Answers to the Most Frequently Asked Questions

Stephen H. Baker

with Dave Mark
Macworld®

Mac® Programming

FAQs

by Stephen H. Baker
with Dave Mark
About the Authors

Stephen H. Baker graduated from Rice University with a B.S. in Electrical Engineering and Computer Science. Since then he has been involved in extensive Macintosh and Windows development, both as a developer and as a manager. Steve is currently the Director of Release Management at Visix Software, Inc., a company that develops and supports client/server application software.

Dave Mark has written a number of best-selling Macintosh programming books, including *Learn C on the Macintosh*, *Learn C++ on the Macintosh*, the *Macintosh Programming Primer* series, and *Ultimate Mac Programming*. He has also collaborated on a number of Internet-related books, including *Home Sweet Home Page*, *Learn HTML on the Macintosh*, and *Learn Java on the Macintosh*. More recently, Dave has collaborated with Metrowerks, Inc., to create a series of CD-ROM-based products. Dave is also a partner in ZumaC@fe, the first cyber-cafe in New Mexico.
Welcome to the world of IDG Books Worldwide.

IDG Books Worldwide, Inc., is a subsidiary of International Data Group, the world's largest publisher of computer-related information and the leading global provider of information services on information technology. IDG was founded more than 25 years ago and now employs more than 7,700 people worldwide. IDG publishes more than 250 computer publications in 67 countries (see listing below). More than 70 million people read one or more IDG publications each month.

Launched in 1990, IDG Books Worldwide is today the #1 publisher of best-selling computer books in the United States. We are proud to have received 8 awards from the Computer Press Association in recognition of editorial excellence and three from Computer Currents' First Annual Readers' Choice Awards, and our computer-related information and Computerworld Special Reports.

Co mputcrworld Norge, Computcrworld

Computerworld Norge, Computerworld

Computcrworld Focus, Computcrworld

Computerwo rl d

Cable:

Welcome

World Account de N MARK: Communications World, Computerworld, M3C' World| South America.

You can count on our commitment to deliver high-quality books at competitive prices on topics you want to read about. At IDG Books Worldwide, we continue in the IDG tradition of delivering quality for more than 25 years. You'll find no better book on a subject than one from IDG Books Worldwide.

John Kilculleen
President and CEO
IDG Books Worldwide, Inc.
To Kelley, Michael, Matthew, and Jennifer. Thanks for your love, support, and patience. — *SHB*

To Deneen, Daniel, and our newest, Kelley Rowan — LFUEMISFAAZOK? — *DM*
Credits

Senior Vice President and Group Publisher
Brenda McLaughlin

Acquisitions Editor
Nancy E. Dunn

Brand Manager
Melisa Duffy

Managing Editor
Terry Somerson

Editorial Assistants
Dan Hilldale
Jean Leitner

Production Director
Andrew Walker

Supervisor of Page Layout
Craig A. Harrison

Associate Pre-Press Coordinator
Tony Augsburger

Media/Archive Coordinator
Leslie Popplewell

Development Editor
Kenyon Brown

Copy Editors
Kathryn Duggan
Katharine Dvorak
Earl Jackson, Jr.
Michael Welch

Technical Reviewer
Peter Ferrante

Project Coordinator
Ben Schroeter

Production Staff
Vince Burns
Renée Dunn
Mary Ellen Moran
Stephen Noetzel
Chris Pimentel
Elsie Yim

Proofreaders
Mick Arellano
Christine Langin

Indexer
Matthew Spence

Cover Design
Lieu Design
I bought my first Mac during the “first hundred days” back in March of '84. After about a week of fooling around, I set my sights on learning to program the darn thing. That’s when this book really began.

I started with the looseleaf version of Inside Macintosh Apple sent me. It was about 150 pages filled with Pascal and Assembler and cryptic references to these mysterious Toolbox Managers. Inside Macintosh filled me with questions. I was starving for answers.

Fortunately, I had a bunch of friends who were also taken with this new computer. My life was filled with puzzled phone calls, brain deadening all nighters, and pizza at dawn. As the Mac Toolbox became more complex, the questions multiplied. Thank God for e-mail and my buddy Steve!

Steve Baker was by my side through this whole Mac programming odyssey. A saver by nature, Steve was always haunting the nets, taking notes. We’d find a cool piece of sample code, I’d forget where we saw it, Steve would have it neatly organized on his hard drive. Steve kept everything. And now there’s this book.

Mac Programming FAQs is filled with more than 650 Macintosh programming questions. We tried to come up with a nice mix of questions so there’d be something here for everyone. There are answers and there is a boatload of code, too. And there’s a CD in the back that has a searchable database of all the questions, answers, and code found in the book.

I hope you enjoy this book as much as Steve and I enjoyed bringing it to you.

Dave Mark

March 1996
Preface

When I first started programming the Mac, about eight years ago, I had a lot of questions. What is an event? How do I update a window? What is a resource? Basic information necessary to make my Mac do what I wanted it to do. After spending countless days trying to unravel the Inside Macintosh volumes, I began making some progress. However, the more I learned, the more I realized I didn’t know. I had more questions now than when I began. Only now they were getting more complex and much more difficult to find answers to.

Things became much clearer after I read Dave Mark’s first book, Macintosh Programming Primer. In time, I learned my way around the Inside Macintosh volumes. As Apple released new systems with new managers and more capabilities, Dave and others wrote more books on programming the Mac. Apple also revamped their Inside Macintosh series, turning it into a collection of over two dozen volumes.

Online services such as America Online and various newsgroups on the Internet, most notably comp.sys.mac.programmer, also provided a wealth of information. Thousands of programmers used these forums to post questions they had about Macintosh development. These were real life questions that most of us either had encountered or would encounter in the course of our own development.

In the past year I’ve read thousands of pages of Mac programming material and gone through approximately 50,000 online postings from all levels of Macintosh programmers. Some of these postings asked questions about programming the Mac; others were responses to these questions; and the rest were opinions about a wide range of topics. After a while, I began seeing the same questions being posted again and again. Most of these were problems that I had faced or techniques that I really could have used. What a great source of information. Now, whenever I had a question, all I had to do was peruse a few thousand messages and I would have my answer. Or I could read the thousands of pages of the Inside Macintosh series to piece together a solution. What I really needed was a book that dealt with these frequently asked questions, or FAQs.

Mac Programming FAQs is a collection of 650 of the questions that get repeatedly asked by Mac programmers. While putting this book together Dave and I had a few decisions to make. Do we go wild on the prose and spend pages on each question? Do we concentrate on just a few managers and cover them in detail while sacrificing coverage on all of the other topics? Do we target one level of programmer? The answer to each of these was no. The questions selected for this book cover the
majority of the Macintosh Toolbox and range from beginner issues to advanced techniques. The answers are concise and to the point, allowing for a maximum coverage of topics. Complete and detailed code is provided in the projects on the CD in the back of the book.

*Mac Programming FAQs* was written to serve several functions. First and foremost, it is meant to answer those nagging questions that routinely come up in Mac programming projects. It is also meant to be a valuable guide to help you learn a number of advanced techniques. And finally, it is meant to be a source for new ideas and approaches that you can incorporate into your projects. With the large volume of questions, the wide range of topics, and the abundant code samples, there will be something in here for everyone.

### How This Book is Organized

The FAQs presented in this book are organized into 33 chapters. Each chapter represents a separate Toolbox Manager or development topic to facilitate the locating of an answer to a particular problem. Some of the chapters, such as Chapter 11, Gestalt Manager, consist of only a handful of FAQs while others are made up of as many as 40. Those chapters that contain a large number of FAQs are further divided into subsections.

The first part of this book provides a summary listing of all 650 FAQs, ordered by their appearance in the book. This listing will enable you to scan the material included to locate the questions you want. Next come 33 fun-filled chapters of FAQs. These chapters are listed below.

**Chapter 1: Apple Event Manager**

Apple events is one of the most complicated topics in Mac programming. This chapter answers some basic questions to get you started incorporating Apple events into your applications. It also tells you where you can get all the information you’ll need to really dig into this topic.

**Chapter 2: Color**

The Color chapter discusses how color is implemented on a Macintosh. It describes how palettes and color tables are used and how they can be manipulated to produce different effects. This chapter also covers how to draw in color. It is related to Chapter 23, QuickDraw: Drawing.
Chapter 3: Cursors

This chapter discusses the manipulation of cursors. It shows how to change the shape of, hide, and move the cursor in an application. Several FAQs also deal with the topic of rotating cursors.

Chapter 4: Development Environment and Language Issues

This chapter discusses Pascal and Assembly language issues that pertain to the development of C applications. It also includes some questions about the Metrowerks CodeWarrior and Symantec C++ compilers.

Chapter 5: Devices: Serial Ports

The Devices: Serial Ports chapter covers some basic issues related to using the serial port. It discusses how to configure the ports and how to use them to send and receive data.

Chapter 6: Dialogs and Controls

The Dialogs and Controls chapter deals with a number of topics related to using dialogs, dialog items, and controls in your application. It covers the implementation of the different types of dialogs and alerts. It shows how to put scroll bars in a window, how to put the thick border around the default button, and how to perform various manipulations on dialog items and controls. It also discusses the use of event filters to perform specialized processing.

Chapter 7: Events

The core of a Macintosh application is the event handling routines. The Events chapter discusses how your application should process events to behave properly. How to detect when it gets sent to or returns from the background and how to detect when a key is held down are just some examples. There are several questions that show how to get the status of the keyboard, both from events and without events. There are also several questions that deal with the mouse, such as determining the state of the mouse button, detecting a double-click, and even moving the mouse through code. For the more advanced programmers, the GNE filters are explained with some examples.
Chapter 8: Files: File Manager

This chapter covers how to reference, find, and manipulate files, folders, and volumes. Also included in this chapter are questions that deal with the special System folders and determining information about drives attached to your Macintosh.

Chapter 9: Files: Standard File Package

This chapter discusses some basic issues related to the Standard File Package. It shows how to set the folder and volume that initially appear in the Standard File dialogs and discusses customizing the interface.

Chapter 10: Finder and Desktop

This chapter discusses the interaction between the Finder and applications, the Desktop database, and the relationship between files and their icons.

Chapter 11: Gestalt Manager

The Gestalt Manager chapter answers a few questions about obtaining System information with the Gestalt() call. It includes more advanced uses of the Gestalt Manager, registering your own Gestalt selectors, and using the Gestalt mechanism to communicate between processes.

Chapter 12: Graphics Devices

The Graphics Devices chapter focuses on the interaction between an application and display devices. It shows how to get and set different display characteristics of a monitor such as its pixel depth, color mode, and display size. It shows how to handle multiple monitors and shows two different methods of screen fading. It also describes how to access the image of the desktop to perform screen captures.

Chapter 13: Icons

The Icons chapter discusses various uses of icons in an application such as placing icons in menus and dialogs, and drawing icons in the different Finder states.

Chapter 14: Lists and the List Manager

This chapter covers issues that pertain to lists. It covers manipulating the contents and the appearance of lists, retrieving information about the lists, and working with a list's click loop.
Chapter 15: Memory

This chapter covers issues related to an application's use of memory. It discusses handles and pointer manipulation, the system heap, the A5 world, and temporary memory.

Chapter 16: Menus

The Menus chapter covers topics related to using menus in an application. It discusses such issues as manipulating the appearance of menus and menu items, displaying icons as menu and item titles, and creating hierarchical menus. It covers advanced topics such as hiding the menu bar, creating customized windows to display pictures, drawing while a menu is being pulled down, and much more.

Chapter 17: Power Macintosh Programming

This chapter covers topics specific to programming on the Power Macintosh. It discusses universal procedures pointers, fat applications, and other issues.

Chapter 18: Printing

The Printing chapter covers topics related to printing text and graphics. It shows how to get information about the printing environment and how to send data to the printer. It also covers more advanced topics such as using picture comments to perform special printing functions.

Chapter 19: Processes: Notification Manager

This chapter covers topics related to using the Notification Manager. It discusses the different ways of communicating with the user from a background process.

Chapter 20: Processes: Process Manager

This chapter covers topics related to getting information about running processing and launching applications from processes.

Chapter 21: Processes: Time Manager

This chapter covers issues related to the Time Manager. It discusses how to execute a Time Manager task on both a 68K Macintosh and a Power Macintosh. It also shows how to use the Time Manager to measure execution times.
Chapter 22: Processes: Vertical Retrace Manager

This chapter covers issues related to executing VBL tasks. It discusses both system based VBL tasks and slot-based VBLs.

Chapter 23: QuickDraw: Drawing

This chapter covers a number of issues related to drawing on the Macintosh. It discusses working with drawing modes, drawing different types of lines, drawing with patterns, and highlighting. It discusses the manipulation of bitmaps and pixmaps, including flipping and rotation. It shows how to use GWorlds to perform fast offscreen drawing. It also covers a number of more advanced topics such as customizing the QuickDraw bottleneck routines to perform special drawing functions, drawing outside a window, and drawing color selection rectangles.

Chapter 24: QuickDraw: Pictures

This chapter covers pictures as they pertain to QuickDraw. It discusses different issues related to the creation and drawing of pictures. It also discusses picture files, their formats, and how to create them and read from them.

Chapter 25: Resources

Resources are an integral part of any Macintosh program. This chapter discusses accessing resources from files, creating and storing resources, and other issues related to putting resources in an application.

Chapter 26: Scrap Manager

The Scrap Manager chapter covers topics related to storing and accessing data in the scrap. It also shows how to get various types information about the scrap.

Chapter 27: Sound

The Sound chapter covers just about everything you wanted to know about sound on your Macintosh. It discusses playing sound both synchronously and asynchronously from resources and from files. It also discusses recording sound, adjusting the playing volume and other sound characteristics, and a host of other topics.

Chapter 28: Speech Manager

This chapter covers the basics of putting speech in your application. It shows how to get the speech manager to speak a buffer of text, how to access different voices in your system, and how to synchronize synthesized speech to other actions.
Chapter 29: Strings, Bit Manipulation, and Random Numbers

This chapter contains several tidbits of information that are often used in applications. It discusses using Pascal strings in C programs, performing other types of string conversions, working with individual bits, and generating random numbers.

Chapter 30: System Extensions and Patching Traps

System extensions and patching traps are two complex topics. This chapter provides a lot of good information and examples to get you started.

Chapter 31: Text: Fonts and Drawing

The chapter deals with topics involving working with fonts and drawing text. It shows how to get different types of font information. It covers the basics of drawing text as well as drawing dimmed text and rotating text.

Chapter 32: Text: TextEdit

This chapter covers issues related to using TextEdit to put text editing capabilities in an application. It shows how to get, set, and retrieve information about text in a TextEdit field. It also shows how to work with the styles in a multistyled TextEdit field, manipulate the TextEdit cursor, and properly use scroll bars with TextEdit.

Chapter 33: Windows

This chapter covers topics related to windows in an application. It discusses manipulating a window's position, ordering, and appearance. It discusses drawing on and updating windows. It shows how to create a simple floating window as well as implementing scroll bars in a window.

Appendix A: Using the Companion CD-ROM

This appendix describes using the special FAQ database program that's included on the CD-ROM. The information in the database includes all the questions, answers, and source code snippets that are discussed in the book. The FAQ database program enables you to view all of the questions and answers in the FAQ database. The true benefit of the program is that it enables you to search FAQs by keyword or by topic. You can conduct searches of the database to find answers to your questions.
Appendix B: Gestalt Manager Selector Codes

This appendix lists the Apple-defined Gestalt Manager selector codes. Apple defines two distinct kinds of selector codes: environmental selectors, which supply information you can use to control the behavior of your application, and informational selectors, which supply information you can't use to determine what hardware or software features are available.

Conventions Used in This Book

For easy reference, each FAQ is identified by its own FAQ number. The FAQ number consists of two parts: the number of the chapter in which the FAQ resides and the order number of the FAQ within that chapter. For example, FAQ 12-2, “What is the main graphics device?” is the second question in Chapter 12.

In the answers to some FAQs you will see references to other FAQs via “See FAQ...” These are questions that are an integral part of the current FAQ's answer. Rather than duplicate the text of the referenced answer, I send you a pointer to it. These references appear like this:

See FAQ 6-30, “How can I set the fonts and colors of my dialog items?”

At the end of many FAQs you will see references to other FAQs via “See also FAQ...” These referenced questions contain information that will enhance the current FAQ.

See also FAQ 6-14, “How do I deactivate the default button?”

Many FAQs deal with topics that could be categorized under one or more chapters other than the chapter in which it is currently located. These other chapters are referenced via “See also Chapter...” and appear at the end of a FAQ like this:

See also Chapter 24, “QuickDraw: Pictures.”

At the end of some of the chapters there is a list of related FAQs from other chapters in the book. These FAQs are under the heading “Topic-Related FAQs” and are referenced the same way as the related FAQs of an individual FAQ.
What's on the CD-ROM

The CD-ROM in the back of the book contains a database of the information presented in the book. Along with this data, we've included a program that enables you to search the FAQs for keywords quickly and view related FAQs easily. Appendix A describes the interface of the FAQ database program.

Also included on the CD are two folders that contain source code found in the book. The first folder is called "FAQ Code Snippets." It contains the longer code examples from the FAQs. This information is provided in text format and is organized by chapter and question. The second folder is called "FAQ Code Projects." It contains sample executables of the more interesting and fun code from the book, along with everything you need to build them. These projects include all the source and resource files.

I Want to Hear from You

A large part of Mac Programming FAQs was made possible by the curiosity, perseverance, and love of programming of all you Mac programmers out there. The Internet and the various on-line services have provided us with the vehicle we needed to communicate with each other on a grand scale. I have enjoyed reading all of your questions, responses, and views on life. With them I feel that I have been able to put together a solid collection of programming questions and answers. If you have any comments on this book, have a favorite FAQ that you would like to see included in future versions of this book, or just want to drop me a line, you can reach me through e-mail at REKABS@aol.com. It is my hope that this book becomes a valuable tool in your Macintosh development arsenal.

Steve Baker
Acknowledgments

I’d like to thank the people who helped make *Mac Programming FAQs* possible. This book required many late nights and a lot of hard work by a number of people and a whole bunch of moral support to keep me going.

First and foremost I want to thank my friend Dave who helped me write this book. Eight years ago Dave introduced me to the world of Mac programming. He has been a mentor ever since.

Next I would like to thank Nancy Dunn, Ken Brown, and the rest of the crew at IDG who were responsible for helping transform an idea and a thousand sheets of paper into the creation you are now holding in your hands.

I would also like to thank my technical reviewer, Peter Ferrante, who read through my material and gave me a lot of great feedback.

During the past year my family endured my long hours and endless work. A special thanks to my wife, Kelley, who corrected my late night editing mistakes and took care of things while I was FAQing. Thanks to my boys, Michael and Matthew, for letting me work when I needed to and playing with me when I needed a break. And thanks to my little girl, Jennifer, who wasn’t here when the book started but entered our lives soon after, bringing even more joy to my life.

Additional thanks go to the rest of my family for their support and encouragement. To my parents, Richard and Lilly Baker, for raising a stubborn son and teaching me not to quit. To my sister, Sondra, to my brother, Ron, and his family, Carolyn, Lauren, and Jacob.

Thanks to my in-laws for taking me on a much needed break to Disney World: to Dan and Joleen Megilligan, to Shannon and Danny, to Chris and his family, Sharon, Kelsey, and Alissa.

And finally I would like to thank all those Mac programmers who had the courage and perseverance to reach out and ask questions.
## Contents at a Glance

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>ix</td>
</tr>
<tr>
<td>Preface</td>
<td>xi</td>
</tr>
<tr>
<td>Chapter 1: Apple Event Manager</td>
<td>1</td>
</tr>
<tr>
<td>Chapter 2: Color</td>
<td>11</td>
</tr>
<tr>
<td>Chapter 3: Cursors</td>
<td>39</td>
</tr>
<tr>
<td>Chapter 4: Development Environment and Language Issues</td>
<td>57</td>
</tr>
<tr>
<td>Chapter 5: Devices: Serial Ports</td>
<td>67</td>
</tr>
<tr>
<td>Chapter 6: Dialogs and Controls</td>
<td>75</td>
</tr>
<tr>
<td>Chapter 7: Events</td>
<td>119</td>
</tr>
<tr>
<td>Chapter 8: Files: File Manager</td>
<td>143</td>
</tr>
<tr>
<td>Chapter 9: Files: Standard File Package</td>
<td>183</td>
</tr>
<tr>
<td>Chapter 10: Finder and Desktop</td>
<td>195</td>
</tr>
<tr>
<td>Chapter 11: Gestalt Manager</td>
<td>205</td>
</tr>
<tr>
<td>Chapter 12: Graphics Devices</td>
<td>211</td>
</tr>
<tr>
<td>Chapter 13: Icons</td>
<td>237</td>
</tr>
<tr>
<td>Chapter 14: Lists and the List Manager</td>
<td>243</td>
</tr>
<tr>
<td>Chapter 15: Memory</td>
<td>259</td>
</tr>
<tr>
<td>Chapter 16: Menus</td>
<td>271</td>
</tr>
<tr>
<td>Chapter 17: Power Macintosh Programming</td>
<td>305</td>
</tr>
<tr>
<td>Chapter 18: Printing</td>
<td>315</td>
</tr>
<tr>
<td>Chapter 19: Processes: Notification Manager</td>
<td>327</td>
</tr>
<tr>
<td>Chapter 20: Processes: Process Manager</td>
<td>339</td>
</tr>
<tr>
<td>Chapter 21: Processes: Time Manager</td>
<td>349</td>
</tr>
<tr>
<td>Chapter 22: Processes: Vertical Retrace Manager</td>
<td>363</td>
</tr>
</tbody>
</table>
# Table of Contents

**Foreword** ................................................................................................................. ix

**Preface** ....................................................................................................................... xi

## Chapter 1: Apple Event Manager ................................................................................. 1

- What are the four required Apple events? ................................................................. 1
- How do I learn how to program with Apple events? ............................................... 3
- How can I determine what Apple events an application supports? ......................... 3
- How do I add support for the required Apple events in my program? .................... 4
- How does the Finder tell my application that one of its documents was double-clicked? .... 5
- How can I identify the sender of an Apple event? .................................................. 6
- Can an application send Apple events to itself? ....................................................... 7
- How can I prevent an Apple event that my application sends to itself from getting recorded? ......................................................................................................................... 8
- Why do I get error -903 when I try to send an Apple event with AESend()? ........ 8
- Can I send an Apple event from a code resource? .................................................. 8
- Does AEPutParamPtr() (and other AEPut...() functions) copy the data passed to it? ................................................................................................................................. 8

## Chapter 2: Color ........................................................................................................... 11

- What are the RGB and HSV color models? ............................................................... 12
- What is a color lookup table (CLUT)? ......................................................................... 14
- What is a color palette? ............................................................................................... 14
- What is the difference between pmCourteous, pmTolerant, and pmExplicit colors? .... 14
- What is an inverse table? ............................................................................................ 15
- How do I convert from one color model to another? ................................................ 17
- How can I create a grayscale ramp? ........................................................................... 17
- How do I reference shades of gray in my program? ................................................. 18
- Why does inverting a color often produce unexpected results on a 256-color system? .... 19
- What RGB value does the system use for dimmed buttons, menus, and window titles? 19
- How do I get and set the current highlight color? ..................................................... 20
- How can I get drawing commands to draw in the highlight color? ......................... 21
- What are the RGB values of the eight colors used in Basic QuickDraw? ................. 22
- Why do different colors passed to RGBForeColor() appear the same when drawn on the screen? .............................................................................................................. 23
- How are 48-bit RGB values displayed when my screen is only 16 bits? .................... 23
- Where can I find documentation on the Palette Manager and Color Manager? ......... 24
- How do I create a palette? ......................................................................................... 24
Macworld Mac Programming FAQs

Chapter 3: Cursors

What is an animated cursor? ................................................................. 40
How do I change the shape of the cursor? ........................................ 40
How do I set the cursor back to the arrow cursor after I set it to one of the other standard cursors? ................................................................. 42
How do I get my program to change the cursor to another cursor when over a certain area in my window? ................................................................. 42
When my application returns to the foreground, why doesn’t the cursor return to the cursor shape it was in immediately before it was sent to the background? .......... 44
Why does my program work fine with a black-and-white cursor but crashes with a color cursor? ................................................................. 44
How do I hide the cursor? ................................................................. 44
Why won’t ShowCursor() make the cursor visible again after I call HideCursor() more than once? ................................................................. 45
Will a hidden cursor report mouse-down and mouse-moved events? .... 45
How can I get the location of the cursor? ........................................ 46
How do I get my program to move the cursor to a specific location on the screen? ................................................................. 46
How do I put a spinning cursor in my program? ................................ 49
How can I spin the cursor using an interrupt routine? ................. 52
Why does my VBL task crash when I try to spin a cursor? ............. 55
How do I animate a color cursor? ................................................................. 56
Chapter 4: Development Environment and Language Issues .................. 57
What are the differences between C and Pascal that I need to know about in order to work with Inside Macintosh? ................................................................. 58
When do some routines have to be declared with the “pascal” keyword? ............ 60
What do ONEDWORDINLINE(), TWOWORDINLINE() and THREEMWORDINLINE() macros in the universal header files do? ..................................................... 60
Are PowerPC and 68K Assembly the same? ......................................................... 61
How do I put Assembly code in my CodeWarrior C project? ............................... 61
How do I put Assembly in my Symantec C project? ............................................. 62
How do I access global variables in an Assembly routine in a CodeWarrior program? 62
How do I access variables in a block of inline Assembly in my Symantec C program? 63
How do I pass Assembly language function parameters in a CodeWarrior C project? 63
What is the difference between CodeWarrior Gold, Academic, Bronze, and Discover Programming for Macintosh? ................................................................. 64
Can Symantec C++ version 8.0 produce 68K executables? .................................... 65

Chapter 5: Devices: Serial Ports ................................................................. 67
How do I access the serial ports? ........................................................................... 67
Is there any way of finding out the names of all the serial ports available on a particular machine? .......................................................... 69
How can I tell if the Mac only contains one serial port, or if both a printer and modem port are present? .............................................................. 70
What are the default settings when a serial port is first opened? ......................... 70
How can I configure the serial ports? ................................................................. 71
Why do I have to open the output device driver for the serial port before opening the input device driver, even if I'm not going to use it? .......................... 72
How can I increase the size of the input buffer over the default 64-byte buffer? ....... 72
What transmission rates can the serial port support? .......................................... 73
How do I send and receive data over the serial port? ......................................... 74
How can I tell when data has been read by the serial port into the input buffer? .... 74

