by a complete set of source code examples.

#### You will learn how to

- display and manipulate windows
- send and receive Apple events
- add icons to your applications
- manage scroll bars and dialog boxes
- create pull-down, pop-up, and hierarchical menus

This new edition has been revamped to reflect recent changes in Macintosh software, including System 7, and new versions of THINK C™ and ResEdit™. All the code has been rewritten from the ground up, specifically designed with System 7 in mind.

When you have completed the **Primer**, you will have the essential skills needed to build your own full-scale, System 7-savvy, Macintosh applications.

**Dave Mark** is the author of the bestselling *Learn C on the Macintosh* and a columnist for *MacTutor*. His company, M/MAC, does Macintosh consulting.

**Cartwright Reed** is President of Intelligence at Large, a Macintosh software and hardware development company in Philadelphia.

#### Also available:

*Macintosh C Programming Primer, Volume II* Dave Mark

*Macintosh Pascal Programming Primer* Dave Mark/Cartwright Reed

Cover concept by Doliber/Skeffington

Addison-Wesley Publishing Company