Chapter 6: Dialogs and Controls ................................................................. 75
What is a control definition function (‘CDEF’)? .................................................. 76
What is a user item? ....................................................................................... 76
How do I put up a modal dialog box? ............................................................... 77
How do I put up a movable modal dialog box? .................................................. 79
How do I put up a modeless dialog box? ........................................................... 79
How do I create a color dialog box? ................................................................. 80
Why can’t I draw on a dialog box when it comes up? Why do the dialog box items act as if they are located at different coordinates? .............................. 81
How can I put a PICT item in a background of a dialog box without obscuring the other items? ................................................................. 81
Why can I only draw in the eight basic colors in a dialog box? ............................ 82
Why can’t my modal dialog box detect disk-insert events? ................................. 82
What are Apple’s user-interface guidelines concerning the placement of items in a dialog box? ................................................................. 82
How do I get the thick border around the default button in a dialog or alert box? .......... 83
How do I set the default item and the default cancel item in a dialog box? ................. 88
How do I deactivate the default button? ........................................................................ 88
How do I make the buttons the same color as my color dialog boxes? ......................... 89
How do I get controls to dim with light gray instead of bitmapped gray? ..................... 89
How can I determine the type of a control? ...................................................................... 91
Why can't I click on a button after I move it with MoveControl()? ............................ 92
Why don't the buttons and other controls get refreshed when my window comes to the front? ........................................................................................................................ 92
How do I activate and deactivate a control? ............................................................... 93
How do I put scroll bars in a window? ........................................................................ 94
What is the size of a standard scroll bar? ...................................................................... 99
How do I use user items to draw in a dialog box? .......................................................... 100
Why doesn't FindDialogItem() find an item when I pass it the mouse location from an event? ....................................................................................................................................... 101
Why doesn't FindDialogItem() return the proper dialog box item number? .................. 101
Why can't I disable an edit text field in a dialog box with HiliteControl()? ................. 102
How do I get the text from an editable text item in a dialog box? ................................. 102
How do you deactivate (gray-out) static text items and edit text items in a dialog box? ... 102
How can I have variables in static text fields? ............................................................ 103
How can I set the fonts and colors of dialog box items? .................................................. 104
How can I set the background of the text-entry fields in a dialog box? ....................... 105
How do I put an event filter in a dialog box? ................................................................. 106
How can I get an edit text item in a dialog box to only accept numbers as input? ....... 109
How can I stop an edit text item from echoing typed characters for a password entry? ... 111
Why do I continuously get update events in my event filter procedure of ModalDialog()? 113
When I add an event filter to ModalDialog(), how do I get the Return key to activate the default item? ......................................................................................................................... 113
How do I put up an alert and do I close it? ..................................................................... 114
How can I put my own icon in the upper-left corner of an alert? .................................. 114
How can I get an alert to play a sound other than the System alert sound? .................. 115
How can I customize the text in an alert box? ............................................................. 116
How does the alert stage value get incremented, and how can I reset it? ..................... 116

Chapter 7: Events ........................................................................................................... 119
What is a GNE filter? .................................................................................................... 120
What is the duration of a tick? ..................................................................................... 120
How do I detect if the user is holding a key down? ..................................................... 120
How do I read the keys of the keyboard without getting an event from the event queue? . 121
Why isn't GetKeys() passing back the key values that I expect? ............................... 122
How can I detect multiple simultaneous keystrokes on a single keyboard? ............... 123
How can I tell when the Shift, Control, Command, or Option key is pressed? .......... 123
How do I detect that the arrow keys have been hit? .................................................. 125
Why aren't keyDown events generated when I hit the Shift key or the Option key? ..... 126
Why don't I receive the key-up event, even when I set the eventMask in
WaitNextEvent( ) to every Event? ................................................................................ 126
How can I get the character code and virtual key code of a key pressed by the user? .......... 127
How do I capture keystrokes in all applications? .............................................................. 128
How can I tell if the mouse button is up or down? ............................................................. 129
How do I detect a double-click? ........................................................................................ 129
How can I get the current mouse coordinates? ................................................................. 130
Are the mouse coordinates returned in the where field of the event structure in local or global coordinates? .................................................................................................. 131
How can I tell when the mouse moves if there are no mouse-moved events? ..................... 131
How can I tell what part of my window the user clicked in? ............................................. 133
Is there an event priority that determines the order in which events are returned by the Event Manager? ........................................................................................................ 134
How can I tell how much time has elapsed between events? ............................................. 134
What is the purpose of the sleep parameter in WaitNextEvent()? ...................................... 135
How can I tell what type of OS event is posted? ................................................................ 135
When my program receives an activate event, how can I tell if the window is activated or deactivated? ........................................................................................................ 136
How can I tell when my application goes to the background and returns to the foreground? ...................................................................................................................... 137
Why won't my application receive suspend or resume events? ........................................ 138
How do I send my program an event? ............................................................................... 138
Is it possible to post an event to the head of the event queue? .......................................... 138
Why does my application get continuous update events? ................................................. 139
How do I delay my program for a time that will be consistent on all machines? .............. 139
How do I create and install a GNE filter? .......................................................................... 139
Why can't my GNE filter detect OS events (such as when an application comes to the foreground)? ........................................................................................................... 141

Chapter 8: Files: File Manager ......................................................................................... 143

What is an FSSpec? ........................................................................................................... 144
What is the difference between the default directory and volume and the current directory and volume? ................................................................. 145
What are the creator ID and file type of a file? ................................................................. 146
What is a working directory? ............................................................................................ 146
What are resource and data forks of a file? ....................................................................... 147
What is the drive queue? ................................................................................................... 147
How do I specify a path name, and how do I include and separate the volume name, directory names, and file name? ................................................................. 148
What is the maximum length of file, directory, and volume names? ................................ 148
How do I get the full path of a file referenced by a vRefNum, parent directory ID, and file name? ........................................................................................................... 149
How can I get and set the default volume and folder? ...................................................... 150
How do I find the DirID and VolRefNum of a file located in the same folder as my application? .............................................................................................................. 150
How can I get the directory ID of a folder from the full path name and from the parent folder's ID and directory name? ........................................................................ 151
How do I find the name of a folder from dirID and vRefNum? .......................................... 152
XXVili Macworld Mac Programming FAQs

How do I get the vRefNum of the startup volume? ........................................................... 153
Can WDRefNum and vRefNums be used interchangeably, or are working
directories obsolete? ..................................................................................................... 153
Why don’t the current directory and volume used in the Standard File Package calls
set the default directory and volume used in File Manager calls? ............................ 153
How do I access the top directory of a volume and on the Desktop? ........................... 154
How do I get the directory and volume of my application file? ..................................... 154
How do I convert a working directory to a volume reference number and directory ID? .. 155
How do I find invisible files and folders in a directory? ............................................. 155
How can I get all of the files in a directory? ................................................................... 156
How can I recurse through all files and folders in a directory? ..................................... 156
How can I find all mounted volumes? ......................................................................... 158
How do I search a volume for files that match a set of search criteria? ......................... 159
How do I access the System (Blessed) folder? ............................................................ 162
Where should my application store temporary files? ................................................... 163
How do I find the special folders such as the System folder and the Preferences folder? ... 164
How do I read data from and write data to a file? ................................................................ 166
Can I interchange the file reference numbers that I get back from opening a data fork
of a file and those that I get from opening a resource fork of a file? .......................... 167
How can I tell if a file is open, and if it is open, how can I tell which fork of the file
is open? ................................................................................................................. 167
How do I delete only the resource fork or the data fork of a file? ................................ 168
How do I delete a folder? ............................................................................................ 169
How do I alter a file without automatically changing the time stamp? ......................... 169
How do I create an invisible file? ............................................................................... 170
Why can’t I create a file smaller than 4K on my hard drive, and why does the size of
the same file increase when I move it to a larger drive? ........................................... 172
How do I open files whose names start with a period like driver names do? ..................... 172
How do I open a file in the current directory with FSOpen()? ..................................... 172
Why do I get the compiler error, “Error: ‘ioNameptr’ is not a struct/union/class member,”
when I try to access the ioNameptr field of an HParamBlockRec structure? .......... 173
How can I get the Finder label menu color and label string of a file? ........................... 174
How can I tell if there is a diskette in my diskette drive? ........................................... 175
How can I tell if a disk is locked? ............................................................................... 176
How do I tell if a Macintosh volume is a floppy or hard disk? .................................... 177
How can I determine the amount of free space on a volume? .................................... 180

Chapter 9: Files: Standard File Package ........................................................................ 183

What is the difference between the dialog hook function and the filter function in
CustomGetFile() and CustomPutFile()? ................................................................. 184
How do I open a file the user has selected using StandardGetFile()? ......................... 184
How can I tell if the Cancel button in a standard file dialog box gets hit? ..................... 185
How can I select which files are displayed in the file list of the Standard Get File
routines? ................................................................................................................... 186
I have a custom dialog box that I pass to CustomGetFile() in which I allow the user to select a file type to look for. How can I get the Standard File Package to rescan the current directory, calling my file filter procedure if necessary? ................................ 187
How do I display only visible files and folders in a standard file dialog box? ................................ 188
How do I tell the StandardGetFile() dialog to display every type of file? ................................ 188
How do I set which folder should initially be shown in the Standard File Package calls? ........ 189
How do I get the current folder and volume that are initially displayed by the Standard File Package? .................................................................................................. 191
How can I customize StandardGetFile() and StandardPutFile() to display and process my own dialog box? ..................................................................................................... 192
Why doesn’t my dialog hook function for CustomGetFile() recognize the sfHookFirstCall and sfHookLastCall pseudo-item numbers? ....................................... 193
How do I select a folder with the Standard File Package? .................................................. 193

Chapter 10: Finder and Desktop .......................................................... 195
What is the Desktop database? .......................................................................................... 196
How do I get my application to respond to the Finder for actions such as printing a document or opening a document double-clicked on by the user? ........................................ 196
How do I make the Finder see a new file that I created, or if I change the file type, how do I display a new icon for it? ................................................................. 197
How do I force the Macintosh Finder to update its windows after my application has changed a file's FndrInfo? .................................................................. 198
How do I access the Desktop file? ..................................................................................... 199
How can I get the application that opens a file with a given creator? .............................. 200
Why is my application's old icon still displayed on the Desktop after I change the icon? ...... 201
How do I attach an icon to an application? .................................................................... 201
How do I get the icon of a file with a given type and creator? ........................................... 202
Why is my file displaying the standard icon and not my bundled icon? ............................. 203

Chapter 11: Gestalt Manager ........................................................................ 205
How can I tell if Gestalt Manager is available? ......................................................... 205
How do I use Gestalt to get information? ..................................................................... 206
What are the available Gestalt values? ......................................................................... 207
How do I register new selectors in Gestalt? .............................................................. 207
Can my Gestalt selector function have global variables? ............................................. 208
How can I use Gestalt to communicate between processes? ....................................... 209

Chapter 12: Graphics Devices .................................................................... 211
What is a Gamma table? .......................................................................................... 212
What is the main graphics device? .............................................................................. 212
How can I determine the maximum pixel depth that a screen can support? ................. 213
How can I get the screen with the greatest color depth? ............................................ 214
How can I determine which monitor is the main monitor? ......................................... 214
How can I determine how many monitors are attached to my computer? .................... 214
How do I find the current pixel depth and color mode of the screen? .......................... 215
How do I find the pixel size of a monitor? ................................................................. 217
How can I fade my display to black and back again or to another color? .................. 218
How can I fade the screen of a direct color device to black? ........................................ 223
How can I get the image of a screen (as in a screen capture)? ..................................... 227
How can I set the pixel depth and color mode of a monitor? ....................................... 232
How can I set a monitor's setting from gray to color? .................................................. 233
Why does SetDepth() from B/W produce grays instead of colors? ................................. 233
If my window overlaps multiple screens with different depths, how can I draw the contents of the window properly on all screens? ................................................................. 234
How can I determine which screens my window overlaps and which screen the majority of my window exists on? ................................................................. 234

Chapter 13: Icons ........................................................................................................... 237
What are an icon family and icon suite? ................................................................. 237
Which icon types can I put in dialogs and menus? ..................................................... 238
What types of icons can the Finder display? ............................................................. 238
How can I draw an icon as it appears in the Finder in the different Finder states? ........ 239
Why does only a part of my icon draw? ................................................................. 240
How can I create different masks for the icons in an icon family that are the same size but of different bit depths? ................................................................. 240
Why doesn't PlotCIcon() get recorded as part of a picture by OpenPicture()/ClosePicture()? ................................................................. 240
How can I speed up displaying a 'cicn' icon for animation (PlotCIcon() is too slow)? ...... 241

Chapter 14: Lists and the List Manager ........................................................................ 243
What is an LDEF? ........................................................................................................ 244
What is a list's click loop? ......................................................................................... 244
How do I detect a double-click on a cell? ................................................................. 245
Why do the strings in my list have unrecognizable characters in front of them? ............ 245
Is there a limit to the amount of data a list can contain? ........................................... 245
I make a series of changes to a list, it flashes. How can I prevent this? ......................... 245
How can I have my list prevent the selection of multiple cells? ................................. 246
Why won't my list draw or reflect changes? ............................................................... 247
How do I put a list in a Macintosh modal dialog box? .................................................. 247
Can different columns in a list have different widths? .................................................. 250
Can I have a list of more than one type of data (such as strings and icons)? ................... 251
When I create a list, it has no border around it. Is there a way to make a border come up automatically, and if not, how do I draw one? .................................................. 251
How can I change the size and style of text in my lists? ............................................... 251
How do I make a list of icons or pictures? ................................................................. 252
How do I get the size of the current display area of a list? ......................................... 253
How can I test if a cell is selected, and how do I get all of the currently selected cells? .... 253
How can I tell which cells in my list are visible? .......................................................... 254
How can I tell how many rows and columns are in a list? ........................................... 254
How do I install a click loop in my list? ................................................................. 255
Why won't my list's click loop work? ................................................................. 256
Table of Contents

How can I update a text field to display information about a cell in a list when the mouse drags over it during LClick()? ................................................................. 257

Chapter 15: Memory ........................................................................................................ 259

What is a handle? ............................................................................................................. 260
What is temporary memory? ......................................................................................... 261
What is the System heap? ............................................................................................. 261
What is the A5 world? .................................................................................................. 261
If I lock a locked handle and then call HUnlock(), will the handle be unlocked, or do I need to balance HLocks() with HUnlocks()? ............................................. 262
Can I just pass a pointer to a memory block pointer where I need a handle? .......... 262
How can I tell whether a handle was previously locked before I locked it, so I know whether or not to unlock it when I'm done with it? .................................. 263
How can I tell if a purgeable block is still valid? ......................................................... 263
When should I lock a handle? ..................................................................................... 263
Can I change the size of my handle? ........................................................................... 264
What is the best way to copy data from one handle to another handle? ................. 264
Why should I use handles instead of pointers? ......................................................... 264
How can I get a handle to a block of zeroed-out memory? ....................................... 265
How can I get the size of the data referenced by a handle or a pointer? .................. 265
How can I tell how much memory is available to my application? ......................... 266
How do I allocate memory in the System heap instead of the application heap? ...... 266
When my program ends, will it free up all memory that it has allocated? ............... 267
How do I access the A5 world of my application from a function called at interrupt time? 267
When do I need to explicitly set up and restore the A5 world? ................................. 268
What is the A4 register used for in code resources? ................................................. 269
How do I get temporary memory? ............................................................................. 270
What can I do when I need memory but can't get it? ................................................. 270
Can I access more memory than is allocated in my application heap? .................. 270

Chapter 16: Menus ...................................................................................................... 271

What is a hierarchical menu? ...................................................................................... 272
What are MenuHook and MBarHook? .................................................................... 272
How do I add menus to a program? ......................................................................... 273
How do I put the Apple menu in my application? .................................................... 275
How do I add the application menu to my application's menu bar? ....................... 277
How do I put a divider in my menu? ....................................................................... 277
How do I put an icon in my menu next to the menu item text like the Apple menu? .. 277
Why can't a menu item have a small icon and a Command-key equivalent? ........... 278
How do I dim or disable a menu item or an entire menu? ....................................... 279
How can I tell if a menu or menu item is disabled? ................................................. 279
When should I put ellipsis (...) at the end of a menu item? ...................................... 280
How do I put a check mark next to a menu item? ..................................................... 280
How many menu items can I have in a menu? ......................................................... 281
Why can't I individually enable or disable any menu past menu item position 31? .. 281
How can I tell how many items are in a menu? ......................................................... 282
Macworld Mac Programming FAQs

Chapter 17: Power Macintosh Programming ........................................ 305

Chapter 18: Printing ................................................................. 315
Table of Contents

Chapter 19: Processes: Notification Manager ........................................... 327
How can my application get the user's attention when it is in the background? 328
How can my application display a dialog box in front of all other windows on the screen when it is running in the background? 328
How do I issue a notification to the user from an application in the background? 329
When I have my notification set to notify the user by an alert or by playing a 'snd' resource, it works fine. However, when I set it to notify the user by marking the application menu or blinking an icon, it doesn't seem to work. Why? 331
How can I tell if my notification was posted? ........................................ 331
How can I get the Notification Manager to blink a color icon? ................. 331
When do I remove a notification record from the notification queue? ......... 331
Can I remove a notification I have installed before it is issued by the Notification Manager? ................................................................. 333
Are issued notification requests automatically removed from the queue? .... 334
Can I send a notification to a user from a interrupt task such as a VBL task or a Time Manager task? ............................................................... 334
Why does my program crash when it tries to issue a notification to the user? 334
How do I get a notification to execute a response procedure? ..................... 335
Why does my machine hang when my application's notification procedure is activated a second time? The first time it is activated it runs fine. .... 336
How do I access my application's global variables from a notification procedure? 336

Chapter 20: Processes: Process Manager ................................................. 339
What is an interrupt? What is interrupt time? ........................................... 340
What is the difference between the current process and the front process? 340
What are background-only applications? ............................................... 340
How can I tell if my application is the front process? .............................. 341
How do I get a list of all of the processes currently running? .................... 342
How can I find out if a specific process is running on my system and if it supports high-level events? ................................................ 342
How can I get the name and location of my application? .......................... 343
Why won't my application run when it is in the background? .................... 344
How can I get my application to be notified when an application that it launches with LaunchApplication() terminates? ................................. 345
Why does my application terminate after I call LaunchApplication()? ....... 345
How do I launch another application from my application? ............................................. 345
How do I launch an application with LaunchApplication(), but prevent it from
becoming the front application? ................................................................................... 347

Chapter 21: Processes: Time Manager ................................................................. 349
When do I use VBL tasks, and when do I use the Time Manager? .......................... 350
What is the finest resolution of time delay in Time Manager? ................................. 351
How do I specify a period of less than one millisecond? .......................................... 352
How can I determine the amount of time it takes for a routine to run? .................... 352
What is the difference between InsTime() and InsXTime()? ................................... 354
Why does my system crash when I exit my application after installing a periodic Time
Manager task? .............................................................................................................. 354
How do I get a Time Manager task to execute persistently, execute when the application
that installed it is switched out and is no longer in control of the CPU? ........................ 355
How do I execute a Time Manager task in a 68K application? ................................. 355
How do I execute a Time Manager task in a native PowerPC application? ............... 357
Do I need to reinstall a Task Manager task with InsTime each time I want activate it
with PrimeTime? .......................................................................................................... 358
How do I get my Time Manager task to run periodically? ....................................... 359
How do I leave a Time Manager task running after my application quits? .................. 359
How do I stop a periodic Time Manager task from running? ................................... 359
How do I access application globals from my 68K Time Manager task? ................. 360
How do I access application globals from a native PowerPC Time Manager task? .... 361

Chapter 22: Processes: Vertical Retrace Manager .................................................. 363
What is a Vertical Blanking (VBL) interrupt? ............................................................ 364
What is the difference between a system-based VBL and a slot-based VBL? ............... 364
How often will a VBL task get executed? .................................................................. 365
How do I execute a 68K VBL task? ........................................................................... 365
How do I execute a PowerPC native VBL task? ...................................................... 367
How do I execute a slot-based VBL task? ............................................................... 368
How can I get my VBL task to execute after my application quits? ........................... 369
How do I get a VBL task to execute persistently, execute when the application is
switched out and is no longer in control of the CPU? ............................................... 369
How do I make my VBL task run periodically? ....................................................... 371
How do I access application globals from my 68K VBL task? ................................. 372
How do I access application globals from a native PowerPC VBL task? .................. 373
How do I determine the slot number of a screen? .................................................... 374
How do I get the slot number for the main monitor? ............................................... 374
How can I disable a VBL task? ................................................................................. 375
How do I synchronize my drawing to a VBL for smooth animation? ....................... 375

Chapter 23: QuickDraw: Drawing ................................................................. 377
What is a GWorld? ....................................................................................................... 378
What is the gray region? .............................................................................................. 378
### Table of Contents

Chapter 24: QuickDraw: Pictures ................................................................. 425

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are the QuickDraw “bottleneck” functions?</td>
<td>378</td>
</tr>
<tr>
<td>What system initialization do I have to perform before my program can run?</td>
<td>380</td>
</tr>
<tr>
<td>Why can’t I access QuickDraw globals like ‘thePort’ anymore?</td>
<td>381</td>
</tr>
<tr>
<td>What is the difference between highlighting and inverting?</td>
<td>382</td>
</tr>
<tr>
<td>How do I highlight an object?</td>
<td>383</td>
</tr>
<tr>
<td>How do I draw lines and shape outlines of different widths?</td>
<td>384</td>
</tr>
<tr>
<td>How do I draw dashed lines?</td>
<td>385</td>
</tr>
<tr>
<td>How do I draw a shape filled with a solid color?</td>
<td>386</td>
</tr>
<tr>
<td>How do I set the pen pattern with universal headers?</td>
<td>386</td>
</tr>
<tr>
<td>How can I draw in a pattern?</td>
<td>387</td>
</tr>
<tr>
<td>How do I get the standard patterns?</td>
<td>388</td>
</tr>
<tr>
<td>How can I draw in a color pattern?</td>
<td>389</td>
</tr>
<tr>
<td>How can I get and set the RGB value of a single pixel?</td>
<td>390</td>
</tr>
<tr>
<td>Why does my drawing trash the screen whenever I draw in response to an update event for a window that is not the frontmost window?</td>
<td>391</td>
</tr>
<tr>
<td>How do I flip a pixmap (put up its mirror image)?</td>
<td>392</td>
</tr>
<tr>
<td>What can I do to speed up CopyBits()?</td>
<td>396</td>
</tr>
<tr>
<td>How do I rotate a pixmap?</td>
<td>397</td>
</tr>
<tr>
<td>How do I get the most frequently used colors in a pixmap?</td>
<td>398</td>
</tr>
<tr>
<td>Why is the rowBytes field of my pixmap a negative number?</td>
<td>398</td>
</tr>
<tr>
<td>Why do the colors in my image copied with CopyBits() get distorted?</td>
<td>399</td>
</tr>
<tr>
<td>How can I tell if my structure is a bitmap or a pixmap?</td>
<td>399</td>
</tr>
<tr>
<td>Can I pass a pixmap of a color window to CopyBits(), which takes a bitmap as a parameter?</td>
<td>400</td>
</tr>
<tr>
<td>Can CopyBits() copy overlapping rectangles in the same window?</td>
<td>400</td>
</tr>
<tr>
<td>How do I draw in my window using a GWorld?</td>
<td>401</td>
</tr>
<tr>
<td>Why is the rowBytes field of my GWorld’s pixmap larger than what I defined? How much memory should I allocate for my GWorld buffer?</td>
<td>403</td>
</tr>
<tr>
<td>How do I get the address of the pixels of a GWorld’s pixmap?</td>
<td>404</td>
</tr>
<tr>
<td>How do I put contents of a GWorld into a picture?</td>
<td>404</td>
</tr>
<tr>
<td>Why do my colors get distorted when I use CopyBits() from my GWorld?</td>
<td>406</td>
</tr>
<tr>
<td>Why do I get garbage when I use CopyBits() from an offscreen bitmap or a GWorld?</td>
<td>406</td>
</tr>
<tr>
<td>How can I use my own colors in a GWorld?</td>
<td>406</td>
</tr>
<tr>
<td>How do I draw ‘rubber banding’ lines (anchored lines and selection rectangles)?</td>
<td>408</td>
</tr>
<tr>
<td>How can I draw ‘marching ants’ marque?</td>
<td>410</td>
</tr>
<tr>
<td>How can I make a selection rectangle in color?</td>
<td>412</td>
</tr>
<tr>
<td>How do I implement a ZoomRect (a thin gray rectangle that “zooms out” when a window opens or closes)?</td>
<td>414</td>
</tr>
<tr>
<td>How can I make text or an object appear dimmed or grayed-out without redrawing it?</td>
<td>416</td>
</tr>
<tr>
<td>Is there a way to draw outside a window on the Desktop?</td>
<td>417</td>
</tr>
<tr>
<td>How can I replace the standard QuickDraw low-level or “bottleneck” functions?</td>
<td>419</td>
</tr>
<tr>
<td>Are the QuickDraw “bottleneck” functions called at interrupt time?</td>
<td>424</td>
</tr>
<tr>
<td>What is a picture?</td>
<td>426</td>
</tr>
</tbody>
</table>
How do I create a picture? ................................................................................................ 426
If I want to record a picture, do I have to do it offscreen so it won't disrupt my windows? 428
How can I combine two pictures into one? ........................................................................ 428
How can I create a picture from a pixmap? ........................................................................ 429
Why do pictures I record sometimes draw as empty space? ............................................... 429
How can I determine if the large picture I am recording with OpenCPicture() recorded
  successfully or if I ran out of memory? ........................................................................... 430
Can I interrupt picture recording, execute some other calls, and then continue recording? 431
Why can't I put CopyMask() in my picture? What other routines don't work in picture
  definitions? ................................................................................................................... 432
How do I create a picture of the contents of a window? ..................................................... 432
How can I control the resolution of a picture that I create? ................................................ 433
To destroy a PicHandle, when do I call KillPicture(), ReleaseResource(), and
  DisposeHandle()? ......................................................................................................... 434
How can I draw a picture with a transparent background instead of a white background? 435
How can I draw a picture in a mode other than srcCopy mode? ......................................... 437
How can I center a picture in a window? ........................................................................... 437
How can my application process picture comments when drawing a picture? .................... 438
How do I rotate a picture? ................................................................................................ 439
What is the format of a PICT file? .................................................................................... 439
How do I write a picture to a PICT file? ........................................................................... 440
How do I read a picture from a PICT file? ......................................................................... 441
How do I get a palette or color table of a picture? ............................................................ 442
How do I get the size (bounding rectangle) of a picture? ................................................... 443

Chapter 25: Resources ..................................................................................................... 445

What is the resource map? ............................................................................................. 446
Are resource types case sensitive? .................................................................................. 446
How do I put a code resource in my application? ............................................................ 446
What does detaching a resource do, and when should I do it? ........................................... 447
How do I load a resource into the System heap? ............................................................. 447
How can I get the machine name and user name? ........................................................... 449
What are the valid resource IDs that I can use for my resources in my application? .......... 449
The Resource Manager functions do not return a value; how do I tell if they were
  successful or if an error occurred? ................................................................................ 449
How can I determine the order that resources will be returned by GetIndResource()? ...... 449
How can I tell if a purgeable resource has been purged? ................................................. 450
How do I update a Macintosh resource? ......................................................................... 450
What does the "1" stand for in the Resource Manager routines? ....................................... 451
Why can't I open the resource fork of a file I just created with FSpCreate()? .................... 451
How do I save a resource to a file? .................................................................................. 451
How do I copy resources from one resource file to another? ......................................... 452
How can I set the current resource file? .......................................................................... 453
How do I access resources from another file? ............................................................... 453
How can an application get the file reference number of its own resource file? ............. 453
Given a handle to a resource, how can I determine which resource file it came from? ...... 454
If I have multiple resource files open, what is the order in which they are searched for a resource? .............................................................................................................. 455
How can I get the number of resources of a particular type in a resource file? ........ 456
How can I limit the searching for a resource to check only a single file? ............... 456
Is there a maximum number of items for Macintosh resources in a resource file? .. 456
How does an application access the resources in its own resource fork? In the System file? ........................................................................................................................................ 457

Chapter 26: Scrap Manager ............................................................................. 459
What is the “scrap”? .......................................................................................... 460
How can I tell if there is data in the scrap? ....................................................... 460
How can I tell what types of data are in the scrap? ........................................... 460
How can I determine if a certain type of data is in the desk scrap? ................. 462
How can I tell if the data in the scrap has been changed by another application? 462
How do I determine the size of the handle to pass to GetScrap()? ....................... 462
What types of data can the desk scrap hold? .................................................... 463
How do I read data from and write data to the scrap? ...................................... 463
When data is put on the scrap, does it replace the data that is already there? .... 463
Why doesn’t the data I copy onto the scrap get retrieved when I paste? Why does only the old data get pasted? ................................................. 464
How do I clear data from the scrap? ................................................................. 464
How can I read all of the data from the desk scrap, including more than one item of the same type? ................................................................. 464
When does the scrap get copied to a file? .......................................................... 465
How can I tell if the scrap is in memory or has been written to a file? ............... 466
Why doesn’t GetScrap() and PutScrap() from a TextEdit field or DialogCopy() from a dialog affect the desk scrap? ................................................. 466

Chapter 27: Sound .......................................................................................... 467
What is double buffering? .................................................................................. 468
What are sound commands? ............................................................................. 468
How do I play a ‘snd ’ resource? ............................................................... 469
How do I play large sounds that won’t fit in memory? ..................................... 472
How do I play a sound continuously from disk, either a large ‘snd ’ resource or an AIFF file? ................................................................. 472
How can I begin playing a sound from a point other than the beginning? .... 474
How do I play more than one sound at a time? ............................................. 474
How many sound channels can play sounds concurrently? ............................ 475
How do I stop a sound that is playing? .......................................................... 475
How can I change the playback rate, to slow down or speed up the sound playing? 477
How do I install a sound and play different notes with it? ............................... 478
Why doesn’t my sound play asynchronously? .............................................. 478
How do I set up callback and completion routines for playing asynchronous sounds? 479
How can I tell when a sound playing asynchronously has completed playing? 479
Why do I get static when I play a sound? ....................................................... 483
How can I synchronize the playing of sound by sound channels? .................................................. 483
How can I get and set the volume of my computer? ................................................................ 485
How can I adjust the left and right volumes, like a balance control (panning)? ......................... 488
How do I adjust the volume at which a channel plays a sound? ................................................ 488
Why does my volume go down every time another channel plays a sound? ............................... 491
How do I record sound in my application? ................................................................................. 491
How do I record a sound without the standard recording dialog? .............................................. 492
How do I create a 'snd' resource from sampled data? ............................................................... 495
How do I create an AIFF file from sampled data? ...................................................................... 497
How can I determine how long an asynchronous sound has been playing or still has remaining until completion? ............................................................................................ 500
How can I record sound asynchronously? .................................................................................. 501
How do I execute a sound command? ......................................................................................... 504
How can I store more than 128 commands in a sound channel? ............................................... 505
How do I unlock and release a sound resource and dispose of a sound channel when I play a sound asynchronously? ........................................................................................................ 506
Can I dispose of my sound handle and sound channel in my completion routine? .................... 506
How can I calculate the sample rate of a sound playing on a sound channel from the value returned from the getRateCmd sound command? ......................................................... 507
Why can’t I get a ‘snd’ resource? I know it is in my resource file. ............................................. 507
When is the callback routine for my sound channel executed? .................................................. 507

Chapter 28: Speech Manager .................................................................................................. 509
How can I have Speech Manager speak a buffer of text? .......................................................... 509
How can I stop a channel from speaking once it starts synthesizing text? ................................. 510
How can I tell when my speech channel has completed speaking text? ...................................... 510
How can I get a list of all of the available voices? ................................................................. 511
How can I select a voice other than the default voice to speak? ............................................. 511
How can I synchronize speech from a speech channel with other actions? .............................. 512

Chapter 29: Strings, Bit Manipulation, and Random Numbers ............................................. 515
How do I convert a C string to a Pascal string and vice versa? .................................................. 516
How can I convert between a Str255 and an OSType? ............................................................. 517
How do I get the handle to a string? ......................................................................................... 517
How do I convert a string to all uppercase or all lowercase? .................................................... 518
Where can I find functions like strcpy and strcat for Pascal strings? ...................................... 518
How do I convert strings to numbers and numbers to strings? .............................................. 519
How can I test if a single bit is set? How can I set a single bit? ................................................ 520
Random() returns values from -32767 to 32767. How can I get random numbers in a smaller specified range? ........................................................................................................ 522
How are random numbers generated on the Mac? .................................................................... 522
Why aren’t the numbers generated by Random() random? ................................................... 522
How can I get random numbers greater than 32767? ............................................................ 523
Chapter 30: System Extensions and Patching Traps ................................................. 525
What is a trap? ........................................................................................................... 526
What is the Unimplemented() routine? ................................................................. 527
What is a System extension (‘INIT’)? ................................................................. 528
What is the difference between a head patch and a tail patch? ............................... 528
How do I create a System extension? ................................................................. 529
How can my System extension tell when INIT time is over? .................................... 531
How do I get my System extension to show its icon at startup? ............................ 531
How can I control the execution of my System extension? ....................................... 532
How do I get my extension and control panel to communicate or share data? .......... 532
Why do I have problems getting a PicHandle at INIT time? ................................. 533
What does patching a trap mean? ........................................................................... 533
How do I patch a trap? .......................................................................................... 534
Where can I find the trap numbers for routines? .................................................... 537
How do I get the address of a trap? ....................................................................... 537
How can I tell if a System routine is available in my System? ................................. 538
How can I tell if a routine is an Operating System or a Toolbox trap? ................. 540
If I patch a trap from my application, will it affect all applications? ....................... 541
How can I have a patch to a trap and have it affect all applications? ...................... 542
Can two applications patch the same trap at the same time? ............................... 542
Can my patch routines move memory? .................................................................. 543
How can I examine keydown and mouse events in all applications? ....................... 543

Chapter 31: Text: Fonts and Drawing ..................................................................... 545
What are the System and the application fonts? .................................................... 546
What is a font family? ............................................................................................ 546
What is the difference between an outlined font and a bitmapped font? ............... 546
How do I get the font family ID of a font from the name of the font? ...................... 547
How do I get the name of a font family from the font family ID? .......................... 548
How can I get all available fonts in the System? .................................................. 548
How can I tell if a graphics port will use a bitmapped font or an outline font? ....... 548
Why does text drawn with a font ID sometimes appear different on different machines? 548
How do I determine the height in pixels of a font? .............................................. 549
How do I draw text in my window? ..................................................................... 549
How do I change the font, style, and size of the text that gets drawn? ...................... 550
How can I determine the current text characteristics (font, size, mode, and style) that a graphics port will use to draw text? .............................................................................. 551
Is there any way to determine the length of styled text that is drawn given the font, size, style, etc? ........................................................................................................ 552
Can I use transfer modes when drawing text? .................................................... 552
Why doesn’t the text that I draw respond to the mode settings I select with PenMode? 552
How can I get the current textMode, so I can change it and then restore it when I’m done? ............................................................................................................ 553
How do I draw dimmed (grayed-out) text? .......................................................... 553
How can I draw rotated text? .............................................................................. 554
Chapter 32: Text: TextEdit ................................................................. 561

How can I draw a block of text in a specified rectangular area? ......................... 562
How can I display static text of more than 255 characters? .............................. 562
How do I get the text of a TextEdit field? ...................................................... 563
How do I get the text and the style from a multistyle TE field? ......................... 563
How can I determine if a TextEdit field is an old type TextEdit field or a style-aware
TextEdit field? ......................................................................................... 565
How can I determine the line height of a TextEdit field? .................................. 565
How do I process a mouse click in a TextEdit field? ........................................ 566
How do I get the style of the text in a TextEdit field? ........................................ 566
How can I get the start of each line in my TE field? ......................................... 567
Why doesn’t changing the sizes of my destination and view rectangles affect the word
wrapping in my TextEdit field? .................................................................. 567
Why doesn’t my text change when I call TESetText? ........................................ 567
How can I draw text with TETextBox() so that the background behind the text is
maintained allowing me to draw over a picture? ............................................. 567
How do I get TextEdit to display more than 32K of text? ................................. 568
How do I put scroll bars in my TextEdit field? ............................................... 568
How do I update the scroll bars in response to automatic scrolling caused by dragging
outside the window? ................................................................................... 569
How do I set the selection range in a TextEdit field? ...................................... 570
How do I set the cursor position in a TE field? ............................................. 571
Why won’t my TE caret blink? ...................................................................... 571
How can I modify the caret blinking rate of the TextEditing insertion point caret?........ 571

Chapter 33: Windows ................................................................. 573

What is the difference between local and global coordinates? ......................... 574
What is a window definition procedure (‘WDEF’)? ......................................... 575
How do I set the appearance of a window? ................................................. 576
How do I find the position of my window on the screen? ............................... 577
How do I get the current size of my window? ............................................. 577
How can I get all of the open windows in my application? Is there a list of all open
windows somewhere? ............................................................................... 578
How do I center a window on the screen? ................................................. 579
How do I get the top window of my application? ......................................... 580
What effect does hiding a window have on the window’s position in the application’s
window list? ......................................................................................... 580
How do I open a new window behind an already open window in my application? .... 580
How do I create a floating window or palette, a window that always stays on top of
my windows, in my program? ................................................................. 581
How can I tell when a window becomes activated or deactivated? ..................... 581
Is it possible to have windows of shapes other than the standard rectangular shape? .... 588
Is the origin of a window the upper-left corner of the window, or the upper-left
corner of the drawing area, below the title bar? ........................................ 589
How can I force my window to redraw when I want to make a change? ............ 589
Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can I set up custom colors in my window?</td>
<td>590</td>
</tr>
<tr>
<td>How do I get the GrafPort of the desktop?</td>
<td>591</td>
</tr>
<tr>
<td>Why can I only draw in the eight basic colors in my window?</td>
<td>592</td>
</tr>
<tr>
<td>How can I copy the contents of one window to another?</td>
<td>592</td>
</tr>
<tr>
<td>When I call SelectWindow() to activate a window I cannot always draw to it. My drawing affects the window that was previously the active window even though the new one is now on top and active. Why?</td>
<td>593</td>
</tr>
<tr>
<td>How can I tell what part of my window needs to be redrawn in response to an update event?</td>
<td>593</td>
</tr>
<tr>
<td>If there are multiple windows in my application, how do I select which windows to draw to?</td>
<td>593</td>
</tr>
<tr>
<td>How can I put a color background in my window?</td>
<td>593</td>
</tr>
<tr>
<td>How do I draw the grow box in my window?</td>
<td>594</td>
</tr>
<tr>
<td>How can my application determine a window title bar’s height in pixels?</td>
<td>595</td>
</tr>
<tr>
<td>Why don’t my scroll bars get refreshed when my window comes to the front?</td>
<td>595</td>
</tr>
<tr>
<td>How do I draw a Macintosh grow box without the scroll bars?</td>
<td>596</td>
</tr>
<tr>
<td>How do I prevent scroll bars from getting drawn on?</td>
<td>596</td>
</tr>
<tr>
<td>I want to access fields in a window’s WindowRecord structure but all I have are WindowPtrs which are pointers to GrafPorts. How can I access a window’s WindowRecord from a WindowPtr?</td>
<td>597</td>
</tr>
<tr>
<td>Can I store my own data in the refCon field of a window or dialog?</td>
<td>598</td>
</tr>
<tr>
<td>How can I attach data to a window structure for my own use?</td>
<td>598</td>
</tr>
<tr>
<td>Appendix A: Using the Companion CD-ROM</td>
<td>601</td>
</tr>
<tr>
<td>Appendix B: Gestalt Manager Selector Codes</td>
<td>609</td>
</tr>
<tr>
<td>Index</td>
<td>631</td>
</tr>
<tr>
<td>CD-ROM Disc Instructions</td>
<td>641</td>
</tr>
<tr>
<td>License Agreement</td>
<td>643</td>
</tr>
<tr>
<td>Reader Response Card</td>
<td>Back of Book</td>
</tr>
</tbody>
</table>
Macworld Mac Programming FAQs
Apple Event Manager

Due to the enormous size and complexity of the Inside Macintosh reference material, Apple events have become a notoriously difficult topic for Mac programmers to master. As a way to decrease the learning curve, this chapter answers questions relating to Apple events. You'll learn where to go to get started with Apple events, the basic Apple events that every Mac application should support, and more.

What are the four required Apple events?

A high-level event is an event that one application can send to another to provide interapplication communication capabilities. An Apple event is a special type of high-level event that follows the Apple Event Interprocess Messaging Protocol (AEIMP). The Finder uses Apple events to communicate with applications. It can request that the application perform such tasks as open or print a document. Every Apple event has both a class, or suite, and an ID. The class defines the event's basic category. The ID distinguishes the event from other Apple events in the same suite. The most commonly supported suite is the Required suite. It contains four required Apple events that every application should support.

The four required Apple events are “open application,” “open document,” “print document,” and “quit application.” All of your Macintosh applications should support these four events, even if you don’t intend to make your applications scriptable or recordable. Each of these four events has an associated four-character code.
The "open application" event is also known as 'oapp'. Your application will receive an 'oapp' event when it starts up. Your 'oapp' handler, which you installed using AEInstallEventHandler(), should respond to the event by doing any application-specific initialization.

The "open document" event is also known as 'odoc'. Your application gets an 'odoc' event from the Finder whenever the user asks the Finder to open a document or set of documents. If your application is recordable (as it should be), it will send an 'odoc' to itself when the user opens a file using your application. Your 'odoc' handler will receive the 'odoc' Apple event as a parameter and will retrieve the list of documents from the event using AECountItems() to count the number of files in the event and AEGetNthPtr() to retrieve the information for a specific document. The handler will then open each document in turn, until they are all opened.

The "print document" event is also known as 'pdoc'. Your application gets a 'pdoc' event in much the same way as it gets an 'odoc' event. If the user selects a document (or list of documents) to print in the Finder, the Finder will package information about the document(s) in an Apple event, and your 'pdoc' handler will print the document. If the user selects Print from your File menu, you should send a 'pdoc' event to yourself, asking your handler to print the document(s).

The "quit application" event is also known as 'quit'. At the very least, your application will get a 'quit' event when the user shuts down the computer with your application still open. Your 'quit' handler should do whatever is necessary to get your application to exit. If your application uses a global flag to determine if it should drop out of the main event loop, your 'quit' handler can just set the flag to the "drop out of the loop" value. Be sure you prompt the user to save all changes and that you respond correctly if the user elects to "Cancel" the quit.

The constants for the four required events are kAEOpenApplication, kAEOpenDocuments, kAEPrintDocuments, and kAEQuitApplication, which correspond to 'oapp', 'odoc', 'pdoc', and 'quit', respectively.

Related FAQs

- See also FAQ 1-4, "How do I add support for the required Apple events in my program?"

- See also FAQ 1-5, "How does the Finder tell my application that one of its documents was double-clicked?"
How do I learn how to program with Apple events?

There is a lot of reading material to go through to really learn about Apple events. The Apple events bible is *Inside Macintosh: Interapplication Communication*. Unfortunately, the book is about 1000 pages long and very hard to understand. If you want to program using Apple events, however, you’d best get yourself a copy.

Next, check out *Ultimate Mac Programming* by Dave Mark. The first few chapters offer a good introduction to Apple events with some small sample programs that are pretty easy to understand. If you have access to back issues of *MacTech Magazine* or the *MacTech* CD-ROM, you’ll find a number of articles on Apple event programming. (Some of these articles are also included as appendices in the back of *Ultimate Mac Programming*.)

The control panel/application combination called AETracker, by C. K. Haun, gives you a lot of information about the Apple events you are building. In addition, there are some MacsBug ‘dcmd’s that let you debug your Apple events at a lower level.

There is also a boatload of excellent source code out there to claw through and cannibalize. An especially rich source of good code is the PowerPlant framework, which has built-in support for building scriptable, attachable, and recordable applications.

As you learn about Apple events, you’ll need a copy of the *Apple Events Registry*, which describes the different Apple event suites that are supported by the Macintosh programming community. Though a paper copy is nice, an electronic version of the Registry is far preferable (you can search it and copy snippets from it).

How can I determine what Apple events an application supports?

To determine the Apple events supported by an application, you’ll need to examine the application’s ‘aete’ resource. You can do this using Resorcerer (ResEdit’s ‘aete’ editor is not very informative) or, if you don’t have access to Resorcerer, you can use the Script Editor that comes with AppleScript. (Be sure you have both AppleScript and the Script Editor installed on your machine.)

Inside the Script Editor, select Open Dictionary from the File menu. You’ll be prompted for an application to open. The resulting window will list the Apple event suites supported by this application, as well as the supported objects and attributes within each suite.
One way to get to know an application’s ‘aete’ resource is to use the Script Editor to write some experimental scripts that manipulate the application. If you are going to program using Apple events, you will find it helpful to learn an OSA-compatible scripting language such as AppleScript or UserTalk.

How do I add support for the required Apple events in my program?

Your application, at a minimum, should support the four required Apple events (kAEOpenApplication, kAEOpenDocuments, kAEPrintDocuments, and kAEQuitApplication). The first thing your application must do to support these is tell the Event Manager which Apple events you plan to handle and which routines you plan to handle them with. Once you assign a routine to an Apple event, the Event Manager takes over and automatically calls the routine whenever that Apple event occurs. The routine AEInstallEventHandler() is used to associate an Apple event with its handler, as follows:

```c
AEEventHandlerUPP gDoOpenAppUPP;

gDoOpenAppUPP = NewAEEventHandlerProc( DoOpenApp );
err = AEInstallEventHandler( kCoreEventClass,
                           kAEOpenApplication,
                           gDoOpenAppUPP, OL, false );
```

The first parameter of AEInstallEventHandler() is the event class of the Apple event. The required Apple events are all members of the core class.

Once you’ve installed your Apple event handlers, your program will enter the main event loop. Whenever an Apple event occurs, the Event Manager places the constant kHighLevelEvent in event. When this happens, all you have to do is pass the event on to the routine AEProcessAppleEvent(), which will automatically call your handler, as follows:

```c
switch ( eventPtr->what )
{
    case kHighLevelEvent:
        DrawEventString( "\pHigh level event: " );
        AEProcessAppleEvent( eventPtr );
        break;
}
```
An Apple event handler routine has three parameters. The first two are of type Apple event and the third is a long. The handler of kAEOpenApplication starts the application by opening a window. This handler is defined as follows:

```pascal
OSErr DoOpenApp( AppleEvent theAppleEvent,
                  AppleEvent reply, long refCon )
{
    WindowPtr window;
    window = CreateWindow();
    if ( window == nil )
        return( -1 );
    ShowWindow( window );
    return( noErr );
}
```

Related FAQs

- See also FAQ 1-1, “What are the four required Apple events?”
- See also FAQ 1-5, “How does the Finder tell my application that one of its documents was double-clicked?”

**How does the Finder tell my application that one of its documents was double-clicked?**

In the olden days, when an application started up, it would call some Toolbox routines to see if there were any documents it needed to open. Once it passed that chunk of initialization code, there was no way for the Finder to tell it to open another document.

Nowadays, the Finder communicates with your application using Apple events. For example, suppose you wrote a word processor called DickyWrite. When the user double-clicks on a DickyWrite document in the Finder, the Finder packages a description of the document in an “open document” Apple event and sends the Apple event to DickyWrite. When DickyWrite started up, one of the first things it did was to tell the Apple Event Manager which of its routines to send to which Apple events. For example, DickyWrite designated one of its functions (known as a
“handler”) specifically to handle “open document” events. When the Finder sends the “open document” event to DickyWrite, the event goes right to DickyWrite’s “open document” handler and the handler opens the document.

If you want your application to be recordable, you'll need to learn how to send Apple events, just like the Finder does. For example, when the user selects Open from your application’s File menu, you’ll first prompt the user to select a document to open. Once you know the document that needs to be opened, embed the information describing the document in an Apple event and send an “open document” event to yourself. Your “open document” handler will receive the event and open the document. Since the document was opened as a result of an “open document” Apple event, the act of opening the document will be recorded and captured in a script (if recording is turned on).

Related FAQs

☐ See also FAQ 1-1, “What are the four required Apple events?”

☐ See also FAQ 1-4, “How do I add support for the required Apple events in my program?”

Related Topic

☐ See also Chapter 10, “Finder and Desktop.”

**How can I identify the sender of an Apple event?**

To find the target ID or process serial number of the sender of an Apple event, use AEGetAttributePtr() to extract the address attribute from the received Apple event, as follows:

```c
Size            bufSize, actualSize;
DescType        retrievedType;
TargetID        retrievedData;

bufSize = sizeof(retrievedData);
AEGetAttributePtr( receivedAppleEvent, keyAddressAttr,
                   typeWildCard, &retrievedType, (Ptr)(&retrievedData),
                   bufSize, &actualSize);
```

The second parameter, keyAddressAttr, specifies that you are interested in retrieving an attribute containing the address descriptor for the sender. The third parameter specifies that you don’t care if the retrieved descriptor is of the form
TargetID or ProcessSerialNumber. If you want to retrieve a TargetID only, use typeTargetID instead of typeWildCard.

You can get the process serial number from the target ID of the sender of an Apple event by passing the port name (the name field of the TargetID structure) to the GetProcessSerialNumberFromPortName() function. Once you get the serial number, you can call GetProcessInformation() to get information about the process.

The retrieved Data buffer can later be used with AECreateDesc to create an address to be passed to AESend.

Can an application send Apple events to itself?

Yes. In fact, the ability to send an Apple event to itself is vital in order for an application to be recordable. If your application is recordable, users will be able to record all of their actions as a sequence of Apple events using a tool such as the Script Editor. For example, users of a recordable word processor might record a sequence where they open a document, change all occurrences of the Palatino font to Bookman, then save the changes. They might then save the sequence of Apple events generated by these actions as an AppleScript script and rerun this sequence, this time on a whole series of files to which they'd like the same changes to apply.

To send an Apple event to itself, the application uses AECreateAppleEvent(), passing in an AEAddressDesc set up to point to itself. Here's some code that sets up the AEAddressDesc:

```c
ProcessSerialNumber thePSN;
AEDesc theTarget;

thePSN.highLongOfPSN = OL;
thePSN.lowLongOfPSN = kCurrentProcess;
theErr = AECreateDesc( typeProcessSerialNumber,

  (Ptr)&thePSN, sizeof(thePSN), &theTarget);
```

To learn more about creating a recordable application, check out the code in Chapter 2 of Ultimate Mac Programming, as well as the DocDemo PowerPlant program that ships with CodeWarrior.

Related FAQ

See also FAQ 1-8, "How can I prevent an Apple event that my application sends to itself from getting recorded?"
**How can I prevent an Apple event that my application sends to itself from getting recorded?**

By default, all Apple events that an application sends itself are recorded when Apple event recording is turned on. This can be prevented by specifying the constant kAEDontRecord in the sendMode parameter of AESend().

**Related FAQ**

See also FAQ 1-7, “Can an application send Apple events to itself?”

---

**Why do I get error -903 when I try to send an Apple event with AESend()?**

According to *Inside Macintosh*, -903 is a noPortErr. Most likely, this error means that you don't have the isHighLevelEventAware flag set in your application's SIZE resource. Remember, there are two sides to every Apple event. Both the sender and receiver must have their high-level-event-aware flag set.

**Related Topic**

See also Chapter 25, “Resources.”

---

**Can I send an Apple event from a code resource?**

Any type of code resource, including INITs, can send an Apple event. They cannot, however, receive a reply to one. Apple event replies are received through WaitNextEvent(). Since code resources do not have a main event loop, they cannot receive any replies to Apple events.

**Related Topic**

See also Chapter 25, “Resources.”

---

**Does AEPutParamPtr() (and other AEPut...() functions) copy the data passed to it?**

In general, the AEPut...() functions are used to add data to an Apple event. As an example, you might call AEPutParamPtr() to add a buffer of data (along with its associated type) and keyword to an Apple event. The AEPut...() functions
allocate a new block of memory and copy the data passed to them into that block of memory. Once the AEPut...() call returns, you are free to deallocate the buffer you used to pass the data into AEPut...().

Be sure you don't deallocate an Apple event passed into an Apple event handler. However, if you send an event using AESend(), you are welcome to dispose of the event.

**Topic-Related FAQ**

See also FAQ 10-2, “How do I get my application to respond to the Finder for actions such as printing a document or opening a document double-clicked on by the user?”
Macworld Mac Programming FAQs
When the Macintosh was first introduced in 1984, it presented us with a small black-and-white screen. Creating shapes with different patterns and manipulating images at the bit-level was exciting at first. However it served merely to whet our appetites. We were given the inch, but we soon wanted the mile. We wanted color and lots of it—reds, greens, blues, and everything in between. In early 1987, Apple gave it to us when they introduced the Macintosh II and Color QuickDraw. Along with these new color capabilities came a lot of complexity. This chapter tackles many of the important color-related topics involving Color QuickDraw, the Color Manager, and the Palette. It answers FAQs about how color gets displayed on the screen. You'll learn how to create, modify, and then load a palette; how to fetch and change a window's palette; and how to animate a palette (creating some very cool effects). In addition, you'll learn about the data structures used by the Color and Palette Managers (most notably the color table and its associated 'clut' resource), different color models (like HSV and RGB), grayscale, color ramping, and a lot more.

There are three subsections in this chapter:

- Definitions
- Drawing in Color
- Palettes and Color Tables
Definitions

This section answers FAQs about some important terms that are discussed in the following sections. It defines the two most important structures related to color systems as well as color characteristics.

What are the RGB and HSV color models?

The RGB color model defines a color as a combination of the three primary colors: red, green, and blue. The RGB data structure consists of three unsigned short integers, each one representing one of the three colors.

The CMY color model defines a color as a combination of Cyan, Magenta, and Yellow. The relationship between RGB and CMY colors can be simply defined by the following formulas:

- cyan = 1.0 - red
- magenta = 1.0 - green
- yellow = 1.0 - blue

In reality, however, the conversion is a little more complex.

The HSV color model defines a color in terms of hue, saturation, and value as follows:

- The hue component represents a position on the color circle. If viewed as degrees in the color circle, 0 (or 360) degrees represents red, 120 degrees represents green, and 240 degrees represents blue. The hue component is represented in the HSV color structure as a value between 0 and 1.

- The saturation component represents the color intensity. A saturation equal to 0 produces a shade of gray. A saturation equal to 1 produces a pure color.

- The value component represents the brightness of the color. A value equal to 0 produces black. A value equal to 1 means the color is at its brightest.

The three HSV components are three SmallFract types.
A third color model, HSL, is very similar to the HSV color model. Its components are hue, lightness, and saturation. The hue and saturation components are the same as in the HSV model. The lightness component is similar to the HSV model's value component except that a maximum value for lightness means that the color is white. The brightest and most intense colors are produced by a lightness value of exactly half the maximum.

To see how these different color models work, bring up the Color control panel. From one of the pop-up menus, select Other. This brings up the Color Picker dialog box (see Figure 2-1). On the left side of the dialog box are two items that represent the RGB Color Model and the HSL Color Model. Click on both of them and examine the color controls that appear in the dialog box. Play around a bit and see the effects that different combinations the color component have.
What is a color lookup table (CLUT)?

A color lookup table (CLUT) is a data structure that maps indexed values into actual colors. A device's color table contains the complete set of colors that are available at a given time. The value stored for each pixel on the screen of an indexed-pixel device is an index into that device's color table.

What is a color palette?

A palette is a set of colors used by an application to define which colors will be available for screen display. The colors of a palette are loaded into a device's color table as they are needed. Each entry in the palette describes the RGB value of the color, how the color is to be used, and the tolerance value of the color if the color can be approximated. The Palette Manager handles the manipulation of the palette colors into a color table. It determines which colors need to be put into the color table to make the palette's colors available.

When a window comes to the foreground, the Palette Manager determines an optimized color set based upon the window's palette and the palette's of other windows in the background, and arranges the GDevice color table according to that color set.

What is the difference between pmCourteous, pmTolerant, and pmExplicit colors?

The pmCourteous, pmTolerant, and pmExplicit colors are usage flags that tell the Palette Manager how to treat colors, as follows:

- pmCourteous colors accept whatever value the Color Manager determines is the closest match available in the color table.

- pmTolerant colors accept the value, within a specified range, in the color table that the Color Manager determines is the closest match available. If no color in the color table falls within the range, then the
Color Manager loads the color into the table. The Palette Manager tests to see if a color is in the range by calculating the maximum difference of the three RGB components. A tolerance level of 0x0000 indicates the requirement for an exact match. A tolerance level of 0x5000 generally provides sufficient color matching.

- `pmExplicit` colors specify a color index value into a graphic device's color table rather than an RGB color.
- `pmTolerant + pmExplicit` indicates that the desired colors will be located at the specified indexes in all color tables that a window touches.

These flags are assigned to the `ciUsage` field of a `ColorInfo` structure. Each of a palette's color entries contains a `ColorInfo` structure to describe the color.

**What is an inverse table?**

An inverse table is maintained by the Color Manager for each graphics device. It maps a given RGB value to the corresponding index into the graphics device's color table. The Color Manager uses this table to quickly determine the index of the color that best matches a given RGB value.

At startup, the Color Manager builds an inverse table for each graphics device and stores it in the graphic device's `gdITable` field. Each time a color is requested, the Color Manager checks to see if the graphic device's color lookup table has changed. If it has, the inverse table is rebuilt. As a test to see if the table has changed, the `ctSeed` field of the graphics device is compared to the `iTabSeed` field of the graphic device's inverse table. If they are different, then the color lookup table has been modified. Each of the functions that modify a color table increments the value of the table's `ctSeed` field. This way, the inverse table and the color table will always be in sync.

When an application sets the foreground color with `RGBForeColor()` and then draws with that color, Color QuickDraw has to determine how to handle the 48 bits of color information. If the device is a direct device, the bits of each of the RGB components are manipulated to produce either a 24-bit or 16-bit number.

Life is not as simple with an indexed device, however. Indexed pixel maps store indexes into a color table, not the actual RGB data. When an application requests a color, the Color Manager must determine which color in the graphic device's color table (and consequently, in the device's CLUT) most closely matches the requested
The Color Manager’s Color2Index function returns an index to the color that is stored in the pixel map by Color QuickDraw.

To determine the best color match from the color table in the fastest amount of time, the Color Manager uses an inverse table. The ability of the inverse table to accurately match a given color is dependent upon the resolution of the color table, which can have a value of 3, 4, or 5. The resolution describes how many colors are used to represent each of the three RGB components. If the resolution is 3, then each of the red, green, and blue components will be 3 bits long. The size of the table will then be $2^9$ or 512 entries, with each entry representing an index into the table. For devices that support 8 bits of color, or 256 colors, each index is one byte long. Tables with resolutions of 4 and 5 take up approximately 4K and 32K, respectively.

How does the resolution affect the accuracy of matching a color, you ask? Well, remember that the color requested by Color QuickDraw is a full 48 bits — that’s 16 bits per RGB component. The 16 bits of each component must be whittled down to the resolution of the inverse table. This is done by truncating and discarding the lower bytes, using the correct number of high order bytes to match. The remaining bits are used to create an index into the inverse table.

This is most clearly exhibited in a table with a resolution of 4. Suppose you have an RGB color (0x1234, 0x5678, 0x9ABC). To create an index into an inverse table, you take the high 4 bits of each component — 1, 5, and 9. Combined, these bits produce an index of 0x0159 into the inverse table. In a 256-color table, there will clearly be more entries than there are colors. Consequently, there will be a lot of duplicate entries. However, what this buys you is the ability to more closely match a given RGB color with an entry into the table.

**Related FAQs**

- See also FAQ 2-14, “Why do different colors passed to RGBForeColor() appear the same when drawn on the screen?”
- See also FAQ 2-15, “How are 48-bit RGB values displayed when my screen is only 16 bits?”

**Drawing in Color**

This section of the Color chapter contains FAQs that deal with the use of color by the system and in your applications. It includes information on how the system uses special colors to dim items and draw in the user-defined highlight color. It also discusses how colors are displayed on the screen and how to work with shades of gray.
How do I convert from one color model to another?

There are six functions that you use to do conversion among the color models:

- CMY2RGB()
- HSL2RGB()
- HSV2RGB()
- RGB2CMY()
- RGB2HSL()
- RGB2HSV()

Each of these functions does exactly what its name implies (CMY2RGB() converts a CMY color to an RGB color, for example).

Related FAQ

See also FAQ 2-1, "What are the RGB and HSV color models?"

How can I create a grayscale ramp?

A grayscale ramp is a contiguous series of grays, ranging from black to white.

To create a palette of kNumColors colors that represents a grayscale ramp, set the three RGB components of the first palette entry to 0. Then, go through the remaining palette entries and set the component values to (index * (0xFFFF/ kNumColors)), as follows:

```c
//***************
#define kNumColors 255 // number of different grays

void MakeGrayPalette(PaletteHandle *hGrayPalette)
{
    RGBColor rgbColor;
    long index;

    *hGrayPalette = NewPalette( kNumColors , NULL, pmTolerant + pmExplicit , 0x000000 );
```
rgbColor.green = 0;
rgbColor.blue = 0;
for (index = 0; index < kNumColors; index++)
{
    rgbColor.red = (index * 0xffff) / kNumColors ;
    rgbColor.blue = rgbColor.green = rgbColor.red;
    SetEntryColor( *hGrayPalette, index, &rgbColor );
}

Another method of getting a grayscale ramp is to load one of the default grayscale color table 'clut' resources. These are color tables with IDs 34, 36, and 40 for pixel depths of 2, 4, and 8 bits respectively.

On a direct color device, simply specify RGB values where the three RGB component colors are equal.

Related FAQ

See also FAQ 2-8, “How do I reference shades of gray in my program?”

How do I reference shades of gray in my program?

A particular shade of gray is defined as an RGB value where the red, green, and blue components are equal. Setting the RGB components to 0xFFFF produces black. Setting them to 0x0000 produces white. The code below creates a medium shade of gray.

RGBColor rgbColor;
rgbColor.blue = rgbColor.green = rgbColor.red = 0xFFFF/2;

In the HSV color system, different shades of gray can be obtained by setting the saturation field to 0 and alternating the value field. When the saturation field is 0, modifying the hue field has no effect. Setting the value field to 0 produces black. Setting the value field to 1 produces white. The code below produces a medium shade of gray.

HSVColor myHSVColor;
myHSVColor.saturation = 0;
myHSVColor.value = 0.5;
Related FAQ

See also FAQ 2-7, "How can I create a grayscale ramp?"

Why does inverting a color often produce unexpected results on a 256-color system?

On direct devices, inverting a color inverts the actual RGB-component bit values. When the RGB value (0x0000, 0x8000, 0xFFFF) is inverted, it becomes (0xFFFF, 0x7FFF, 0x0000). On an indexed device, Color QuickDraw performs inversion on the pixel indexes, not on the color at the indexes. The results will depend on the contents of the graphics device’s color lookup table.

What RGB value does the system use for dimmed buttons, menus, and window titles?

The gray color used for dimming button, menu, and window titles is not a fixed RGB value. Instead, it is the RGB value that best represents the midpoint between black and white on a particular device. To determine what color this is, you can call the function GetGray(), as follows:

```c
Boolean GetGray( GDHandle gDevice, RGBColor *backGround, RGBColor *foreGround );
```

GetGray() passes back the best available color between the foreGround and backGround colors on the specified gDevice. If an intermediate color is found, then it is passed back in foreGround and GetGray() returns true. If no distinguishable intermediate color is available, then GetGray() returns false. To get the intermediate gray color used for dimming, pass black and white as the two colors.

Related FAQs

See also FAQ 6-16, "How do I get my controls to dim with light gray instead of bitmapped gray?"

See also FAQ 6-28, "How do you deactivate (gray-out) static text items and edit text items in a dialog box?"
How do I get and set the current highlight color?

The current RGB value of the highlight color is stored in the global variable HiliteRGB. This value can be retrieved by the access routine LMGetHiliteRGB() defined in the universal headers. The following call retrieves the highlight color in the currentHilite variable:

```c
RGBColor currentHilite;
LMGetHiliteRGB( &currentHilite);
```

You can set the highlight color by calling HiliteColor() with a pointer to the desired RGB color as a parameter. This will cause all subsequent invert functions with the highlight transfer mode to draw the background in the specified highlight color. However, it does not set the default highlight color for the system that is stored in the global variable HiliteRGB. The calls to LMGetHiliteRGB() will still return the original highlight color. To modify the HiliteRGB value itself, you must call LMSetHiliteRGB().

The following example first sets drawing to use the highlight transfer mode. It then sets the highlight color to red using HiliteColor() and inverts a rectangle to create a red box. It then gets the RGB highlighting value, sets it to the foreground color and draws some text. The text is drawn in the original highlight color. Next, it sets the highlight color to red using LMSetHiliteRGB() and inverts a rectangle to create a red box. It then gets the RGB highlighting value, sets it to the foreground color, and draws some text. This time, the text is drawn red.

```c
//***************
void SetHilite()
{
    Rect iRect1 = {100, 100, 200, 200};
```
Rect iRect2 = {100, 300, 200, 400};
RGBColor currentHilite,
    redHilite = {0xFFFF, 0x0000, 0x0000};
UInt8 hMode;

    // set hilight mode
hMode = LMGetHiliteMode();
BitClr(&hMode, pHiliteBit);

    // draw red rectangle but text in original hilite color
HiliteColor(&redHilite);
LMSetHiliteMode(hMode);
InvertRect(&iRect1);
LMGetHiliteRGB(&currentHilite);
RGBForeColor(&currentHilite);
MoveTo(200, 350);
DrawString("\pDraws in original color");

    // draw red rectangle and red text
LMSetHiliteRGB(&redHilite);
LMSetHiliteMode(hMode);
InvertRect(&iRect2);
LMGetHiliteRGB(&currentHilite);
RGBForeColor(&currentHilite);
MoveTo(400, 350);
DrawString("\pDraws in Red");
}

Related FAQs

- See also FAQ 2-12, "How can I get drawing commands to draw in the highlight color?"

- See also FAQ 23-6, "What is the difference between highlighting and inverting?"

- See also FAQ 23-7, "How do I highlight an object?"

How can I get drawing commands to draw in the highlight color?

To draw with the current highlight color, you first retrieve the RGB value of the highlight color by calling LMGetHiliteRGB(). Then set the highlight color to be the foreground color by calling RGBForeColor(). Drawing functions will then
draw in the RGB value of the highlight color.

Related FAQs

- See also FAQ 2-11, “How do I get and set the current highlight color?”
- See also FAQ 23-6, “What is the difference between highlighting and inverting?”
- See also FAQ 23-7, “How do I highlight an object?”

What are the RGB values of the eight colors used in Basic QuickDraw?

Basic QuickDraw defines eight colors. The RGB values are defined in Table 2-1.

<table>
<thead>
<tr>
<th>Value</th>
<th>Color</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
<td>$0000</td>
<td>$0000</td>
<td>$0000</td>
</tr>
<tr>
<td>1</td>
<td>Yellow</td>
<td>$FC00</td>
<td>$F37D</td>
<td>$052F</td>
</tr>
<tr>
<td>2</td>
<td>Magenta</td>
<td>$F2D7</td>
<td>$0856</td>
<td>$84EC</td>
</tr>
<tr>
<td>3</td>
<td>Red</td>
<td>$DD6B</td>
<td>$08C2</td>
<td>$06A2</td>
</tr>
<tr>
<td>4</td>
<td>Cyan</td>
<td>$0241</td>
<td>$AB54</td>
<td>$EAFF</td>
</tr>
<tr>
<td>5</td>
<td>Green</td>
<td>$0000</td>
<td>$64AF</td>
<td>$11B0</td>
</tr>
<tr>
<td>6</td>
<td>Blue</td>
<td>$0000</td>
<td>$0000</td>
<td>$D400</td>
</tr>
<tr>
<td>7</td>
<td>White</td>
<td>$FFFF</td>
<td>$FFFF</td>
<td>$FFFF</td>
</tr>
</tbody>
</table>

Related Topic

- See also Chapter 23, “QuickDraw: Drawing.”
**Why do different colors passed to RGBForeColor() appear the same when drawn on the screen?**

When RGBForeColor() is called on an indexed device, the color in the current device's color table that most closely resembles the color passed to RGBForeColor() is used. This means that two different colors may map to the same color in the color table. If a color is specified that resembles very few colors in the table (such as a green color trying to find a match in a color table that contains all shades of red), a large range of colors may get mapped to the same entry in the color table.

On direct devices, this problem will not occur since the colors passed to RGBForeColor() get translated to their 16-bit or 32-bit equivalents.

**Related FAQs**

- See also FAQ 2-5, “What is an inverse table?”
- See also FAQ 2-15, “How are 48-bit RGB values displayed when my screen is only 16 bits?”

**How are 48-bit RGB values displayed when my screen is only 16 bits?**

The system converts a 48-bit RGB value to a 16-bit pixel by storing the most significant 5 bits of each 16-bit color component. The high bit of the pixel is unused.

**Related FAQs**

- See also FAQ 2-5, “What is an inverse table?”
- See also FAQ 2-14, “Why do different colors passed to RGBForeColor() appear the same when drawn on the screen?”

**Palettes and Color Tables**

This section contains FAQs that deal with palettes and color lookup tables. It describes how colors are displayed on the screen and shows how to customize the
color environment to display colors other than those defined in the system default color tables. This section also provides tips and techniques on how to use and manipulate colors in your application.

**Where can I find documentation on the Palette Manager and Color Manager?**

The Palette Manager and Color Manager are not documented in the standard *New Inside Macintosh* volumes, nor are they part of the standard Toolbox Assistant. They are covered in the book *Advanced Color Imaging on the Mac OS*, published by Addison-Wesley, and in the online version called *Advanced Color Imaging Reference*. There is also a separate module for Toolbox assistance called “Advanced Color Ref.qv” that covers these topics.

**How do I create a palette?**

There are two ways to get a palette in your application. The first is to define a ‘pltt’ resource with a resource editor and load it into your application with `GetNewPalette()`. The second is to create the palette from scratch, as follows:

```
NewPalette( numEntries, srcColors, srcUsage, srcTolerance )
```

If you pass a color table into the `srcColors` parameter, the palette will contain the colors in the color table. If you pass nil into the `srcColors` parameter, all of the colors will have RGB values of 0,0,0.

The following sample creates a palette of 128 different shades of red:

```//**************************
PaletteHandle MakeRedPalette()
{
RGBColor c;
long i;
PaletteHandle redPalette;
redPalette = NewPalette( 128, nil, pmTolerant, 0x0000 );
c.green = 0;
c.blue = 0;
for ( i=0; i<128; i++ )
{
```
CHAPTER 2: Color

How do I change the color palette?

You can change the color values of a palette entry by calling the following:

\[
\text{SetEntryPalette( thePalette, paletteEntry, newRGB );}
\]

The usage and tolerance values of a palette entry can be changed by calling the following:

\[
\text{SetEntryUsage( thePalette, paletteEntry, newUsage, newTolerance );}
\]

The colors and settings are not visibly changed until \text{ActivatePalette()} is called.

How can I convert an RGB color into an index to a palette?

There's no single call that will return a palette index that matches an RGB color. However, there are some methods that will enable you to find the matching palette index.

If you have a palette that has been activated and has the usage flags of pmTolerant + pmExplicit with a tolerance of 0, then you can call \text{Color2Index()}, which will return the index of the best match in the color lookup table. This index will correspond to the index in your palette. The palette should be the same size as the color table; otherwise, a preexisting color in the palette may be returned.

If you have an activated palette that is set to a tolerance of 0, but the usage does not specify pmExplicit, you can get the best matching color and then search your palette yourself. Call \text{Color2Index()} to get the best match to your RGB color in the color table. Then call \text{Index2Color()} to get the RGB value at this index in the color table. Now you have the proper RGB value to search for in the palette.

If you do not want to have the screen modified by loading your palette's colors into the color lookup table, you can perform the conversion functions in a GWorld.
First, convert your palette to a color table by calling PaletteToCTab(). Then insert the color table into the GWorld with UpdateGWorld(). Now you can use the methods previously described.

How do I get the screen to use the colors in my palette?

Whenever a window is activated or moved, the color table of the devices on which the window exists is automatically updated to reflect the colors in the window’s palette. This is done with the ActivatePalette() command. You can manually load the palette of a window by calling ActivatePalette(). When you change a window’s palette by calling a function such as SetEntryColor(), you must call ActivatePalette() if you want the changes to take effect immediately.

If you have a palette to be loaded that is not associated with a window, first call SetPalette() to set the palette as the window’s palette and then call ActivatePalette().

ActivatePalette() does not affect off-screen graphics worlds.

How can I draw with a color at a specific location in my palette?

PMForeColor() takes an index into the current graphics port’s palette and sets the appropriate RGB color as the foreground color. In order to get an accurate representation of the palette, the usage flags must be set to pmTolerant with a tolerance level of 0. This means that the exact palette colors will be retrieved.

The following example draws an array of squares containing the color entries of the palette belonging to the specified palette. It gets the number of palette entries from the pmEntries field of the palette structure. To make this example work with another palette, call SetPalette() to assign the palette to a window and then call this function.

```c
//****************
#define kNumCol 16
#define kBoxSize 16

void DisplayPaletteColors( WindowPtr cWindow )
```
How do I load a color table into a palette and a palette into a color table?

To load a color table into an existing palette, call CTab2Palette(). CTab2Palette() resizes the palette to match the color table. NewPalette() loads the colors from an existing color table into the newly created palette.

To load a palette into a color table, call Palette2CTab(). Palette2CTab() resizes the color table to match the palette.

How can I get the default application palette?

The default palette for an application can be retrieved by calling GetPalette() with a parameter of -1, as follows:

```c
PaletteHandle hDefaultAppPalette;

hDefaultAppPalette = GetPalette( (WindowPt) -1 );
```
How can I get the palette of a window?

GetPalette() returns a handle to the palette of the specified window. If the window is not a color window or has no palette, then GetPalette() returns nil.

Related FAQ

See also FAQ 2-23, “How can I get the default application palette?”

Related Topic

See also Chapter 33, “Windows.”

How do I assign a color palette to a window?

There are a number of ways to assign a palette to a window. The easiest is to create a palette ‘pltt’ resource with the same ID as the window. The Palette Manager will automatically attach the palette to the window when the window opens.

You can attach a palette to a window programmatically by calling SetPalette() or NSetPalette(). The palette can be a ‘pltt’ resource that does not have the same ID as the window that was loaded with GetNewPalette() or a palette created in your code with NewPalette(). NSetPalette() gives you more control over how the window responds to changes in the color environment.

You can define a default palette for your application by creating a ‘pltt’ resource and assigning it an ID of 0. When a color window is opened, the Palette Manager looks for a ‘pltt’ resource with the same ID. If none is found, and a palette has not been assigned to the window by SetPalette() or NSetPalette(), the Palette Manager looks for the default application palette (a palette with the resource ID of 0). If there is none, then the system default palette (the palette with ID 0 in the System file) is used.
You can also define a default application palette programmatically by creating a palette or retrieving a 'pltt' resource and then calling SetPalette() or NSetPalette() with the first parameter, a WindowPtr, set to -1 as in the following example:

```
SetPalette((WindowPtr)-1, hDefaultAppPalette, TRUE);
```

**Related Topic**

See also Chapter 33, "Windows."

---

**How do I set up a default palette for all windows in my application?**

You can define a default palette for your application by creating a 'pltt' resource and assigning it an ID of 0. When your application starts up, the Palette Manager loads the application's default palette and stores it in the low-memory global variable called AppPalette. Whenever a new color window is opened by your application, the Palette Manager automatically searches for a palette resource that it can use to create a palette for the window. If a 'pltt' resource with the same resource ID as the window cannot be found, then the application creates a palette from the default palette (the palette with resource ID of 0). If a default application palette does not exist, then the default system palette resource (the 'pltt' resource in the System file) is used.

You can also define a default application palette programmatically by calling SetPalette() or NSetPalette() with the first parameter, a WindowPtr, set to -1 as in the following example:

```
SetPalette((WindowPtr)-1, hDefaultAppPalette, TRUE);
```

The palette can be a 'pltt' resource that was loaded with GetNewPalette() or a palette created in your code with NewPalette().

**Related FAQ**

See also FAQ 2-25, "How do I assign a color palette to a window?"

**Related Topic**

See also Chapter 33, "Windows."
**Does a palette that is attached to a window get disposed of when the window is closed?**

No. If you attach a palette to a window with `SetPalette()` or `NSetPalette()`, then the palette is owned by your application and must be disposed of explicitly. You dispose of the palette with `DisposePalette()`.

If a palette is automatically assigned to a window by the Window Manager and Palette Manager because the palette and the window have the same ID, then the palette is automatically disposed of when the window is disposed of. The same is true if the window was created with the default application resource.

**Related Topic**

See also Chapter 33, “Windows.”

---

**Why is my window not immediately affected when I change my palette?**

Whenever you modify a window’s palette, you have to call `ActivatePalette()` afterwards for the changes to take effect. This allows you to make a series of calls modifying the window’s palette and then force a single update.

`ActivatePalette()` is automatically called when a window is opened, activated, or moved.

**Related Topic**

See also Chapter 33, “Windows.”

---

**How can I get a picture that does not use the default palette to display properly in a window?**

To properly display a picture that does not use the default palette in a window, create or retrieve a palette that contains the colors used by the picture. You can get this color information of a picture with `GetPictInfo()`. Then assign the palette to the window with `SetPalette()` or `NSetPalette()`. Each time the window
becomes active, its palette will be installed into the color tables of all graphics devices that the window touches. Graphics devices that are not touched by the window will not have their color tables affected.

Related FAQ

See also FAQ 24-21, "How do I get a palette or color table of a picture?"

Related Topics

See also Chapter 24, “QuickDraw: Pictures.”

See also Chapter 33, “Windows.”

**How can I get the current color table of my screen?**

To get the color table of a screen, you must first get its graphics device. The color table is stored in the pmTable field of the graphic device’s pixel map, gdPMap.

The following code gets the color table of the graphics device of the main screen:

```c
//**************
CTabHandle GetMainColorTable()
{
    GDHandle    gDevice;
    PixMapHandle hPixMap;
    CTabHandle  hColorTable;

    gDevice = GetMainDevice();

    HLock((Handle)gDevice);
    hPixMap    = (*gDevice)->gdPMap;
    HLock((Handle)hPixMap);
    hColorTable = (*hPixMap)->pmTable;
    HUnlock((Handle)hPixMap);
    HUnlock((Handle)gDevice);

    return(hColorTable);
}
```
How can I get the RGB value of a specific index into the color table, and how can I get the index of a particular RGB color in the color table?

The routine `Index2Color()` returns the RGB color that is stored at a specific location in the color table of the current graphics device.

The routine `Color2Index()` returns the index into the color table of the color that best matches the RGB color.

The following example displays all of the colors currently loaded in the graphics device, `theDevice`, on the window, `cWindow`. Since the current window’s palette is loaded into the color table when the window is made active, this displays the color table with the window’s palette loaded into it. To get the state of the color table with a minimal effect on the window’s palette, set the window’s palette to be a small palette, such as a palette with one black entry and one white entry.

```c
//**********************
void DoShowColorTable( GDHandle theDevice )
{
    PixMapHandle screenPMapH;
    int pixelDepth;

    screenPMapH = (*theDevice)->gdPMap;
    pixelDepth = (*screenPMapH)->pixelSize;

    switch( pixelDepth )
    {
        case 1:
            DisplayColors( 1, 2 );
            break;
        case 2:
            DisplayColors( 2, 2 );
            break;
        case 4:
            DisplayColors( 4, 4 );
            break;
        case 8:
            DisplayColors( 16, 16 );
            break;
        default:
            break;
    }
}
```
How do I restore the color environment when my application goes to the background?

To restore the color environment when your application goes to the background, call the following series of routines:

```
RestoreDeviceClut(nil);
PaintBehind(FrontWindow(), GetGrayRgn());
PaintOne(nil, GetGrayRgn());
DrawMenuBar();
```
RestoreDeviceClut() sets the color table of a graphics device to its default state. PaintBehind() and PaintOne() update visible windows on the screen. And finally, to set the apple in the menu bar to the right color, DrawMenuBar() is called to redraw the menu bar.

What are the default color table IDs?

The default color lookup table IDs for color systems are listed in Table 2-2.

<table>
<thead>
<tr>
<th>Pixel Depth</th>
<th>Color Table ID</th>
<th>Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>black, white</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>black, 50% gray, highlight color, white</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>black, 14 colors, white</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>black, 254 colors, white</td>
</tr>
</tbody>
</table>

The default color lookup tables in gray mode represent a grayscale ramp with gray values evenly spaced between white and black. The 'clut' resource IDs are listed in Table 2-3.

<table>
<thead>
<tr>
<th>Pixel Depth</th>
<th>Color Table ID</th>
<th>Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>black, white</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>black, 1/3 gray, 2/3 gray, white</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
<td>black, 14 shades of gray, white</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>black, 254 shades of gray, white</td>
</tr>
</tbody>
</table>

The IDs of the default color tables for color systems with the highlight color added are listed in Table 2-4.
Table 2-4:
IDs for Default Color Tables with Highlight Color Added

<table>
<thead>
<tr>
<th>Pixel Depth</th>
<th>Color Table ID</th>
<th>Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>66</td>
<td>black, 50% gray, highlight color, white</td>
</tr>
<tr>
<td>4</td>
<td>68</td>
<td>black, 14 colors including highlight color, white</td>
</tr>
<tr>
<td>8</td>
<td>72</td>
<td>black, 254 colors including highlight color, white</td>
</tr>
</tbody>
</table>

Related FAQ

See also FAQ 2-23, “How can I get the default application palette?”

How can I monitor the colors in the color table of an indexed device?

You can monitor the colors in a color table of an indexed device by using explicit colors. First, create a grid of colors of a size that matches the number of colors displayed by your screen. Then create a palette of explicit colors of the same size. Use PMForeColor() to set the foreground color and PaintRect() to fill each square with the corresponding color in the color table. For example, to fill the first square you would call the following:

```
PMForeColor(0);
PaintRect( &destRect );
```

This would fill the grid with the color in the first entry in the device’s color table.

How do I use animated colors?

To use animated colors, you must first create an animated color entry in a palette. This is done by setting the usage category of a palette entry to pmAnimated either when the palette is created or with a SetEntryUsage() call. After assigning the palette to a window, draw the animated colors with PmForeColor() or PmBackColor(). To create the color table animation, call AnimateEntry() or AnimatePalette() to change the color in the video device’s color table. AnimateEntry() changes a single palette color to a specific RGB color.
AnimatePalette() changes a contiguous set of palette entries by transferring values from a color table.

The following example displays the standard 8-bit color table, creates a palette of animated entries, and loads the color table into the palette. First, the palette is assigned to the window and an array containing all of the colors in the palette is displayed. It then cycles all of the colors through the array by moving the color in each entry to the previous position with AnimatePalette(), which copies the entries from the temporary color table to the palette. The first color is moved to the last position in the array with AnimateEntry() to allow for continuous cycling of colors. The new palette entries are then stored in the temporary color table structure to allow the next iteration through the loop to copy new colors to the palette.

```c
//***************
void DoAnimate()
{
    CTabHandle hColorTable, StoreCTab;
    PaletteHandle newPalette;
    RGBColor changeColor;
    long finalTicks;
    int i;

    hColorTable = GetCTable(8);
    StoreCTab = GetCTable(8);
    newPalette = NewPalette(256, hColorTable, pmAnimated, Ox00);

    SetPalette(pWindow, newPalette, TRUE);
    DisplayPaletteColors(pWindow);

    for (i = 1; i <= 256; i++)
    {
        GetEntryColor(newPalette, 0, &changeColor);
        AnimatePalette(pWindow, StoreCTab, 1, 0, 255);
        AnimateEntry(pWindow, 255, &changeColor);

        Palette2CTab(newPalette, StoreCTab);
        Delay(3, &finalTicks);
    }
}
#define kNumCol 16
#define kBoxSize 16

void DisplayPaletteColors(WindowPtr cWindow)
{
    Rect r;
    int index, row, col;
```
short numPalEntries;
PaletteHandle hPalette;

hPalette = GetPalette( cWindow );
numPalEntries = (*hPalette)->pmEntries;

for ( index = 0; index < numPalEntries; index++ )
{
    row = index / kNumCol;
col = index % kNumCol;

    r.top = row * kBoxSize;
    r.left = col * kBoxSize;
    r.bottom = r.top + kBoxSize;
    r.right = r.left + kBoxSize;

    PmForeColor( index );
    PaintRect( &r );
}

Why can't I get animated colors to work when I'm in 16-bit color mode?

Color animation requires color tables. Color tables are used by indexed devices
only. Since screens with 16- and 32-bit pixel depth do not have color tables, they
cannot support animated colors.

What effect does the resolution of graphics device's inverse table have?

See FAQ 2-5, "What is an inverse table?"

Topic-Related FAQs

See also FAQ 6-9, "Why can I only draw in the eight basic colors in a dialog box?"

See also FAQ 6-15, "How do I make the buttons the same color as my color dialog
boxes?"
See also FAQ 6-30, "How can I set the fonts and colors of dialog box items?"

See also FAQ 12-9, "How can I fade my display to black and back again, and can I fade to another color?"

See also FAQ 23-6, "What is the difference between highlighting and inverting?"

See also FAQ 23-10, "How do I draw a shape filled with a solid color?"

See also FAQ 23-20, "How do I get the most frequently used colors in a pixmap?"

See also FAQ 23-22, "Why do the colors in my image copied with CopyBits() get distorted?"

See also FAQ 23-32, "How can I use my own colors in a GWorld?"

See also FAQ 24-21, "How do I get a palette or color table of a picture?"

See also FAQ 33-16, "How can I set up custom colors in my window?"

See also FAQ 33-18, "Why can I only draw in the eight basic colors in my window?"
Cursors

The cursor acts as one of vehicles through which we can communicate to our computers, telling it what we want done and where. In turn, the cursor can also present the user with subtle bits of information such as “Hey, I’m busy right now” or “You are over a text field, so type.” This chapter covers a wide range of these cursor-related issues. You’ll learn how to animate a cursor (like a spinning beach ball, moving watch hand, or counting fingers), how to move the cursor to a specific location on the screen, how to hide and show the cursor, how to change the cursor depending on the region of a window it is over, and more.

There are five subsections in this chapter:

- Definitions
- Setting the Cursor
- Hiding and Showing the Cursor
- Cursor Location
- Spinning the Cursor
Definitions

This section defines animated cursors. (Animated cursors are just one of the many concepts that will be discussed in this chapter.)

What is an animated cursor?

An animated cursor is a collection of black-and-white cursor ('CURS') resources, grouped together in a single resource, an 'acur' resource. The 'acur' resource can be passed to SpinCursor() and RotateCursor() to display a spinning cursor. These two routines continuously cycle through the individual cursors in the 'acur' resource and display them.

Setting the Cursor

This section contains FAQs that deal with changing the shape of the cursor. It presents the standard cursors and how to access them as well as some strategies for using cursors properly.

How do I change the shape of the cursor?

There are two calls that are mainly used to change the shape of the cursor, SetCursor() and SetCCursor(). SetCursor() is used to set the cursor to a black-and-white cursor, a 'CURS' resource. It takes a pointer to a cursor structure as a parameter. SetCCursor() is used to set the cursor to a color cursor, a 'crsr' resource. It takes a handle to a color cursor as a parameter.

GetCursor() can be used to get a handle to a specified 'CURS' resource that can be displayed with SetCursor(). There are four standard 'CURS' resources provided by the system. They are the text I-beam cursor, the crosshair cursor, the plus cursor, and the watch cursor.

The resource ID constants for the standard 'CURS' resources are as follows:

- iBeamCursor = 1
- crossCursor = 2
- plusCursor = 3
- watchCursor = 4
The arrow cursor is stored in the arrow field of the QuickDraw global variable qd. To set the cursor to the arrow cursor, call SetCursor(qd.arrow). InitCursor() also resets the shape of the cursor to the standard arrow cursor.

The following code loads and sets the crosshair cursor and then resets the cursor to the arrow cursor:

```c
void ChangeCursor()
{
    Cursor theCrossCrsr;
    CursHandle hCurs;

    hCurs = GetCursor( crossCursor );
    HLock( (Handle) hCurs);
    theCrossCrsr = **hCurs;
    HUnlock( (Handle) hCurs);

    SetCursor( &theCrossCrsr );

    // do stuff
    SetCursor( &qd.arrow );
}

GetCCursor() is used to load a color cursor (a 'crsr' resource) into memory. The color cursor can then be set as the cursor with SetCCursor(). Unlike GetCursor(), GetCCursor() creates a new CCRsr record each time it is called. Therefore, it should not be called before each call to SetCCursor() unless the color cursor is disposed of after each use. The following code, which loads and displays a color cursor, shows an example of this:

```c
void ChangeCCursor()
{
    CCRsrHandle hCCurs;

    hCCurs = GetCCursor(128);
    SetCCursor( hCCurs);

    // do stuff
    SetCursor( &qd.arrow );
```
DisposeCCursor( hCCurs);

The shape of the cursor can also be changed by the routines that handle spinning cursors, SpinCursor() and RotateCursor().

Related FAQ

See also FAQ 3-12, "How do I put a spinning cursor in my program?"

**How do I set the cursor back to the arrow cursor after I set it to one of the other standard cursors?**

To set the cursor back to the arrow cursor, call InitCursor() or call SetCursor() and pass the arrow field of the qd QuickDraw global variable to it.

The other standard cursors (the text I-beam cursor, the crosshair cursor, the plus cursor, and the watch cursor) are resources accessed through constants by the GetCursor() call. The arrow cursor is stored in the arrow field of the QuickDraw global variable qd. The following call sets the cursor to the standard arrow cursor:

SetCursor( &qd.arrow );

InitCursor() sets the current cursor to the standard arrow cursor and makes it visible.

**How do I get my program to change the cursor to another cursor when over a certain area in my window?**

To change the cursor when it moves over a certain area in the window, you must first determine if the cursor is in the target area and then change the cursor with SetCursor() or SetCCursor().

There are two methods to test if the cursor is over the target area. The first is to use PtInRect(), PtInRgn(), or some similar type of cursor location checking routine. You should call this routine in your eventloop. Then call SetCursor() or SetCCursor() to set the cursor to the desired shape.
The other method is to test for the mouse-moving event generated by \texttt{WaitNextEvent()}. When the mouse moves out of the region passed to \texttt{WaitNextEvent()}, a mouse-moving message is generated. To detect when the mouse moves into a specific area, create a region that contains all points in the current region. Then use \texttt{DiffRegion()} to remove, from that region, all of the areas that you want to detect. Pass the resulting region to \texttt{WaitNextEvent()}. Each time the cursor moves out of the main region into one of your specified regions, a mouse-moving message will be generated. To detect when your mouse leaves this area, pass the region in which the cursor is positioned to \texttt{WaitNextEvent()}. This will give you a mouse-moving message when the cursor moves out of the region.

In the following code, a large rectangular region filled almost completely with the graphics port is created and a region containing the frame of the picture is created. The picture region is then subtracted from the larger region.

\begin{verbatim}
SetRectRgn(arrowRgn, -32000, -32000, 32000, 32000);
RectRgn( pictRgn, &CmyPicture->picFrame);
DiffRgn(arrowRgn, pictRgn, arrowRgn);
\end{verbatim}

Assuming the mouse is in the \texttt{arrowRgn}, a mouse-moving event will be generated when the cursor moves into the picture, and consequently, out of \texttt{arrowRgn}. You can then change the cursor. When the cursor is over the picture, pass the picture region to \texttt{WaitNextEvent()}. When the cursor moves out of the picture region, another mouse-moving event will be generated. Then change the cursor back to its original state and pass \texttt{arrowRgn} to \texttt{WaitNextEvent()} again.

\section*{Related FAQs}

\begin{itemize}
  \item See also FAQ 3-9, "Will a hidden cursor report mouse-down and mouse-moving events?"
  \item See also FAQ 7-17, "How can I tell when the mouse moves if there are no mouse-moving events?"
\end{itemize}
When my application returns to the foreground, why doesn’t the cursor return to the cursor shape it was in immediately before it was sent to the background?

It is up to your application to set the cursor to the type you want when it returns to the foreground. Whenever you receive a resume event, you can call SetCursor() or SetCCursor() to reset the cursor. This will undo whatever changes other applications have made. Since these calls are very “inexpensive,” you can even set the cursor each time through your event loop.

Why does my program work fine with a black-and-white cursor but crashes with a color cursor?

Unlike GetCursor(), which only loads a resource, GetCCursor() creates a new color cursor, CCursor, record each time it is called. GetCCursor() should not be called before each call to SetCCursor() unless the color cursor it had previously created is disposed of after each use. The solution is to load the ‘crsr’ resource once at the beginning of your program and use the handle in subsequent SetCCursor() calls.

Related FAQ

See also FAQ 3-14, “Why does my VBL task crash when I try to spin a cursor?”

Hiding and Showing the Cursor

This section contains FAQs that deal with hiding the cursor. It shows how you can hide the cursor and talks about some ramifications that may have.

How do I hide the cursor?

There are a few different calls that can be used to hide the cursor. The most obvious is the call HideCursor(). To redisplay the cursor, call ShowCursor(). Each call to HideCursor() must be balanced by a corresponding ShowCursor() to redisplay the cursor.

Calling InitCursor() will also redisplay a hidden cursor, but it will reset it to
the default arrow cursor. `ShieldCursor()` will hide the cursor while it is moving or while the cursor is in a specified rectangle. `ShowCursor()` gets the cursor out of the mode set by `ShieldCursor()`.

`ObscureCursor()` hides the cursor until it is moved. It can be made visible by calling `ShowCursor()` or by moving the cursor. Once the cursor is moved and is redisplayed, `ShowCursor()` does not have to be called again.

**Why won't `ShowCursor()` make the cursor visible again after I call `HideCursor()` more than once?**

The cursor is only visible when the cursor level is set to 0. Each call to `HideCursor()` decrements the cursor level. Each call to `ShowCursor()` increments the cursor level. Only when each `HideCursor()` is balanced by a call to `ShowCursor()` will the cursor level be returned to 0 and the cursor made visible. `InitCursor()` will reset the cursor level to 0 as well as resetting the cursor to the arrow cursor.

**Will a hidden cursor report mouse-down and mouse-moved events?**

Yes, cursors hidden by `HideCursor()` still cause mouse-down events and mouse-moved messages to be posted.

**Related FAQs**

- See also FAQ 3-4, “How do I get my program to change the cursor to another cursor when over a certain area in my window?”
- See also FAQ 7-17, “How can I tell when the mouse moves if there are no mouse-moved events?”

**Related Topic**

- See also Chapter 7, “Events.”
Cursor Location

This section contains FAQs that deal with the movement of the cursor. In order for your program to respond to cursor actions taken by the user, it must know where the cursor is positioned on the screen. Also presented is a FAQ that tells you how to move the cursor programmatically — something that is a little tricky and not common knowledge.

How can I get the location of the cursor?

At any time, the current location of the cursor can be retrieved by the function GetMouse(). The location returned is in local coordinates of the current graphics port.

The mouse location can also be determined by examining the event structure passed back by WaitNextEvent(). The where field of the structure contains the location of the cursor at the time the event was posted. This location is in global coordinates.

How do I get my program to move the cursor to a specific location on the screen?

When deciding how to move the mouse programmatically, you should first consider why you want to do this. If you are writing a demo-like program and only need to display the cursor, as opposed to requiring the processing of actual mouse button events and mouse movements, you can hide the actual cursor and move an image of it. This could be done using CopyBits() to alternately send a bitmap of the cursor image and the appropriate section of the window’s background to the screen to produce the effect of smooth movement of the cursor.

Moving the actual cursor gets a bit more complicated. If you have an older machine, you will have to rely on the following three low memory globals:

- RawMouse (0x082C): // the current mouse location
- MTemp (0x0828)
- CrsrNew (0x08CE): // set after you change MTemp and RawMouse
You can get and set the mouse position by reading the point stored at RawMouse. To change the cursor position, set both RawMouse and MTemp to the new position and set CrsrNew to 1 to indicate that it has changed.

If you have a newer machine, you can utilize the Cursor Device Manager. You can traverse through the list of cursor devices by calling CrsrDevNextDevice(). The first member of the list, and usually the only member, is retrieved by passing nil as the argument. The current cursor location is retrieved by accessing the whichCursor where field. The cursor is moved by calling CrsrDevMoveTo().

You can test if the Cursor Device Manager is available by calling NGetTrapAddress() to check for the existence of the Cursor Device Manager trap, 0xAADB.

The following sample code moves the cursor diagonally down the screen until the user presses the mouse button. If the Cursor Device Manager is available, the sample uses its calls. If it is not available, then the sample manually manipulates the low memory globals to move the mouse.

```
#include <CursorDevices.h>

// cursor device manager trap
#define _CrsrDevDispatch OxAADB

// low memory globals for older method
#define MTemp 0x828
#define RawMouse 0x82c
#define CrsrNew 0x08CE

void NewCrsrDeviceMouseMove(void);
void OldMouseMove (void);

void DoMoveCursor()
{
    if (NGetTrapAddress(_CrsrDevDispatch, ToolTrap) !=
        NGetTrapAddress(_Unimplemented, ToolTrap))
        NewCrsrDeviceMouseMove();
    else
        OldMouseMove();
}
```
/**********
void NewCrsrDeviceMouseMove()
{
    CrsrDevicePtr myMouse;
    Point newPoint;
    long delayTicks;

    // Get first device with one button.
    // This will probably be the standard mouse
    // start at head of cursor dev list
    myMouse = nil;
    do
    {
        // get the next cursor device
        CrsrDevNextDevice(&myMouse);
    }
    while ( (myMouse != nil) && (myMouse->cntButtons != 1) );

    newPoint = myMouse->whichCursor->where;
    do
    {
        newPoint.h += 1;
        newPoint.v += 1;

        // Move cursor to new position
        CrsrDevMoveTo(myMouse, (long)newPoint.h,
                      (long)newPoint.v);
        Delay(10,&delayTicks);
    }
    while (!Button());
}

/**********
void OldMouseMove ()
{
    Point where;
    long delayTicks;
    int i;

    // Get current cursor position
where = *(Point *)RawMouse;

do
{
    where.v += 1;
    where.h += 1;

    // Set new cursor position
    *(Point *)RawMouse = where;
    *(Point *)MTemp = where;
    *(Ptr)CrsrNew = 0xFFFF;

    Delay(10,&delayTicks);
}
while (!Button());

Spinning Cursors

Whenever a program performs some processing that will take a fair amount of time, it should present a spinning cursor to let the user know that it is working. This section contains FAQs that deal with the various methods that can be used to display spinning, or animated, cursors. Using VBL tasks to change the cursor at interrupt time is also covered.

How do I put a spinning cursor in my program?

A spinning, or animated, cursor is used to signal to the user that your application is performing an action that takes some time. The appearance of a spinning cursor is produced by continuously displaying a series of cursor images. Each image is slightly different than the one before, as shown in Figure 3-1.
There are two basic methods of spinning a cursor: you can manually display some 'CURS' resources or have the 'CURS' resources that are grouped in a single 'acur' displayed. The calls that are used to perform these functions are as follows:

- **SetCursor()**: Sets the specified black-and-white cursor as the current 'CURS' resource.
- **SetCCursor()**: Sets the specified color cursor as the current 'crsr' resource.
- **InitCursorCtrl()**: Loads the resources necessary for displaying an animated cursor.
- **RotateCursor(), SpinCursor()**: Display an animated cursor ('acur') resource.

To display a series of cursors with **SetCursor()**, you can create an array of cursor handles. Then every time you want to set the cursor to the next image, you just set it to the next handle in the next array position. The following code circulates through an array, hCurArray, of 8 cursor handles. The cursor is changed every 1/3 of a second (20 ticks).
void mySpinCursor()
{
    CursHandle hCursArray[8];
    long finalTicks;
    int i;

    for ( i = 0; i < 8; i++)
        hCursArray[i] = GetCursor(128 + i);

    for ( i = 0; i < 24; i++)
    {
        SetCursor( *(hCursArray[i % 8]) );
        Delay(20, &finalTicks);
    }
}

To display an animated cursor, you must first get the 'acur' resource data. Next, InitCursorCtl() needs to be called to load the necessary resources. InitCursorCtl() takes a handle to an animated cursor. If nil is passed to InitCursorCtl(), the 'acur' resource with the resource ID of 0 is loaded, along with the corresponding 'CURS' resources. To perform the actual animation, SpinCursor() or RotateCursor() is called. The rate at which either of these two functions is called determines the speed of the cursor animation. The following code shows how to spin a cursor with SpinCursor():

void RotateSpinCursor()
{
    Handle myAcur;
    long finalTicks;
    int i;

    myAcur = GetResource('acur',128);

    InitCursorCtl((acurHandle)myAcur);
    for ( i = 0; i < 20; i++)
    {
        SpinCursor(0);
        Delay(20, &finalTicks);
    }

    You can spin the cursor in your program's code or create a VBL task to spin it. To rotate the cursor from your program, periodically call one of the previously-
described functions while you are performing your task. So as not to call the function too frequently, change the cursor only after a certain period of time has expired since the last cursor change. You can test this by calling `TickCount()` to get the current tick count and compare it with the value generated the last time that the cursor was modified. If enough time has elapsed since the last changing of the cursor, call one of the appropriate functions above to change to the next cursor. When using this method to change the cursor in your program, the smoothness of the cursor rotation is dependent upon the processing time. Too much time dedicated to processing may result in a “jerky” cursor rotation.

Rotating the cursor from a VBL task will result in a relatively constant and smooth motion. To use a VBL task, your program must get the handles of the ‘CURS’ resources that will be used, lock them in memory, and install the VBL task. The VBL task must be set up to access the application’s globals and call `SetCursor()`, cycling through the different cursors with each call. The timing of the cursor setting is controlled by the frequency with which the VBL task is called.

**Related FAQs**

- See also FAQ 3-2, “How do I change the shape of the cursor?”
- See also FAQ 3-13, “How can I spin the cursor using an interrupt routine?”
- See also FAQ 3-14, “Why does my VBL task crash when I try to spin a cursor?”
- See also FAQ 21-1, “When do I use VBL tasks, and when do I use the Time Manager?”

**How can I spin the cursor using an interrupt routine?**

To have your cursor spin independent of your application’s processing, you can use a VBL task. First, you should have your application load all of the cursors that you are going to cycle through and store their handles as globals or in the structure passed to the VBL task. `GetCursor()` moves memory so it cannot be called from an interrupt routine. The main routine should lock the handles.

Create and install a VBL task that has access to cursor handles loaded by the main routine, either through the application’s globals or through the data passed to it as a parameter. In the VBL task, call `SetCursor()` to continuously...
cycle through all of the cursors. Before changing the cursor, check the low-memory global, CrsrBusy, by calling LMGetCrsrBusy(). If this routine returns true, then the VBL task cannot change the cursor.

You don’t want your VBL task to execute when your application goes into the background, so use a system-based VBL task that does not get executed when the application goes to the background.

Unlike SetCursor(), SetCCursor() moves memory. It cannot be called from a VBL task. This means that you cannot use this method to spin a color cursor.

The following code gets the handles to four cursors that are used to display a rotating cursor and then installs a VBL that circulates through the four cursors by calling SetCursor(). This code is written for 68K Macintoshes.

```c
/* **************************************************************************/
/* number of VBL interrupts before task executes
#define kInterval 6

/* structure to piggy back pointer to application's
// A5 world pointer on task structure to pass to
// VBL task
typedef struct VBLInfo
{
    VBLTask      myVBLTask;
    long         vblA5;
}
VBLInfo, *VBLInfoPtr;

VBLInfo myVBLInfo;

pascal long GetVBLInfo (void)
    = Ox2E88;       // MOVE.L AO,(SP)

CursHandle hCursArray[4];
short     cursorCount = 0;

/* ***********************
void InstallVBL ()
{
    hCursArray[0] = GetCursor(128);
    hCursArray[1] = GetCursor(129);
    hCursArray[2] = GetCursor(130);
```
hCursArray[3] = GetCursor(131);

myVBLInfo.myVBLTask.qType = vType;
myVBLInfo.myVBLTask.vblAddr = NewVBLProc( MyVBLTask);
myVBLInfo.myVBLTask.vblCount = kInterval;
// store pointer to application's A5 world
myVBLInfo.vblA5 = SetCurrentA5();

VInstall((QElemPtr) &myVBLInfo.myVBLTask);

void MyVBLTask()
{
    long curA5;
    VBLInfoPtr recPtr;

    recPtr = (VBLInfoPtr) GetVBLInfo();
    // setup app's A5 world
    curA5 = SetA5(recPtr->vblA5);

    // process the application's globals in here
    SetCursor( *(hCursArray[curorCount++ % 4]) );

    curA5 = SetA5(curA5);

    // Reset vblCount so VBL task runs again
    recPtr->myVBLTask.vblCount = kInterval;
}

Related FAQs

- See also FAQ 3-12, “How do I put a spinning cursor in my program?”
- See also FAQ 3-14, “Why does my VBL task crash when I try to spin a cursor?”
- See also FAQ 21-1, “When do I use VBL tasks, and when do I use the Time Manager?”

Related Topic

- See also Chapter 22, “Processes: Vertical Rerease Manager.”
Why does my VBL task crash when I try to spin a cursor?

Here are some rules that are related to cursors and VBL tasks:

- Do not call GetCursor() from the VBL task. Load the cursors from the main program and then access them from your VBL task.

- Do not call SetCCursor(), since it can move or purge memory. You can only deal with universal header cursors from within a VBL task.

- Lock the cursor handles in your main program before you call the VBL task.

- Before you alter the cursor, check the global variable CrsrBusy to see if another process is modifying the cursor. If CrsrBusy is true, then do not call SetCursor().

- Make sure that your VBL task gets disabled when your application gets moved into the background so it does not try to modify the cursor then.

Related FAQs

- See also FAQ 3-6, “Why does my program work fine with a black-and-white cursor but crashes with a color cursor?”

- See also FAQ 3-12, “How do I put a spinning cursor in my program?”

- See also FAQ 3-13, “How can I spin the cursor using an interrupt routine?”

- See also FAQ 21-1, “When do I use VBL tasks, and when do I use the Time Manager?”

Related Topic

- See also Chapter 22, “Processes: Vertical Retrace Manager.”
How do I animate a color cursor?

You cannot animate a color cursor from a VBL or Time Manager task since, unlike the original SetCursor() call, SetCCursor() can move or purge memory. InitCursorCtl(), SpinCursor(), and RotateCursor() only work with monochrome cursors (‘CURS’ resources), not with color cursors. If you want to animate a color cursor, you must call SetCCursor() from your application at regular intervals.
Development Environment and Language Issues

We programmers spend a great deal of time in front of our computers writing code. Because of this, some of the most important programs we work with are those that make up our development environment. Two of the most common C development environments are Metrowerks CodeWarrior and Symantec C++. This chapter discusses a few issues that deal with these products.

Also discussed in this chapter are issues that deal with Assembly and Pascal as they pertain to C programming. I can proudly say that I am a C programmer. At the same time, I must admit that I try to avoid Assembly programming whenever possible. However, there are many times when Assembly language is necessary. Since most of the development environments currently do not allow or have just started allowing you to include PowerPC Assembly in your programs, the discussion here is limited to 68K Assembly. For the most part, this is not problem, since many of the issues that required you to use Assembly on the 68K do not require you to use it on
the PowerPC. It is also important to have at least a rudimentary knowledge of Pascal. Since the Macintosh Toolbox was written in Pascal and most of the Inside Macintosh manuals are presented in Pascal, understanding certain issues related to the language will make your life a little easier. This chapter contains a few questions that routinely get asked about Pascal issues.

There are three subsections in this chapter:

- Pascal Issues
- Assembly and C
- Development Environments

Pascal Issues

This section contains FAQs that deal with the Pascal language. The Toolbox was originally written in Pascal. That is, the way Inside Macintosh presents its calls, and most of its examples are in Pascal. So there are a few aspects of the Pascal language that are important to know when going through the Inside Macintosh volumes.

What are the differences between C and Pascal that I need to know about in order to work with Inside Macintosh?

The function definitions and sample code in Inside Macintosh are written in Pascal. Here are some differences between C and Pascal that will help you work with Inside Macintosh while doing C Mac Programming:

- A function parameter that is preceded by VAR indicates a variable passed by a reference. When this function is called, the address of the variable must be passed as the parameter.

- Pascal automatically passes arguments greater than four bytes by their address. In C, any parameter that is greater than four bytes in size must
have its address explicitly passed as the parameter. The function
definition in *Inside Macintosh* does not require a VAR designation in
front of the parameter.

A common example is a function that takes a Rect as a parameter, such
as InvalRect(), which is listed in *Inside Macintosh* as:

```pascal
PROCEDURE InvalRect( badRect: Rect );
```

When called from a C routine, it looks like this:

```c
Rect myRect;
InvalRect( &myRect );
```

- Hexidecimal values are denoted in Pascal by a '$'. For example,
  $ABCD$ is equivalent to 0xABCD.

- Addresses are denoted by '@' in Pascal. In C, they are denoted by '&'.
  A pointer to a structure references its items via '.' in Pascal. In C, '->'
  is used.

- The "is equal to" comparison symbol in Pascal is a single '='. In C, it is
  '=='. Make sure you use '==' for "logical equal" in your C code when
  you are translating a block of code from *Inside Macintosh*.

- Pascal functions do not return a value with a return statement. They
  assign the function name the value to be returned in the code. For
  example:

```pascal
FUNCTION GetLargerNumber( x, y : Integer )
BEGIN
  IF x > y THEN
    GetLargerNumber := x;
  ELSE
    GetLargerNumber := y;
END
```

- When casting a value to another type in Pascal, the parentheses
  surround the value, as in `WindowPtr(nil)`. In C, the parentheses
  surround the type, as in `(WindowPtr) nil`.

- Pascal strings are sized strings whose first byte contains the number of
  characters in the string. C strings are null-terminated where the end of
  the string is marked by a null character.
When do some routines have to be declared with the “pascal” keyword?

Any routine that gets called by the Toolbox must be declared with the ‘pascal’ keyword. This is because whenever the Toolbox calls a call-back function, it uses the Pascal convention for passing parameters onto the stack. Examples of routines called by the Toolbox are call-back functions (such as scroll bar procedures and sound callback procedures) as well as VBL tasks, Time Manager tasks, and event filters for dialogs.

Assembly and C

This section contains FAQs that deal with integrating Assembly language into your C project. Since most of the development environments do not allow or have just started allowing you to include PowerPC Assembly language in programs, the FAQs in this section all deal with the 68K Assembly language.

What do ONEWORDINLINE(), TWOWORDINLINE() and THREEWORDINLINE() macros in the universal header files do?

ONEWORDINLINE, TWOWORDINLINE, and THREEWORDINLINE are used in inline Assembly function definitions.

ONEWORDINLINE(trapNum) is a macro that expands to:

= trapNum

TWOWORDINLINE(w1, w2) is a macro that gets expanded to:

= {w1, w2}

The arguments get placed on the stack or in registers, and the Assembly commands get executed. So the following lines:

extern pascal void CloseDialog(DialogRef theDialog)
ONEWORDINLINE(0xA982);
Are PowerPC and 68K Assembly the same?

No. Any 68K Assembly code that you have written will not work on your Power Mac.

How do I put Assembly code in my CodeWarrior C project?

The PPC compilers do not include a built-in assembler. However, the 68K compiler does have a built-in assembler that allows you to include Assembly code in your C programs.

A block of Assembly code cannot exist within a C function. It must be inside an Assembly function that is declared with the asm qualifier as follows:

```c
asm void DoAsmFunc( void )
{
    rts
}
```

The Assembly function can be called like any other function:

```c
asm void DoAsmFunc( void );

void main(void)
{
    DoAsmFunc();
}
```

Here are some rules that must be followed when putting Assembly in CodeWarrior projects:

- A block of Assembly code cannot exist within a C function. It must be inside an Assembly function that is declared with the asm qualifier.
- The Assembly function must end with an RTS statement.
Comments must use the C or C++ convention (designated by // or /* */).

Hex constants must be in C style, not Pascal style (0xABCD, not $ABCD).

**How do I put Assembly in my Symantec C project?**

At this time, inline Assembly is only supported in 68K applications. In the Symantec environment, 68K applications are built with the Think C Project Manager.

In a Think C project, inline Assembly can appear anywhere a C statement can appear. It must lie within a C function as in the following example:

```c
void MyFunction()
{
    extern WindowRecord theWindow;

    asm
    {
        move.1 theWindow.refCon, d0
    }
}
```

**How do I access global variables in an Assembly routine in a CodeWarrior program?**

An Assembly routine in a CodeWarrior C program can reference global variables by name. The following sample Assembly program copies the contents of the global variable myGlobalInt into the d0 register:

```c
int myGlobalInt;

asm void myFunc(void)
{
    // move myGlobalInt into d0
    move.w myGlobalInt, d0

    rts
}
```
How do I access variables in a block of inline Assembly in my Symantec C program?

Variables that are accessible to the C function in which the inline Assembly exists can be referenced by name by the inline Assembly code, as follows:

```c
//************
void MyCFunction( long lValue )
{
    asm
    {
        move.l lValue, d0
    }
}
```

How do I pass Assembly language function parameters in a CodeWarrior C project?

You can pass parameters to an Assembly language function and declare them in the function as you would a C routine, as shown in the following example. To access them in the routine, call `fralloc`. This makes space on the stack for the local stack variables and reserves registers for the local register variables. You can then reference the routine parameters and local variables by name. Upon exit, call `frfree` to free the stack and restore the registers.

```c
//**************
static asm short MyFunc( short myParam )
{
    short myLocal;

    fralloc +

    move.w     myParam, myLocal
    move.w     myLocal, d0

    frfree
    rts
}
```
Development Environments

This section contains FAQs that deal with two of the most popular C development environments: Metrowerks CodeWarrior and Symantec C++.

What is the difference between CodeWarrior Gold, Academic, Bronze, and Discover Programming for Macintosh?

CodeWarrior Gold is the full CodeWarrior product. You get support for C, C++, Pascal, and Java (the front end) and can generate code for 68K, PowerPC, Windows NT, Windows 95, BeBox, and MagicCap (the back end). More front ends and back ends are being created all the time, so CodeWarrior Gold definitely gives you the most "bang for your buck." You also get MPW, MacApp, PowerPlant, and the complete Microsoft Foundation Classes (MFC), along with a bunch of cool tools and utilities. CodeWarrior Gold comes with two free updates.

CodeWarrior Academic is the same package as Gold, except that it is not licensed for commercial development (freeware and shareware is fine), and you need to be a student to purchase it.

CodeWarrior Bronze is like Gold and Academic, except that it only generates code for a 680x0 Macintosh. Bronze-generated applications will run under emulation on a Power Macintosh.

Discover Programming for Macintosh is a complete 680x0-version of CodeWarrior (you can save, create new projects, and so forth), but it was designed as a teaching tool. It comes with a series of books that take you through the process of programming the Macintosh. The first book on the Discover CD, the new 2nd edition of Learn C on the Macintosh, assumes no previous programming experience and teaches you C. The rest of the books teach you C++, the Mac Toolbox, and PowerPlant. Each book comes with an associated Apple Guide file that launches the book projects, and a Guide file that contains a list of cool programming-related web sites. (The Guide file sends Apple events to NetScape to get you to a selected site.) Discover is $79. If you upgrade from Discover to Gold, you get your $79 back.
Can Symantec C++ version 8.0 produce 68K executables?

The Symantec C++ 8.0 compiler does not compile 68K code; it only produces PowerPC native code. Symantec C version 7.0 is included with the Symantec C++ 8.0 compiler to produce the 68K code. However, Symantec will most likely produce a 68K compiler that runs in the 8.0 environment soon.

Topic-Related FAQs

- See also FAQ 8-41, "Why do I get the compiler error, "Error: 'ioNameptr' is not a struct/union/class member", when I try to access the ioNameptr field of an HParam BlockRec structure?"

- See also FAQ 29-1, "How do I convert a C string to a Pascal string and vice versa?"

- See also FAQ 29-5, "Where can I find functions like strcpy and strcat for Pascal strings?"
This chapter deals with issues relating to communicating via the serial ports.
You'll learn how to access the serial ports from within your programs, how to get and change serial port characteristics such as baud rate and input buffer sizes, how to send and receive data using the serial port, and more.

How do I access the serial ports?

The standard serial ports for the Macintosh are the modem port and the serial port. The Serial Driver uses separate device drivers for the input and output functions, so you need to open both drivers for two-way communication. On Macintosh computers with two serial ports, you access the modem port through the .AIN and .AOut drivers, and the printer port through the .BIN and .BOUT drivers. On computers with only one serial port, there is only one serial channel, which you access through the .AIN and .AOut drivers. This serial port can be used for either modem or printer connections.

The serial port drivers are opened by Device Manager OpenDriver() or PBOpen() functions. The output driver should always be opened first because the Serial Driver initializes its local variables for both the input and output drivers when you open the output driver. Opening the output driver also installs interrupt handlers and allocates and locks buffer storage for both input and output. If you attempt to open the .BIN or .BOUT driver on a Macintosh with only one serial port, the opening routine will return the openErr result code.
Before you close the serial drivers, you should call \texttt{KillIO()} to terminate all current and pending operations. \texttt{KillIO()} only needs to be called on the output driver to terminate both input and pending operations. Then you can close the drivers.

Other serial drivers may be installed on a machine along with additional serial ports. To get all of the available serial resources, you should use the Communication Toolbox Resource Manager.

The following routines open and close both drivers of the modem port:

```c
//*******************************
short inputARefNum, outputARefNum;

OSErr OpenTheAPort()
{
    OSErr err;

    err = OpenDriver( "\p.AOut", &outputARefNum);
    if (err != noErr)
        return( err );

    err = OpenDriver( "\p.AIn", &inputARefNum);
    if (err != noErr)
        return( err );
}

//*******************************
OSErr CloseTheAPort()
{
    OSErr err;

    err = KillIO( outputARefNum );
    if (err == noErr)
        CloseDriver( inputARefNum);
    if (err == noErr)
        CloseDriver( outputARefNum );

    return( err );
}
```
Is there any way of finding out the names of all the serial ports available on a particular machine?

The Communications Resource Manager provides routines to iterate through all of the serial devices, as shown in the following example. The function CRMSearch(), when passed a CRMRec structure with crmDeviceID set to 0, returns the first element in the list. Subsequent calls return the next device. The crmAttributes field of the CRMRec pointer returned by CRMSearch() represents a pointer to a CRMSerialRecord structure. From this, you can get the input driver name, the output driver name, and the serial port name.

```c
//****************************
void FindPorts ()
{

  Handle portOutNames, portInNames, names;
  CRMRec theCRMRec, *found ;
  CRMSerialRecord *serial ;
  short deviceId, numFound = 0;

  portOutNames = NewHandle ( OL ) ;
  portInNames = NewHandle ( OL ) ;
  names = NewHandle ( OL ) ;

  do
  {
    theCRMRec.crmDeviceType = crmSerialDevice ;
    theCRMRec.crmDeviceID = 0;

    found = CRMSearch ( &theCRMRec ) ;
    if ( found )
    {
      serial = ( CRMSerialRecord * )
        found->crmAttributes ;
      deviceId = found->crmDeviceID ;
      PtrAndHand ( &serial->outputDriverName ,
        portOutNames , sizeof ( serial->outputDriverName ) ) ;
      PtrAndHand ( &serial->inputDriverName ,
        portInNames , sizeof ( serial->inputDriverName ) ) ;
      PtrAndHand ( &serial->name , names ,
        sizeof ( serial->name ) ) ;
    }
  }

  //****************************
```
while( found );
}

**How can I tell if the Mac only contains one serial port, or if both a printer and modem port are present?**

A quick method for checking the ports is to attempt to open the .Bin or .BOut serial drivers. If a Macintosh has only one serial port, attempting to open the .Bin or .BOut driver will return the openErr result code. However, you may get this error for other reasons.

Another method of checking would be to test the names of all of the drivers. Test for the serial port named “Printer-Modem Port,” which is the port used for both modem and printer communication (as on the Duo and 500 series PowerBooks).

**Related FAQ**

See also FAQ 5-2, “Is there any way of finding out the names of all the serial ports available on a particular machine?”

**What are the default settings when a serial port is first opened?**

When the serial port is initially opened, it is set up to communicate at the following settings:

- 9600 bits per second
- 8 bits per character
- no parity
- 2 stop bits
- CTS/DTR hardware handshaking
- input buffer of size 64 bytes
You can change these settings by calling the functions `SerReset()`, `Control()`, and `SerSetBuf()`.

**How can I configure the serial ports?**

The input and output drivers of the serial ports can be reset with the `SerReset()` command. The values that can be configured are the port speed, data bits, parity, and stop bits. The first parameter of `SerReset()` is the reference number of the driver to be reset. The second parameter is a combination of predefined constants representing the new settings. The valid constants are as follows:

- \texttt{baud300} = \texttt{380}; \{300 baud\}
- \texttt{baud600} = \texttt{189}; \{600 baud\}
- \texttt{baud1200} = \texttt{94}; \{1200 baud\}
- \texttt{baud1800} = \texttt{62}; \{1800 baud\}
- \texttt{baud2400} = \texttt{46}; \{2400 baud\}
- \texttt{baud3600} = \texttt{30}; \{3600 baud\}
- \texttt{baud4800} = \texttt{22}; \{4800 baud\}
- \texttt{baud7200} = \texttt{14}; \{9600 baud\}
- \texttt{baud9600} = \texttt{10}; \{3600 baud\}
- \texttt{baud14400} = \texttt{6}; \{14400 baud\}
- \texttt{baud19200} = \texttt{4}; \{19200 baud\}
- \texttt{baud28800} = \texttt{2}; \{28800 baud\}
- \texttt{baud38400} = \texttt{1}; \{38400 baud\}
- \texttt{baud57600} = \texttt{0}; \{57600 baud\}
- \texttt{stop10} = \texttt{16384}; \{1 stop bit\}
- \texttt{stop15} = \texttt{-32768}; \{1.5 stop bits\}
stop20 = -16384; {2 stop bits}
noParity = 0; {no parity}
oddParity = 4096; {odd parity}
evenParity = 12288; {even parity}
data5 = 0; {5 data bits}
data6 = 2048; {6 data bits}
data7 = 1024; {7 data bits}
data8 = 3072; {8 data bits}

These constants can be combined with the logical OR operation (added). For example, to set the port to 9600 baud, eight data bits, two stop bits, and no parity bit, SerReset() is called as follows:

SerReset( outputRefnum, baud9600 + data8 + stop20 + noParity);

Why do I have to open the output device driver for the serial port before opening the input device driver, even if I'm not going to use it?

When the output driver is opened, the Serial Driver initializes its local variables for both the input and output drivers, installs input handlers, and allocates and locks buffer storage for both drivers.

How can I increase the size of the input buffer over the default 64-byte buffer?

When a serial port is open it uses as its input buffer the driver's internal 64-byte buffer. This may not be large enough if you are going to be receiving large amounts of data. You can specify a larger buffer to be used by the input serial driver with the
SerSetBuf() command. Create a buffer, and pass a pointer to it to the appropriate input driver. It is essential that you restore the original serial buffer before your program exits by calling SerSetBuf() with a value of zero for the serBLen parameter.

The following example routines set a 1K buffer as the input buffer of the .AIIn driver and then restore the original serial buffer:

```c
//*************
Handle hSerBuffer;

OSErr IncreaseInputBufSize( short inputRefNum)
{
    OSErr err;

    hSerBuffer = NewHandle( 1024 );
    HLock( hSerBuffer );

    if ( hSerBuffer != nil )
        err = SerSetBuf(inputRefNum, *hSerBuffer, 1024);

    return(err);
}

void RestoreInputBuffer(short inputRefNum)
{
    SerSetBuf(inputRefNum, *hSerBuffer, 0);
    HUnlock( hSerBuffer );
    DisposeHandle( hSerBuffer );
}
```

**What transmission rates can the serial port support?**

The Serial Driver can support transmission rates ranging from 300 to 57600 baud. The actual rates supported are dependent upon the hardware.
How do I send and receive data over the serial port?

After the device drivers of the serial port have been opened and the handshaking mode and data transmission parameters have been set, you can write to and read from the serial ports by using the PBWrite() and PBRead() commands. The input driver reference number is specified in the iorefnum field of PBRead(), and the output driver reference number is specified in the iorefnum field of PBWrite().

PBRead() and PBWrite() allow you to read and write data asynchronously. You can also use FSWRead() and FSWrite() — however, these will only read and write synchronously.

How can I tell when data has been read by the serial port into the input buffer?

You can call SerGetBuf() to get the number of bytes in an input driver’s buffer. If you are using an asynchronous PBRead() call, you can test the ioreult field. When PBRead() is executed, it returns immediately and sets the field ioreult=1. When it reads in new data, it resets the ioreult field to noErr.
As its name implies, this chapter focuses on the Dialog Manager, Alert Manager, and Control Manager. You'll learn how to use a dialog filter procedure (a 'filterproc') to prevent the echo of characters typed in an editable text field, disable a dialog box's OK button if a text field is empty, and restrict the set of values entered in a TextEdit field. You'll learn about user items, how to set a dialog box's default and cancel item, and the right way to put that thick border around the default item. You'll also learn how to move a control, get a control's type, and activate and deactivate a control.

There are six subsections in this chapter:

- Definitions
- Putting Up Dialog Boxes
- Controls
- Working with Dialog Box Items
- Using Dialog Event Filter Functions
- Alerts
Definitions

This section defines two important terms that are discussed in this chapter: CDEF and user item.

What is a control definition function ('CDEF')?

Control definition functions determine the appearance and behavior of a control. They are stored as code resources of type 'CDEF'. The System file contains three standard control definition functions, as follows:

- The ‘CDEF’ with ID 0 defines buttons, check boxes, and radio buttons.
- The ‘CDEF’ with ID 1 defines scroll bars.
- The ‘CDEF’ with ID 63 defines pop-up menus.

Customized control definition functions can be created to define new types of controls and variations of basic controls. Sliders and dials are examples of nonstandard controls that can be created with a ‘CDEF’.

What is a user item?

A user item is a type of dialog box item that designates a rectangle in the dialog box. The rectangular area of the user item responds to mouse clicks and causes ModalDialog() to return with the itemHit parameter set as the user item’s item number. Updating the user item is also handled automatically. When any portion of the user item’s rectangle needs to be redrawn, a user-defined drawing procedure is automatically called. This enables a control that consists of pictures, formatted static text, or any user-defined graphics to be handled easily in a dialog box.

Putting Up Dialog Boxes

This section answers FAQs about putting dialog boxes in an application. There are three types of dialogs boxes: modal, movable modal, and modeless. Each dialog box type requires different code to process its interaction with the user. This section shows how this is done.
How do I put up a modal dialog box?

Processing user interaction with a modal dialog box is a relatively simple task thanks to the routine ModalDialog(). The easiest way to display a dialog box is to create a dialog box resource and retrieve it with GetNewDialog(). To begin user interaction with the dialog box, set the current port to the dialog box and call ModalDialog(). The function returns each time the user clicks on an item. As a result, the item number of the dialog box item selected by the user is returned in the second parameter. Standard item number values for the OK and Cancel buttons are 1 and 2, respectively. Processing usually continues by calling ModalDialog() in a loop until one of these items is selected, at which time the appropriate action is performed in response to the selected item.

The first parameter of ModalDialog() accepts a pointer to a custom filter function. A custom filter lets you examine and modify all events passed to the dialog box for more control.

When you create a modal dialog box with the NewDialog() or NewCDialog() calls, specify the constant dBoxProc (a value of 1) as the dialog box's definition ID. In your resource editor, select the dialog box that corresponds to this value.
The following example gets and displays a modal dialog box with resource ID 128:

```c
void IncludeEventFilter()
{
    DialogPtr myDialog;
    GrafPtr oldPort;
    short itemHit;

    myDialog = GetNewDialog( 128, nil, (WindowPtr)-1 );
    GetPort(&oldPort);
    SetPort(myDialog);

    do
    {
        ModalDialog(nil, &itemHit );

        switch (itemHit)
        {
            case 3:
                // do action for item 3
                break;
            case 4:
                // do action for item 4
                break;
        }
    } while ( itemHit != 1 && itemHit != 2 );
    SetPort( oldPort);
}
```
How do I put up a movable modal dialog box?

A movable modal dialog box can be moved but not made inactive by another window in its application. It does, however, allow the user to bring another application to the front by either clicking on a window of the application or selecting the application’s name from the Application or Apple menu. The movable modal dialog box has a title bar but no close box, as shown in Figure 6-3.

The processing of a movable modal dialog box is implemented the same way as a non-modal dialog box, except that it is more restrictive in its event handling. The user must respond to it before performing any other work in the same application. If the user clicks on another window in the application while the movable modal dialog box is displayed, the application should play the alert sound. When the user clicks a button to perform the action, the dialog box should close.

When you create a movable modal dialog box with the NewDialog() or NewCDialog() calls, specify the constant movableDBoxProc (a value of 5) as the dialog box’s definition ID. In your resource editor, select the dialog box that corresponds to this value.

How do I put up a modeless dialog box?

A modeless dialog box does not require the user to respond to it before doing anything else. It has the appearance of a document window, including a close box (see Figure 6-4), and can be moved, made active and inactive, and closed like a standard document window.
A modeless dialog box is put into a program by first loading it and making it visible, the same as with a modal dialog box. When an event is returned by `WaitNextEvent()`, it is first passed to the call `IsDialogEvent()` to determine if the event is related to the dialog box. If `IsDialogEvent()` returns false, then the event is not related to a dialog box and should be handled normally. If true is returned, the event should be passed to `DialogSelect()`, which handles most of the events relating to dialog boxes. If the event involved an enabled item (such as a mouse-down event in a button or a key-down event with an active TextEdit field in the dialog box), `DialogSelect()` returns true and passes back the item number of the item and a pointer to the dialog box. The item should be processed as it would with `ModalDialog()`.

The modeless dialog box should not close itself when the user clicks on a button to perform an action. Instead, it should remain open to allow the user to repeat the action.

When you create a modeless dialog box with the `NewDialog()` or `NewCDialog()` calls, specify the constant `noGrowDocProc` (a value of 4) as the dialog box’s definition ID. In your resource editor, select the dialog box that corresponds to this value.

**How do I create a color dialog box?**

If you are creating a dialog box from scratch, call `NewColorDialog()`. If you are getting a dialog box from a resource file, add a ‘dctb’ (dialog color table) resource with the same ID as your dialog box. The presence of this resource tells the Dialog Manager to open a color dialog box. If the resource is not there, the Dialog Manager will create a black-and-white dialog box.
Related FAQs

- See also FAQ 6-30, “How can I set the fonts and colors of dialog box items?”
- See also FAQ 33-16, “How can I set up custom colors in a window?”
- See also FAQ 33-18, “Why can I only draw in the eight basic colors in a window?”

**Why can't I draw on a dialog box when it comes up? Why do the dialog box items act as if they are located at different coordinates?**

You must make the dialog box the current GrafPort by calling SetPort(). GetNewDialog() does not set the DialogPtr to be the current port.

**How can I put a PICT item in a background of a dialog box without obscuring the other items?**

There are two problems to overcome here. The first is drawing the picture so that it doesn’t obscure other items in the dialog box. The second is preventing the background picture from masking mouse-down events from other items. The easiest way to do this is to hide the picture item to prevent it from receiving mouse-down events, manually draw the picture, and then draw the controls.

After you have created a dialog box with GetNewDialog() but before it is made visible, call GetDialogItem() to get the handle to the ‘PICT’ referenced by the picture item and the coordinates of the PICT in the dialog box. Next, you hide the picture item with HideDialogItem() so you won’t have to worry about mouse-down events being masked from other items. To draw the picture, call DrawPicture() in response to update events. (If the dialog box is a modal dialog box, this will be done in a filter function.) After the picture is drawn by DrawPicture(), you can draw the rest of the controls by calling UpdateDialog().

Related Topic

- See also Chapter 24, “QuickDraw: Pictures.”
Why can I only draw in the eight basic colors in a dialog box?

Make sure you have a color dialog box. NewCDialog() will create a color dialog box, and GetNewDialog() will return a dialog box resource. If the resource has a 'dctb' associated with it, it is a color dialog box.

Related FAQ
- See also FAQ 33-18, "Why can I only draw in the eight basic colors in my window?"

Related Topics
- See also Chapter 2, "Color."
- See also Chapter 23, "QuickDraw: Drawing."

Why can't my modal dialog box detect disk-insert events?

ModalDialog() calls GetNextEvent() with diskEvts masked out. Consequently, the Event Manager won't handle diskEvts and won't mount disks. You can change this by altering the System event mask that is stored in the global variable SysEvtMask, located at the address 0x0144. To do this, get the System event mask by calling LMGetSysEvtMask() and set the disk event bit (represented by the constant diskEvts). Then store the modified mask back into the System event mask with LMSysEvtMask(). The Event Manager will automatically mount all disk drives normally when you call ModalDialog().

Related Topic
- See also Chapter 7, "Events."

What are Apple's user-interface guidelines concerning the placement of items in a dialog box?

When you create your own dialog or alert boxes, you should follow Apple's user-interface guidelines that contain suggestions for placing and naming (among other things) dialog box items.
In a properly aligned dialog box, the OK button is in the lower-right corner. If a Cancel button exists, it should be placed to the left of the OK button. Apple recommends using 13 and 23 white pixels to separate items in dialog and alert boxes (see Figure 6-5). You can find all the information about dialog and alert box design guidelines in the Dialog Manager chapter of Inside Macintosh: Macintosh Toolbox Essentials.

This shows the proper accepted spacing of buttons and text in alert and dialog boxes.

Figure 6-5: This figure shows an example of proper spacing of buttons and text in dialog boxes.

Controls

This section answers FAQs about controls in dialog boxes and in windows. A few questions are dedicated to the default buttons in a dialog box. They deal with how the default items are set and how the thick border is drawn around the default button. Also included are the answers to the common questions of how to move controls and how to put scroll bars in your application's window.

How do I get the thick border around the default button in a dialog or alert box?

There are a couple of ways to draw the thick border around the default button, each having its own advantages and disadvantages. The first method is to surround the default button with a user item and have the draw procedure of the user item automatically draw the border. This method requires more work to set up but allows for the maximum flexibility. To draw the border, the rectangle of the item is retrieved with GetDialogItem() and expands it by 4 pixels in each direction by calling InsetRect(). The pen size is adjusted to size 3,3. Next, the proper color of the border must be determined. If the button is active, then it is drawn in black.
If the button is dimmed, then it is drawn in the best available gray, as determined by the call GetGray(). If a gray is not available, then the border is drawn in a bitmapped gray. The border is drawn with a call to FrameRoundRect() specifying the ovalWidth and ovalHeight parameters as 16. The following example draws a border around the default item using this method:

```plaintext
void ManuallySetDefault()
{
    DialogPtr myDialog;
    GrafPtr oldPort;
    short itemHit, itemType;
    Rect itemRect;
    Handle itemHandle;
    ModalFilterUPP my ModalFilter Proc;

    myModalFilterProc = NewModalFilterProc(MyDlogFilter);

    myDialog = GetNewDialog(128, nil, (WindowPtr)-1);
    GetPort(&oldPort);
    SetPort(myDialog);

    // install draw procedure to draw thick border around
    // default button
    GetDialogitem(myDialog, kDefaultUserItemld, &itemType,
                   &itemHandle, &itemRect);
    SetDialogitem(myDialog, kDefaultUserItemld, itemType,
                  (Handle) NewUserItemProc(FrameDefaultButton),
                  &itemRect);

    do
    {
        ModalDialog(myModalFilterProc),
        &itemHit);
    } while (itemHit != 1 && itemHit != 2);

    SetPort(oldPort);
    DisposeDialog(myDialog);
}
```

Pascal void FrameDefaultButton( DialogPtr myDialog, short itemNum )
{ short itemType; Rect itemRect; ControlHandle theControl; RGBColor fgColor, rgbBlack = {0x0000, 0x0000, 0x0000}, rgbNew = {0xffffff, 0xffffff, 0xffffff}; Boolean bUseGray = false;

GetDItem(myDialog, 1, &itemType, (Handle *)&theControl, &itemRect);
InsetRect( &itemRect, -4, -4);

// check if control is active
if ( (*theControl)->contrlHilite != 0 )
{
    if (GetGray(GetMainDevice(), &rgbBlack, &rgbNew))
    {
        GetForeColor( &fgColor );
        RGBForeColor(&rgbNew);
        bUseGray = true;
    }
    else
    {
        PenPat(&qd.gray);
    }
}
else
{
    PenPat(&qd.black);
}
PenSize(3,3);
FrameRoundRect(&itemRect, 16, 16);

// reset values
PenSize(1,1);
PenPat(&qd.black);
if (bUseGray)
    RGBForeColor(&fgColor);
The second method is much easier to implement, but it is not fully supported and is accompanied by a minor restriction. In this method, you use SetDialogDefaultItem() to set a button as the default button. It uses SetDialogCancelItem() to set a button as the Cancel button. You then get the standard filter procedure with GetStdFilterProc() and explicitly call it with CallModalFilterProc(). The standard filter procedure must be called in this manner for SetDialogDefaultItem() and SetDialogCancelItem() to work. This causes the thick border to be automatically drawn around the default button and causes the Return and Enter keys to automatically activate it. This also causes the Escape key and the Command-period combination to automatically activate the Cancel button. The restriction that I talked about earlier is that this method does not allow the default or Cancel buttons to be dimmed. The following example draws a border around the default item using this method:

```c
//************************
void UseDialogDefault()
{
    DialogPtr myDialog;
    GrafPtr oldPort;
    short itemHit;
    ModalFilterUPP myModalFilterProc;

    myModalFilterProc=NewModalFilterProc(MyOlogFilter);

    myDialog = GetNewDialog( 128, nil, (WindowPtr)-1 );
    GetPort(&oldPort);
    SetPort(myDialog);

    SetDialogDefaultItem (myDialog, 1);
    SetDialogCancelItem (myDialog, 2);

    do
    {  
        ModalDialog(myModalFilterProc), &itemHit);
    }  
    while ( itemHit != 1 && itemHit != 2 );

    SetPort( oldPort);
    DisposeDialog(myDialog);
}
//************************
Pascal Boolean MyDlogFilter( DialogPtr myDialog, EventRecord *pEvent, short *itemHit)
{
    OSerr myErr;
    Boolean returnVal = FALSE;
    GrafPtr oldPort;
    ModalFilterUPP standardProc;

    if(pEvent->what == updateEvt &&
       (WindowPtr) pEvent->message != myDialog)
    {
        DoUpdateWindow((WindowPtr) pEvent->message);
    }
    else
    {
        GetPort(&oldPort);
        SetPort(myDialog);

        myErr = GetStdFilterProc(&standardProc);
        if(!myErr)
        {
            returnVal = CallModalFilterProc(standardProc, myDialog, pEvent, itemHit);

            SetPort(oldPort);
        }
        return (returnVal);
    }
}

Related FAQs

See also FAQ 6-13, “How do I set the default item and the default cancel item in a dialog box?”
How do I set the default item and the default cancel item in a dialog box?

If a dialog box does not have an event filter associated with it, then the button with an ID of 1 is treated as the default item and the button with an ID of 2 is treated as the default cancel item.

There are also a couple of undocumented calls (warning lights are flashing), SetDialogDefaultItem() and SetDialogCancelItem(), that you can use to set the default and cancel items. (These calls are defined in the universal headers.) They do, however, limit your control over the dialog box — you will not be able to dim the default or Cancel buttons.

If you have an event filter associated with your dialog box, you will have to process the default and cancel items inside of it.

Related FAQs

- See also FAQ 6-12, “How do I get the thick border around the default button in a dialog or alert box?”
- See also FAQ 6-14, “How do I deactivate the default button?”
- See also FAQ 6-36, “When I add an event filter to ModalDialog(), how do I get the Return key to activate the default item?”

How do I deactivate the default button?

Deactivating the default button visually dims the button. If the button is in a dialog box executed by ModalDialog() with no filter procedure specified, mouse selection of the button is also disabled. This does not, however, prevent the user from activating the button with the Return or Enter key. To do this, you must use your own event filter procedure.
If you use `SetDialogDefaultItem()` in conjunction with `GetStdFilterProc()` and explicitly call the standard filter, deactivating the button has no effect. Controls are deactivated with the following call:

```c
HiIliteControl( theControl, 255 )
```

**Related FAQs**

- See also FAQ 6-12, “How do I get the thick border around the default button in a dialog or alert box?”
- See also FAQ 6-13, “How do I set the default item and the default cancel item in a dialog box?”
- See also FAQ 6-36, “When I add an event filter to ModalDialog(), how do I get the Return key to activate the default item?”

**How do I make the buttons the same color as my color dialog boxes?**

To change the color of a button, you have to add an item color table (`ictb`) resource that allows you to specify colors of the different parts of dialog box items.

**Related FAQ**

- See also FAQ 6-30, “How can I set the fonts and colors of dialog box items?”

**Related Topic**

- See also Chapter 2, “Color.”

**How do I get controls to dim with light gray instead of bitmapped gray?**

When you dim a control using `HiIliteControl( theControl, 255 )`, the Control Manager will automatically handle the dimming process. If the dialog box of the control is a color dialog box, then a light-shaded gray will be used to draw the item. If the dialog box is not a color dialog box, the item will be drawn using the
bitmapped gray. To create a color dialog box, make sure that the dialog box has an associated 'dctb' resource of the same resource ID in the resource file or call NewCDiag( ) in your program.

You can get the proper shade of gray by calling the function GetGray( ) and passing black-and-white RGB values to it. Next, you set the foreground color to the desired shade of gray with RGBForeColor( ). Then draw. (You can also use this method if you want to dim the thick border around the default button in your dialog box.) If no gray is available for GetGray( ) to return, you can draw the item in the bitmapped gray.

The following code gets the best available gray on the main graphics device with GetGray( ). If none is available, GetGray( ) returns false and a bitmapped gray is used to draw.

```c
//************************
void DrawInGray()
{
GDHandle theDevice;
RGBColor fgColor,
rgbBlack = {0x0000, 0x0000, 0x0000},
rgbNew = {0xFFFF, 0xFFFF, 0xFFFF};
Boolean bUseGray = false;

theDevice = GetMainDevice();
if (GetGray( theDevice, &rgbBlack, &rgbNew))
{
    GetForeColor( &fgColor );
    RGBForeColor(rgbNew);
    bUseGray = true;
}
else
    PenPat(&qd.gray);

// draw in gray
if (bUseGray)
    RGBForeColor(&fgColor);
}
```

Related FAQs

See also FAQ 2-10, "What RGB value does the system use for dimmed buttons, menus, and window titles?"
See also FAQ 6-12, "How do I get the thick border around the default button in a dialog or alert box?"

See also FAQ 6-28, "How do you deactivate (gray-out) static text items and edit text items in a dialog box?"

See also FAQ 23-37, "How can I make text or an object appear dimmed or grayed-out without redrawing it?"

See also FAQ 31-17, "How do I draw dimmed (grayed-out) text?"

How can I determine the type of a control?

GetDlgItemItem() can be used to determine the type of a control, or any dialog box item. It is defined in the universal header files as follows:

```pascal
void GetDlgItemItem(DialogRef theDialog, short itemNo, short *itemType, Handle *item, Rect *box)
```

The type of the dialog box item specified by the second parameter, itemNo, is returned in its third parameter, itemType. The possible values of item types that GetDlgItemItem() may pass back are listed in Table 6-1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>userItem</td>
<td>0</td>
<td>User item</td>
</tr>
<tr>
<td>helpItem</td>
<td>1</td>
<td>Help balloons</td>
</tr>
<tr>
<td>ctrlItem</td>
<td>4</td>
<td>Control</td>
</tr>
<tr>
<td>statText</td>
<td>8</td>
<td>Static text</td>
</tr>
<tr>
<td>editText</td>
<td>16</td>
<td>Edit text</td>
</tr>
<tr>
<td>iconItem</td>
<td>32</td>
<td>Icon</td>
</tr>
<tr>
<td>picItem</td>
<td>64</td>
<td>QuickDraw Picture</td>
</tr>
</tbody>
</table>

If an item is a control, the values listed in Table 6-2 are added to the item parameters to specify which type of control the item is.
Table 6-2:
Control Item Types and Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>btnCtrl</td>
<td>0</td>
<td>Push button control</td>
</tr>
<tr>
<td>chkCtrl</td>
<td>1</td>
<td>Check box control</td>
</tr>
<tr>
<td>radCtrl</td>
<td>2</td>
<td>Radio button control</td>
</tr>
<tr>
<td>resCtrl</td>
<td>3</td>
<td>Other type of control ('CDEF' resource)</td>
</tr>
</tbody>
</table>

Why can't I click on a button after I move it with MoveControl()?

MoveControl() does not affect the DITL resource that maintains the locations of the controls. Consequently, the Dialog Manager does not recognize that the controls have been moved, preventing them from responding to mouse clicks. You must also call SetDialogItem(). The following code moves a control in a dialog box 50 pixels up and to the left:

```c
//****************
void MoveMyControl( short myItemNum)
{
    GetDialogItem(myDialog, myItemNum, &itemType,
                  (Handle *)&theControl, &itemRect);
    OffsetRect(&itemRect, -50, -50);
    MoveControl(theControl, itemRect.left, itemRect.top);
    SetDialogItem(myDialog, myItemNum, itemType,
                  (Handle)theControl, &itemRect);
}
```

Why don't the buttons and other controls get refreshed when my window comes to the front?

Controls in a window do not get automatically redrawn in response to an update event. You have to call DrawControls() or UpdateControls() to draw them.
DrawControls() draws all of the controls that are currently visible in the specified window. UpdateControls() draws all of the controls in the specified region. This region can be the update region of an update event that is stored in the visRgn field of the window by the BeginUpdate() call.

The following code shows commands that would be called in response to an update event for a window. It calls BeginUpdate() to initiate the window update, calls the user-defined drawing function, draws the controls in a window, and completes the window drawing with EndUpdate().

```c
//************************
void ShowMyWindowControls( WindowPtr myWindow )
{
    BeginUpdate( myWindow );
    DoMyWindowDrawing();
    UpdateControls( myWindow, myWindow->visRgn);
    EndUpdate( myWindow );
}
```

**How do I activate and deactivate a control?**

You can activate and deactivate a control with the HiliteControl() command. To make a control active, pass a value of 0 in the hiliteState parameter. To make the control inactive and not respond to mouse clicks, pass a value of 255.

The following example deactivates the button with an item number of itemNum in the dialog box referenced by myDialog:

```c
//***************
void DeactivateMyControl( DialogPtr myDialog, short itemNum )
{
    short itemType;
    Rect itemRect;
    ControlHandle theControl;

    GetDItem(myDialog, itemNum, &itemType,
             (Handle *)&theControl, &itemRect);
    HiliteControl(theControl, 255);
}
```
How do I put scroll bars in a window?

Scroll bars are typically used to display a defined area of a graphics port in a smaller viewing area. By manipulating the scroll bars, you can control which segment of text or graphics is currently displayed. In a more general definition, scroll bars are used to move through a range of values. A minimum value and a maximum value are specified during the configuration of the scroll bars. The current value of the scroll bar defines the position of the thumb that is placed in the scroll bar to represent a ratio of the current value to the range.

To put a scroll bar in a window, you must first create the scroll bar control with NewControl(). The parameters of NewControl() include the window to which the control is defined, the title (for scroll bars, there is no title), the visible flag, the initial value, the minimum and maximum values that define the range, and the procID. The procID parameter specifies the type of control to be created. To create a new scroll bar, pass the constant scrollBarProc.

The following routine creates a vertical scroll bar for a window. For this example, the maximum size of the image that will be displayed in this window is 1000. The maximum value of the scroll bar is defined as this maximum size minus the length of the window.

```c
//******************
#define kVisible true
#define kStartValue 1
#define kMinValue 1
#define kMaxSize 1000
#define kNilRefCon 0L
#define kEmptyTitle "\p"

ControlHandle hScrollBar;

void SetUpScrollBar( WindowPtr window )
{
    Rect vScrollRect;
    short maxValue;

    maxValue = kMaxSize - (window->portRect.bottom - window->portRect.top);

    vScrollRect = window->portRect;
    vScrollRect.top -= 1;
    vScrollRect.bottom += 1;

}  
```
vScrollRect.left = vScrollRect.right - kScrollBarWidth + 1;
vScrollRect.right += 1;

hScrollBar = NewControl( window, &vScrollRect, kEmptyTitle,
                        kVisible, kStartValue, kMinValue, maxValue,
                        scrollBarProc, kNilRefCon);
}

A scroll bar, like other controls, is displayed in a window by DrawControls().
Window Manager routines such as DrawWindow() and ShowWindow() do not
automatically redraw controls in a window. DrawControls() should be called
in response to an update event for the window, as shown in the following
HandleUpdateEvent() routine:

//**************************
HandleUpdateEvent( EventRecord *eventPtr )
{
    WindowPtr window;

    window = (WindowPtr)eventPtr->message;

    BeginUpdate( window );
    DrawControls( window );
    UpdateWindow( window );
    EndUpdate( window );
}

Now let's move on to actually using the scroll bar to manipulate the contents in the
window. 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. The scroll bar is located in the content region of the win-
dow. If the FindWindow() returns that the mouseDown was inContent, then
FindControl() is called to determine if the scroll bar was clicked. FindControl()
passes back a handle to the selected control in its third parameter and returns the
part code indicating what part of the control was selected. You can tell if the scroll
bar was clicked by comparing the selected control passed back from FindControl()
with the handle to the scroll bar that was created in the SetUpScrollBar() routine, scrollBarH. If the control is the scroll bar, you initiate scrolling actions
based upon the part of the scroll bar. If the thumb was the part of the scroll bar that
was clicked, then TrackControl() is called to drag the outline of the thumb up
and down the scroll bar. When the thumb is released, TrackControl() returns.
It is then up to you to force a redraw of the window with InvalRect(). If any
other part of the control was used, call TrackControl() with a pointer to the
scroll bar’s tracking procedure. The tracking procedure, ScrollProc(), is passed
to TrackControl as a universal procedure pointer that was created during the
program’s initialization routine with the following call:

gActionUPP = NewControlActionProc( ScrollProc );

The following routine, HandleMouseDown(), is called in response to a mouseDown
event:

/******************************************
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,
                        &qd.screenBits.bounds );
            break;
        case inContent:
            thePoint = eventPtr->where;
            GlobalToLocal( &thePoint );

            thePart = FindControl( thePoint, window,
                                &theControl );

            if ( theControl == hScrollBar )
                {
                    if ( thePart == inThumb )
                        {
                            thePart = TrackControl( 
                                        theControl, thePoint,
                                        kNilActionProc );
                            InvalRect( &(window->portRect) );
                        }

}
else
{
    thePart = TrackControl(
        theControl, thePoint,
        gActionUPP);
}
}
break;
case inGoAway:
    gDone = true;
break;
}

The next part of the program that we'll look at is the scroll bar's tracking procedure, ScrollProc(). ScrollProc() gets called whenever a mouseDown occurs in the page-up, page-down, up-arrow, or down-arrow region of the scroll bar. These calls are inPageUp, inPageDown, inUpButton, and inDownButton, respectively. First, the current, maximum, and minimum values are determined. If the user clicks in the page-down region, then the current value is increased by the size of the window. If this exceeds the maximum value, then the current value is set to the maximum value. The same procedure is used in response to page-up, except the current value is decreased by the size of the window and the minimum value is used as the limit. If the user clicks the down-arrow button, then the current value is increased by a value we (as the almighty developers) deem appropriate (in the following example, this is 10 pixels). The same procedure is used in response to a click on the up-arrow button, except the current value is decreased by the 10 pixels and the minimum value is used as the limit.

//*******************
pascal void ScrollProc( ControlHandle theControl, 
    short partCode )
{
    short       curCtlValue, maxCtlValue, minCtlValue, 
                pageIncrement;
    WindowPtr   window;

    maxCtlValue = GetCtlMax( theControl );
    curCtlValue = GetCtlValue( theControl );
    minCtlValue = GetCtlMin( theControl );

    window = (**theControl).cont1Owner;
    pageIncrement = ( window->portRect.right -
                      window->portRect.left);
switch ( partCode )
{
    case inPageDown:
        curCtlValue += pageIncrement;
        if ( curCtlValue > maxCtlValue )
            curCtlValue = maxCtlValue;
        SetCtlValue( theControl, curCtlValue );
        UpdateWindow( window );
        break;
    
    case inDownButton:
        curCtlValue += 10;
        if ( curCtlValue > maxCtlValue )
            curCtlValue = maxCtlValue;
        SetCtlValue( theControl, curCtlValue );
        UpdateWindow( window );
        break;
    
    case inPageUp:
        curCtlValue -= pageIncrement;
        if ( curCtlValue < minCtlValue )
            curCtlValue = minCtlValue;
        SetCtlValue( theControl, curCtlValue );
        UpdateWindow( window );
        break;
    
    case inUpButton:
        curCtlValue -= 10;
        if ( curCtlValue < minCtlValue )
            curCtlValue = minCtlValue ;
        SetCtlValue( theControl, curCtlValue );
        UpdateWindow( window );
        break;
}

Well, all of this is nice, except it is all for naught if our window's drawing procedure does not take the current value of the scroll bar into account when drawing. The following UpdateWindow() routine gets the current value of the scroll bar to determine how it should draw the contents. Notice that the scroll bar is excluded from the clipping rectangle so that it does not get drawn over.
CHAPTER 6: Dialogs & Controls

```c
//********************
void UpdateWindow( WindowPtr window )
{
    Rect    windowRect;
    RgnHandle tempRgn;
    short   currentOffset;

    tempRgn = NewRgn();
    GetClip( tempRgn );

    windowRect = window->portRect;
    windowRect.right -= kScrollBarWidth;
    EraseRect( &windowRect );
    ClipRect( &windowRect );

    currentOffset = GetCtlValue(hScrollBar);
    DrawWindowStuff( window, currentOffset );

    SetClip( tempRgn );
    DisposeRgn( tempRgn );
}
```

**Related FAQ**

- See also FAQ 32-14, “How do I put scroll bars in my TextEdit field?”

**What is the size of a standard scroll bar?**

The height of a standard horizontal scroll bar, as well as its width, is 16 pixels.

**Related Topic**

- See also Chapter 33, “Windows.”

**Working with Dialog Box Items**

This section answers FAQs about dialog box items. It discusses how to use user items, deactivate text items, and customize colors and text of dialog box items.
**How do I use user items to draw in a dialog box?**

You can assign to a user item a draw procedure that can draw pictures, text, or anything else. When the area designated by the user item requires an update, the user-defined draw procedure is called.

You are not restricted to keeping the drawing associated with a user item within the boundaries of the user item's rectangle. However, it is a good idea to do so in order to ensure that the user item will update itself whenever necessary. The updates for a user item are generated when the area defined by the user item rectangle requires it. If drawing extends outside the rectangle, then that portion may be erased and require a redraw but no update will be generated.

The following code installs the draw procedure `DrawMyUserItem()` in the user item of an opened dialog box:

```c
//***********
#define kPictId 128
#define kDlogId 129
#define kUserItemid 7

void UseAUserItem()
{
    DialogPtr myDialog;
    GrafPtr oldPort;
    short itemHit, itemType;
    Rect itemRect;
    ControlHandle theControl;

    hPict = GetPicture(kPictId);

    myDialog = GetNewDialog( kDlogId, nil, (WindowPtr)-1 );
    GetPort(&oldPort);
    SetPort(myDialog);

    // install a draw procedure for the user item
    GetDialogItem(myDialog, kUserItemId, &itemType,
                  (Handle *)&theControl, &itemRect);
    SetDialogItem(myDialog, kUserItemId, itemType,
                  (Handle)NewUserItemProc(DrawMyUserItem),
                  &itemRect);

    do
    {
```
CHAPTER 6: Dialogs & Controls

```
ModalDialog(nil, &itemHit);
}
while ( itemHit != 1 && itemHit != 2 );
SetPort( oldPort);

//*************************
Pascal Void DrawMyUserItem( DialogPtr myDialog,
                      short itemNum );
{
  short itemType;
  Rect itemRect;
  ControlHandle theControl;
  GetDialogItem(myDialog, itemNum, &itemType,
               (Handle *)&theControl, &itemRect);
  DrawPicture( hPict, &itemRect);
}
```

**Why doesn’t FindDialogItem() find an item when I pass it the mouse location from an event?**

`FindDialogItem()` expects a point in local coordinates. The cursor location specified in an event is in global coordinates.

**Related FAQ**

- See FAQ 7-16, “Are the mouse coordinates returned in the ‘where’ field of the event structure in local or global coordinates?”

**Why doesn’t FindDialogItem() return the proper dialog box item number?**

`FindDialogItem()` determines which item is located at a particular coordinate in a dialog box. The number returned is zero-based, which means the number of the dialog box item minus one is returned. If the point is occupied by overlapping items (including disabled items), then the item that is first in the item list is returned.
**Why can't I disable an edit text field in a dialog box with HiliteControl()?**

As the name implies, HiliteControl() is used to activate and deactivate controls. Edit text fields are not controls.

To disable a dialog box item, first call GetDialogItem() to get the item. Then set bit 7 of the itemType value (this is equivalent to adding 128 to the value) passed back in the third parameter and call SetDialogItem() with the same parameters. To enable the control, call SetDialogItem() with bit 7 if the itemType value cleared.

**How do I get the text from an editable text item in a dialog box?**

To get the text from an editable (or static) text item in a dialog box, you must first call GetDialogItem() to get a handle to the text item. Then call GetDialogItemText(), which takes the item handle as the first parameter and a Str255 to retrieve the text.

GetDialogItemText() will only retrieve the first 255 characters in an edit text item, regardless of the size of the string pointer.

Related FAQ

See also FAQ 32-3, “How do I get the text of a TextEdit field?”

**How do you deactivate (gray-out) static text items and edit text items in a dialog box?**

There is no command to disable static text and edit text items in a dialog box. These dialog box item types are not controls, so calling HiliteControl() will not work. The text has to be drawn in the proper text mode, grayishTextOr. The problem is that you must make sure that the new text mode only affects the text of the items that you want dimmed. One method of doing this is to customize the low-level text-drawing routine using the SetStdCProcs() function (see the related FAQs for this topic). When QuickDraw calls the low-level text-drawing function,
the pLoci field of the graphics port (in this case, our dialog box), is set to the location of the text to draw. Inside the customized function, get the current pen location and call FindDItem() to determine which dialog box item is about to be drawn. If the item about to be drawn is one that you want highlighted, call TextMode(grayishTextMode) to have dimmed text drawn, and then call the standard text-drawing procedure to have the text drawn. When the standard text-drawing routine returns, reset the text-drawing mode back to its original value.

Related FAQs

See also FAQ 2-10, "What RGB value does the system use for dimmed buttons, menus, and window titles?"

See also FAQ 6-16, "How do I get controls to dim with light gray instead of bitmapped gray?"

See also FAQ 23-3, "What are the QuickDraw 'bottleneck' functions?"

See also FAQ 23-37, "How can I make text or an object appear dimmed or grayed-out without redrawing it?"

See also FAQ 23-39, "How can I replace the standard QuickDraw low-level or 'bottleneck' functions?"

See also FAQ 31-17, "How do I draw dimmed (grayed-out) text?"

How can I have variables in static text fields?

You can specify text to be displayed in static text fields of your dialog or alert box by using the ParamText() call. In the static text field, place a "^0", "^1", "^2", or "^3" marker where you want the variable text to appear (see Figure 6-7). Each of these markers consists of two characters: the circumflex (ASCII 94) and the number.

![Figure 6-7: A dialog box that can accept variables in its static text](image)
To set the value of the text to be inserted into each of these markers call `ParamText()` with the replacement strings, as shown in the following example. (The first parameter of `ParamText()` is the value for "\0" not "\1".) Pass `nil` or "\p" for the strings that you will not need. Each string can be up to 255 characters long. `InitDialog()` initializes the four values to empty strings.

```
//********************
void DisplayNoFileAlert( Str255 myFileName)
{
    ParamText( "\pCannot find the file", myFileName, NIL, NIL);
    StopAlert( 128, NIL);
}
```

**How can I set the fonts and colors of dialog box items?**

When drawing items in an application’s dialog and alert boxes, the Color Manager uses (by default) the same colors used to draw the items in dialog and alert boxes of the System software. (This typically includes a white background.) The Dialog Manager also uses the default System font to draw the text in editable text and static text items. You can customize both the colors and fonts of dialog items by creating an item color table (‘ictb’) resource that has the same resource ID as the dialog or alert box’s item list (‘DITL’) resource. To do this, you must be working with color dialog and alert boxes that have a ‘dctb’ or ‘actb’ resource.

An item color table allows you to specify the font characteristics as well as the font foreground and background colors of editable text or static text items. In controls, you can specify the color of the frame, body, and text.

If you want to create an ‘ictb’ resource yourself, you can study the organization and values for the resource in *Inside Macintosh: Macintosh Toolbox Essentials*. However, there is a better way — you can let Resorcerer do the work for you (if you have it).

To set the color or font of an item with Resorcerer, first open up the ‘DLOG’ resource that contains the item you want to modify. Select the item. Select the Colors and Text Styles menu item from the Item menu. This opens up the Colors and Text Styles dialog box (see Figure 6-8). You use this dialog box to select your color and font settings.
A common action is to set the background color of a button so that it matches the color of a color dialog box. To do this, make a note of the RGB values that you used to set the dialog box content color. This is can be set and retrieved by selecting the Set Dialog Info menu item in the Resorcerer's Dialog menu. Use these same RGB values to set the button colors with the Colors and Text Styles dialog box.

**Related FAQs**

- See also FAQ 6-6, “How do I create a color dialog box?”

- See also FAQ 33-16, “How can I set up custom colors in a window?”

**Related Topics**

- See also Chapter 2, “Color.”

- See also Chapter 31, “Text: Fonts and Drawing.”

**How can I set the background of the text-entry fields in a dialog box?**

- See FAQ 6-30, “How can I set the fonts and colors of dialog box items?”
Using Dialog Event Filter Procedures

This section answers FAQs about event filter functions in dialog boxes and describes how to use a filter function to screen the input of text.

How do I put an event filter in a dialog box?

An event filter is installed for a dialog box by passing a procedure pointer in the first parameter of ModalDialog(). When ModalDialog() is called with no event filter specified, the standard event filter automatically treats Return and Enter key hits as mouse clicks on the default button. When an event filter procedure is installed, this is not done.

The following code installs the event filter, MySimpleDlogFilter(), for a dialog box. The event filter procedure tests to see whether the Return or Enter key has been pressed. If it has, the default button is highlighted for 8 ticks, the item number of the default button is passed back in the itemHit parameter, and true is returned, signifying that the event was processed. It also tests to see whether the Escape key or the Command-period key combination has been pressed. If it has, the Cancel button is highlighted for 8 ticks, the item number of the Cancel button is passed back in the itemHit parameter, and true is returned, signifying that the event was processed. Update events for other windows are also handled to allow the other windows of the application to be refreshed. This allows background applications to update their windows, making up for the fact that when an update event is pending for your application, no other application can be refreshed.

```c
#define kLineFeed 0x0A
#define kCarriageReturn 0x0D
#define kEscape 0x1B
#define kButtonDelay 8

//*************
void IncludeEventFilter()
{
    DialogPtr myDialog;
    GrafPtr oldPort;
    short itemHit;
    ModalFilterUPP myModalFilterProc;
    myModalFilterProc = NewModalFilterProc(MySimpleDlogFilter);
```
myDialog = GetNewDialog( kDlog1Id, nil, (WindowPtr)-1 );
GetPort(&oldPort);
SetPort(myDialog);

do
{
    ModalDialog( MyModalFilterProc,
             &itemHit );
}
while ( itemHit != 1 && itemHit != 2 );

SetPort( oldPort);

**********

Pascal Boolean MySimpleDlogFilter( DialogPtr myDialog,
                   EventRecord *pEvent,
                   short   *itemHit )
{
    short   itemType;
    Rect    itemRect;
    Handle  itemHandle;
    long    finalTicks;
    char    ch;

    switch( pEvent->what )
    {
    case updateEvt:
        // ignore update for the dialog box, it will
        // automatically be handled
        if ( (WindowPtr)pEvent->message != myDialog )
        {
            DoUpdateWindow((WindowPtr)
             pEvent->message);
            break;
        }
    case keyDown:
        // get char code not key code
        ch = pEvent->message & charCodeMask;

        if ( ch == kLineFeed || ch == kCarriageReturn

        break;
    case KeyUp:
        // get char code not key code
        ch = pEvent->message & charCodeMask;

        if ( ch == kLineFeed || ch == kCarriageReturn

        break;
    case ...
GetDlgItem(myDialog, 1, &itemType,  
  &itemHandle, &itemRect);

HiliteControl((ControlHandle)itemHandle,  
  inButton);
Delay(kButtonDelay,&finalTicks);

HiliteControl((ControlHandle)itemHandle,  
  FALSE);

  *itemHit = 1;
  return( TRUE);
}

  // process cancel
else if ( ( ch == kEscape ) ||
          (( ch == '.' ) &&
           (pEvent->modifiers & cmdKey)) )
{
  GetDlgItem(myDialog, 2, &itemType,  
          &itemHandle, &itemRect);

  HiliteControl((ControlHandle)  
              itemHandle,inButton);
  Delay(kButtonDelay,&finalTicks);

  HiliteControl((ControlHandle)  
              itemHandle,FALSE);

  *itemHit = 2;
  return( (ModalFilterUPP) TRUE);
}

default:
  break;
}

return ( FALSE);
Related FAQs

See also FAQ 6-35, "Why do I continuously get update events in my event filter procedure of ModalDialog()?"

See also FAQ 6-36, "When I add an event filter to ModalDialog(), how do I get the Return key to activate the default item?"

**How can I get an edit text item in a dialog box to only accept numbers as input?**

You can restrict user-input to a particular subset by using your own dialog event filter procedure. When the event filter detects a keyDown event that is not the Return, Enter, Escape, or Delete key, it checks to see if the character falls between 0 and 9. If it does, then the character is processed as usual by the standard filter procedure by having the user-defined filter procedure return a false. If the character does not fall within the acceptable range, the character is ignored by having the event filter procedure pass a true, signifying to the standard filter procedure that the event was handled.

The following sample is a user-defined event filter that accepts numbers only and beeps when an invalid character is entered:

```pascal
#define kBackSpace 0x08
#define kLineFeed 0x0A
#define kCarriageReturn 0x01
#define kEscape 0x1B
#define kButtonDelay 8

//***************
pascal Boolean MyNumbersOnlyDialogFilter( DialogPtr myDialog, EventRecord *pEvent, short *itemHit )
{
    :
    switch( pEvent->what )
    {
        :
        case keyDown:
```
// get char code not key code
ch = pEvent->message & charCodeMask;

if ( ch == kLineFeed || ch == kCarriageReturn) {
    // process kLineFeed and kCarriageReturn
}
// process cancel
else if ( ( ch == kEscape ) || (( ch == '.' ) &&
    (pEvent->modifiers & cmdKey) ) )
{
    // process kEscape and Command-period combination
}
else if ( ch == kBackSpace )
{
}
else
{
    if ( ch < '0' || ch > '9' )
    {
        SysBeep(20);
        return( TRUE);
    }
}

default:
    break;
}
return ( FALSE);

Related FAQ

See also FAQ 6-34, “How can I stop an edit text item from echoing typed characters for a password entry?”
How can I stop an edit text item from echoing typed characters for a password entry?

You can process characters input by a user while preventing them from being displayed in the current edit text box by using your own dialog event filter procedure. When the event filter detects a keyDown event that is not the Return, Enter, Escape, or Delete key, it adds the character into a buffer. It then changes the character to some masking character and passes it to the standard event filter by returning false. To do this, it clears the character code contained in the last two bytes of the event.message field by using the following call:

```pascal
pEvent->message &= ~charCodeMask;
```

It then sets the field to a * (asterisk) character by using the following call:

```pascal
pEvent->message |= '*';
```

When the event filter returns false, the standard event filter is called and displays the * character installed by the program into the dialog box.

The following code displays the processing of the keyDown event of a dialog box. It assumes the string strPassword is cleared when the user starts to enter characters.

```pascal
//***************
#define KEscape 0x1B
#define kLineFeed 0x0A
#define kCarriageReturn 0x0D
#define kBackSpace 0x08

pascal Boolean MyPasswordDlogFilter( DialogPtr myDialog, EventRecord *pEvent, short *itemHit )
{
    switch( pEvent->what )
    {
```
case keyDown:
   // get char code not key code
   ch = pEvent->message & charCodeMask;
   if ( ch == kLineFeed || ch == kCarriageReturn )
   {
      // process kLineFeed and kCarriageReturn
   }
   // process cancel
   else if ( ( ch == kEscape ) || ( ch == '.' ) &&
              ( pEvent->modifiers & cmdKey)) )
   {
      // process kEscape and Command-period
      // key combination
   }
   else if ( ch == kBackSpace )
   {
      if ( strPassWord[0] > 0 )
         strPassWord[0]--;
   }
   else
   {
      strPassWord[0]++;
      strPassWord[ strPassWord[0] ] = ch;
      pEvent->message &= ~charCodeMask;
      pEvent->message |= '*';
   }

   default:
      break;
}

return ( FALSE);

Related FAQ

See also FAQ 6-33, "How can I get an edit text item in a dialog box to only accept numbers as input?"
Why do I continuously get update events in my event filter procedure of ModalDialog()?

The update events are not being triggered by your dialog box but by the window below the dialog box in your application. Since you are most likely ignoring events generated by other windows, the update event never gets serviced. To stop the update events from being posted, you must handle the update events from other windows.

Related FAQs

☐ See also FAQ 6-32, “How do I put an event filter in a dialog box?”

☐ See also FAQ 6-36, “When I add an event filter to ModalDialog(), how do I get the Return key to activate the default item?”

When I add an event filter to ModalDialog(), how do I get the Return key to activate the default item?

You have to add the code to manually handle this in your event filter procedure. It is not done automatically like it is when no event filter is passed to ModalDialog().

Related FAQs

☐ See also FAQ 6-12, “How do I get the thick border around the default button in a dialog or alert box?”

☐ See also FAQ 6-13, “How do I set the default item and the default cancel item in a dialog box?”

☐ See also FAQ 6-14, “How do I deactivate the default button?”

☐ See also FAQ 6-32, “How do I put an event filter in a dialog box?”

☐ See also FAQ 6-35, “Why do I continuously get update events in my event filter procedure of ModalDialog()?”
Alerts

This section answers FAQs about alerts. It also describes how you can place your own sound and icon in an alert box.

How do I put up an alert and do I close it?

There are four different calls to display an alert: Alert(), CautionAlert(), NoteAlert(), and StopAlert(). CautionAlert(), NoteAlert(), and StopAlert() display alerts with the standard icons represented by the constants cautionicon (icon ID = 2), noteicon (icon ID = 1), and stopicon (icon ID = 0), respectively. If you have an icon with the same ID as one of the alert icons, the corresponding alert type will display your icon. However, this is not recommended — it is better to use the Alert() call instead.

To display an alert, simply call the appropriate alert function with the ID of the desired alert type ('ALRT') resource. You can pass nil in the filterproc parameter to use the standard procedure.

The alert will automatically close and dispose of itself when you click on an active button, and the item number of the clicked button will be returned.

How can I put my own icon in the upper-left corner of an alert?

To display your own icon in the upper-left corner of an alert, use the Alert() function. Manually put an icon resource in the upper-left corner of the alert’s DITL and assign to it the same resource ID as the icon you would like to display.

You can also display customized icons in CautionAlert(), NoteAlert(), and StopAlert() alerts by including icons with the same IDs as the standard alert icon resource IDs (stop icon ID = 0, caution icon ID = 2, and note icon ID = 1) in your resource file. However, this is not recommended — use Alert() instead.
How can I get an alert to play a sound other than the System alert sound?

By default, the Dialog Manager uses the System alert sound. To play a sound other than the System alert sound, you must create your own customized sound procedure and call the ErrorSound() function to install it.

An alert can have a sound setting defined for each of the four alert stages. The settings represent a sound number that ranges from 0 to 3. When using the System alert sound, sound 0 represents no sound, sound 1 plays it once, sound 2 plays it twice, and sound 3 plays it three times.

A sound procedure receives a sound number as an argument. It should play a sound for sound numbers 1, 2 and 3, and return for sound number of 0, as follows:

```pascal
//*************************
pascal void AlertSoundProc(short sndNum) {

short sndId;
SndChannelPtr myChan = 0L;
Handle mySound;
OSErr err;

if (sndNum == 0)
    return;

switch (sndNum) {
    case 1:
        sndId = 128;
        break;
    case 2:
        sndId = 129;
        break;
    case 3:
        sndId = 130;
        break;
}
```
```c
mySound = GetResource( soundListRsrc, sndId );
err = SndNewChannel( &myChan, 0, 0, OL );
HLock( mySound );
err = SndPlay( myChan, (SndListResource **)mySound, FALSE );
HUnlock( mySound );
err = SndDisposeChannel( myChan, FALSE );
```

//*******************
void MyFunction()
{
    :
ErrorSound( NewSoundProc(AlertSoundProc) );
    :
    Alert( kAlertId, nil );
}

If you call ErrorSound(0), the alert sound is disabled as is the flashing menu bar
that occurs when the volume is set to 0.

If your resource file contains a 'snd' resource with a resource ID of 1, this sound will
be played in place of the System alert sound. Use the ErrorSound() call and a
customized sound procedure instead.

How can I customize the text in an alert box?

See FAQ 6-29, “How can I have variables in static text fields?”

How does the alert stage value get incremented, and how can I reset it?

The alert stage is a number representing how many consecutive times an alert has
been displayed to the user. The alert stage value is incremented if a displayed alert
is the same as the last displayed alert. If it is not the same, then the alert stage is
automatically reset to 0. You can manually set the alert stage to the first stage by
calling ResetAlertStage(). To get the current value of the alert stage, call
GetAlertStage().
CHAPTER 6: Dialogs & Controls

Topic-Related FAQs

- See also FAQ 9-1, "What is the difference between the dialog hook function and the filter function in `CustomGetFile()` and `CustomPutFile()`?"
- See also FAQ 13-2, "Which icon types can I put in dialog boxes and menus?"
- See also FAQ 14-9, "How do I put a list in a Macintosh modal dialog box?"
- See also FAQ 19-2, "How can my application display a dialog box in front of all other windows on the screen when it is running in the background?"
- See also FAQ 32-14, "How do I put scroll bars in my TextEdit field?"
- See also FAQ 33-30, "Can I store my own data in the `refCon` field of a window or dialog box?"
The Event Manager feeds your application's command center by passing along information about user actions such as mouse clicks, keystrokes, and the like. As you make your way through the event-related FAQs, you'll learn how to find out the current state of the keyboard (like which modifier keys are currently down) without waiting for an event as well as how to find out the state of the keyboard when a keyboard-related event was posted. You'll learn how to set up event masks to detect key-up events and how to post events to the event queue. You'll also learn about suspend, resume, mouse-moved, and assorted other events, as well as how to translate character and virtual key codes. This chapter even describes how you can use GNE filters to add special capabilities to your program.

There are five subsections in this chapter:

- Definitions
- Keyboard Events
- Mouse Events
- Processing Events
- GNE Filters
Definitions

This section answers FAQs about some important terms that are discussed in this chapter, including tick count (a simple topic) and GNE filter (a complex topic).

What is a GNE filter?

A GNE filter is a procedure that is called from GetNextEvent() and WaitNextEvent() just before those traps return to an application. (The GNE in the global variable name stands for GetNextEvent(), and the \( j \) stands for “jump.”) When the filter code is called, it is passed the event that will be passed to the application. The filter can examine the event and perform some appropriate processing, change the event and have the modified value passed to the application, or intercept the event by changing it to a null event that is passed to the application.

A GNE filter is installed by placing its address into the low-memory global jGNEFilter, which is located at Ox29A. Multiple filters can be installed by having each one store the address of the filter that was installed before it. When a filter is done executing, it then calls the stored filter.

What is the duration of a tick?

A tick is \( \frac{1}{60} \)th of a second. The number of clock ticks since System startup can be retrieved by calling TickCount().

Keyboard Events

This section answers FAQs about handling keyboard events. A large number of the questions deal with determining the state of the keys (pressed or not pressed) with the GetKeys() function. It also covers how to detect the ever elusive key-up events.

How do I detect if the user is holding a key down?

When the user presses a key, WaitNextEvent() passes back a keyDown event. If the key is held down for a period of time beyond the autoKey threshold, autoKey events are generated continuously until the key is let up. The autoKey
threshold represents the time from the first keyDown event until the autoKey event is generated. The default value is 16 ticks. The autoKey rate is the interval between the generation of autoKey events. The default autoKey rate is 4 ticks. In the autoKey event, the message field of the event structure contains the value of the key pressed.

The following code shows how the autoKey event is processed:

```c
//*************************
void HandleEvent( EventRecord *eventPtr )
{
    WindowPtr theWindow;

    switch ( eventPtr->what )
    {
        case keyDown:
        case autoKey:
            // process key strokes
            break;
    }
}
```

**How do I read the keys of the keyboard without getting an event from the event queue?**

If you want to get the current status of the modifiers key, you could peek at the next event in the event queue with EventAvail() and check the modifiers field of the event record. This, however, will only work if your function is in the foreground. To get the status of any key, you can use GetKeys(). GetKeys() passes back a 128-bit structure with each bit representing the state of a key, identified by its key code. The structure is a group of 16 bytes. The bits within each byte are ordered right to left.

The maximum number of keys for which GetKeys() will reliably return a status is two character keys plus any combination of the five modifier keys.

Figure 7-1 shows the key codes of the Apple Extended Keyboard.
Why isn’t GetKeys() passing back the key values that I expect?

GetKeys() passes back a 128-bit structure with each bit representing the state of a key, identified by its key code. The structure is a group of 16 bytes. The bits within each byte are ordered right to left. This means that if the key with the key code of 0x02 (the D character on the extended keyboard) is pressed, the third key from the right in the first byte will be set. The bit representation is as follows:

```
00000100 00000000 00000000 ...
```

If the key with key code 0x08 is pressed (the C key on an extended keyboard), the bit representation is as follows:

```
0000000 00000001 00000000 ...
```

Note the BitTst() will not work because of this non-sequential numbering scheme. The following example shows how you can test if a key with a specific key code has been pressed:

```c
//******************
short IsKeyPressed(unsigned short keycode )
{
    unsigned char km[16];
    GetKeys( (unsigned long *) km);
    return ( ( km[keycode >> 3] >> (keycode & 7) ) & 1);
}
```

The maximum number of keys for which GetKeys() will reliably return a status is two character keys plus any combination of the five modifier keys.
How can I detect multiple simultaneous keystrokes on a single keyboard?

GetKeys() will allow you to get the status of all the keys on the keyboard. The maximum number of keys for which GetKeys() will reliably return a status is two character keys plus any combination of the five modifier keys.

How can I tell when the Shift, Control, Command, or Option key is pressed?

The state of the Shift, Control, Command, Option, and Caps Lock keys is passed back in the EventRecord modifiers field of the event passed back by WaitNextEvent(). The information is stored in bits 8 through 15. When one of these keys is pressed, the corresponding bit is set. Note that not all keyboards distinguish between the right and left Control, Shift, and Option keys, in which case only the bits corresponding to shiftKey, optionKey, and controlKey are set.

The bit assignments for these keys are listed in Table 7-1.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Command key</td>
</tr>
<tr>
<td>9</td>
<td>Shift key</td>
</tr>
<tr>
<td>10</td>
<td>Caps Lock key</td>
</tr>
<tr>
<td>11</td>
<td>Option key</td>
</tr>
<tr>
<td>12</td>
<td>Control key</td>
</tr>
<tr>
<td>13</td>
<td>right Shift key</td>
</tr>
<tr>
<td>14</td>
<td>right Option key</td>
</tr>
<tr>
<td>15</td>
<td>right Control key</td>
</tr>
</tbody>
</table>
The state of these keys can be tested using the masks that are defined in the Events.h header file. The masks for the keys are as follows:

- cmdKey = 0x0100
- shiftKey = 0x0200
- alphaLock = 0x0400
- optionKey = 0x0800
- controlKey = 0x1000
- rightShiftKey = 0x2000
- rightOptionKey = 0x4000
- rightControlKey = 0x8000

The following code tests to see if the Option key or Shift key was pressed at the time the event was posted:

```c
if ( WaitNextEvent( everyEvent, &event, 30, nil) )
{
    if ( event.modifiers & optionKey )
        // do option key action

    if ( event.modifiers & shiftKey )
        // do shift key action
}
```

It is important to note that the bits are set to represent the key states as they were at the time the event was posted, not at the time you actually test the modifiers field. This is useful for testing Shift key-mouse click combinations or Option key-drag combinations. If you want real-time status checking, use GetKeys().

**Related FAQ**

See also FAQ 7-9, "Why aren't keyDown events generated when I hit the Shift key or the Option key?"
How do I detect that the arrow keys have been hit?

The arrow keys return the following #define character codes:

- RIGHT_ARROW = 0x1D
- LEFT_ARROW = 0x1C
- UP_ARROW = 0x1E
- DOWN_ARROW = 0x1F

The following call is made when WaitNextEvent() gets a keyDown event. It tests to see if the character code in the message field equals the code for one of the arrow keys.

```c
/******************
void HandleKeyDown( EventRecord *eventPtr )
{
    char ch = 0;

    ch = eventPtr->message & charCodeMask;

    switch ( ch )
    {
        case RIGHT_ARROW:
            break;
        case LEFT_ARROW:
            break;
        case UP_ARROW:
            break;
        case DOWN_ARROW:
            break;
        default:
            break;
    }
}
```
Why aren't keyDown events generated when I hit the Shift key or the Option key?

The Shift, Option, Control, and Command keys are modifier keys. They do not generate keyDown events. Instead, their state can be detected by the modifier flag in the event passed back by WaitNextEvent().

Related FAQ

See also FAQ 7-7, “How can I tell when the Shift, Control, Command, or Option key is pressed?”

Why don't I receive the key-up event, even when I set the eventMask in WaitNextEvent() to every Event?

There is a system-level event mask — global variable SysEvtMask at location 0x0144 — that is used by the system to determine which events it should post to the event queue. The normal value for this mask is 0xFFEF, which tells the system to post all events except the keyUp event. To set SysEvtMask, use the call SetEventMask(). To enable the system to post keyUp events, call SetEventMask(0xFFFF). Be careful when setting the SysEvtMask — if you accidentally set it to mask out events, you may not be able to send any input to your machine until you reboot. Therefore, you should only change the SysEvtMask global variable to enable keyUp events. Also make sure you reset SysEvtMask back to what it was before you modified it.

Each pass through WaitNextEvent() resets SysEvtMask back to 0xFFEF. You will have to enable keyUp events before each call to WaitNextEvent().

The following example saves the current value of SysEvtMask and then modifies the mask to enable keyUp events:

```
//********************
Boolean gDone;

void EventLoop( void )
{
    EventRecord event:
```
short oldMask;
oldMask = *(short *)0x0144;
gDone = FALSE;
while ( gDone == FALSE )
{
    SetEventMask( everyEvent);
    if ( WaitNextEvent( everyEvent, &event, 30, nil) )
    {
        HandleEvent( &event );
    }
}
SetEventMask( oldMask);

void HandleEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
        case keyUp:
            // Do keyUp action
            break;

        default:
            break;
    }
}

How can I get the character code and virtual key code of a key pressed by the user?

The virtual key code represents the actual key pressed by the user. It is always the same for a specific physical key on a particular keyboard; it is not affected by any of the modifier keys that are pressed in conjunction with it. The character code represents a particular character. Its value is affected by the modifier keys.
When the user presses a key, it generates a keyDown event. Bits 0 through 7 define the character code. Bits 8 through 15 define the virtual key code. Masks for finding the two codes are defined in Events.h as follows:

```c
#define charCodeMask 0x000000FF
#define keyCodeMask 0x0000FF00
```

```c
//*****************
void HandleKeyDown( EventRecord *eventPtr )
{
    char char_code, virtual_code;

    char_code = eventPtr->message & charCodeMask;
    virtual_code = eventPtr->message & keyCodeMask;
}
```

**How do I capture keystrokes in all applications?**

Keystrokes input by the user can be captured through the use of a GNE filter function. A GNE filter function is called from getNextEvent() and waitNextEvent() immediately before the calls return to your application. It is passed a pointer to the event record that will be passed to your application. The GNE filter can examine the what and message fields to determine if the event is a key event. Since it has access to the actual event that will be passed to the application before the application receives it, the GNE filter can modify the event.

**Related FAQs**

- See also FAQ 7-30, "How do I create and install a GNE filter?"
- See also FAQ 30-21, "How can I examine key-down and mouse events in all applications?"

**Mouse Events**

This section answers FAQs about the mouse and handling mouse events. It covers the methods of detecting double-clicks and mouse-moved events. It also shows how to get the current position of the mouse and how to get the current state of the mouse button.
How can I tell if the mouse button is up or down?

mouseDown and mouseUp events are generated when the mouse button is pressed and released. If you want to get the state of the mouse button, either up or down, at the current time, call Button(). The following code will cause your program to wait until the mouse button is pressed:

```c
while ( !Button() );
```

Another function, StillDown(), checks to see if the mouse button has been continuously pressed since the last mouseDown event. WaitMouseUp() also checks to see if the mouse button has been continuously pressed since the last mouseDown event, but in addition, it removes the mouseUp event from the event queue.

How do I detect a double-click?

There is no double-click message. You have to measure the time of a mouse-down event yourself to determine if it occurred soon enough after a previous mouse-up event to constitute a double-click. The time of a mouse event in ticks is stored in the event.when field. Save this for each mouse-down event. Get the tick count for a mouse-up event and compare it to the tick count from the last mouse-down. The result you get when you subtract the mouse-down tick count from the mouse-up tick count should be less than the double-click threshold. GetDblTime() returns the maximum number of ticks allowed between a mouse-down event and a mouse-up event that constitutes a double-click. This value can be adjusted in the Mouse control panel.

The following code calculates a double-click event:

```c
//*************************************************************************
void HandleEvent( EventRecord *eventPtr )
{
    static long  lastMouseTick = 0;
    static Point lastMousePoint;
    long         doubleClickThreshold;

    switch ( eventPtr->what )
    {
        case mouseDown:
            doubleClickThreshold = GetDblTime();
            lastMouseTick = eventPtr->when;
            lastMousePoint = eventPtr->point;
```

```c
```
if ( ((eventPtr->when - lastMouseTick) <
    doubleClickThreshold) &&
    (abs(eventPtr->where.h -
    lastMousePoint.h) < 5) &&
    (abs(eventPtr->where.v -
    lastMousePoint.v) < 5) )
{
    // Do double click action
}
else
{
    // Do single click action
}
lastMousePoint = eventPtr->where;
brea;}

How can I get the current mouse coordinates?

All events pass back the current mouse coordinates in the where field of the event structure. After you receive an event from WaitNextEvent(), you can get the current position of the mouse from the point contained in event.where, as follows:

    EventRecord event;
    Point mousePoint;
    
    if ( WaitNextEvent( everyEvent, &event, 30, nil) )
    {
        mousePoint = event.where;
        HandleEvent( &event);
    }

If you want to get the current position of the mouse without getting an event, you can use the GetMouse() call, as follows:
Are the mouse coordinates returned in the where field of the event structure in local or global coordinates?

The where field of the event structure contains a point representing the mouse location in global coordinates. To get the coordinate in the window clicked on by the user, translate the coordinate into a local coordinate by calling GlobalToLocal().

How can I tell when the mouse moves if there are no mouse-moved events?

Under certain circumstances, you can determine if the mouse moved by examining an operating system event. A mouse-moved message will be sent whenever the mouse moves out of a region that is in the last parameter of WaitNextEvent(). This can be detected by testing for the value of mouseMovedMessage in the high byte of the message field of an operating system event. The high byte of the event's message field will be set to 0xFA.

To get continuous mouse-moved messages sent to your application, recalculate the region to be passed to WaitNextEvent() after each message is received. The new region should contain the current mouse position and should be small enough to generate a new mouse-moved message the next time the mouse moves.

The following code sets up a mouse region to be the GrayRgn minus the region of the window. Whenever the cursor enters the window rectangle, and consequently moves out of the defined mouse region, a mouse-moved event will be generated. The message field of an osEvt event is examined to see if the high byte is set to 0xFA.

```c
//***************
WindowPtr pWindow;

void EventLoop( void )
{
```
EventRecord event;
RgnHandle mouseRgn;

mouseRgn = NewRgn();
DiffRgn( LMGetGrayRgn(), ((WindowPeek)pWindow)->structRgn, mouseRgn);

if (WaitNextEvent( everyEvent, &event, 30, mouseRgn))
{
    switch (event.what)
    {
        case osEvt:
            // check for suspend or resume event
            if (event.message & 0x01000000)
            {
                if (event.message & 0x00000001)
                    DoActiveFunction();
                else
                    DoDeactiveFunction();
            }

            // check for a mouse-moved event
            if ((event.message & 0xFF000000) == 0xFA000000)
                DoMouseMovedFunction();
            break;
    }
}

Related FAQs

- See also FAQ 3-4, “How do I get my program to change the cursor to another cursor when over a certain area in my window?”
- See also FAQ 3-9, “Will a hidden cursor report mouse-down and mouse-moved events?”
- See also FAQ 7-22, “How can I tell what type of OS event is posted?”
See also FAQ 7-24, “How can I tell when my application goes to the background and returns to the foreground?”

See also FAQ 7-31, “Why can’t my GNE filter detect OS events (such as when an application comes to the foreground)?”

**How can I tell what part of my window the user clicked in?**

When an application receives a mouse-down event, it must determine the on-screen location where the user-selection was made. The window that was clicked is passed back in the `FindWindow()` routine’s second parameter. The routine also returns a value indicating the part of that window in which the clicked point is located. The possible return values are as follows:

- `inDesk = 0;` // none of the following
- `inMenuBar = 1;` // in the menu bar
- `inSysWindow = 2;` // in desk accessory window
- `inContent = 3;` // anywhere in the content region except the size box if the window is active, or anywhere including the size box if the window is inactive
- `inDrag = 4;` // in the drag (title bar) region
- `inGrow = 5;` // in the size box (active window only)
- `inGoAway = 6;` // in the close box
- `inZoomIn = 7;` // in the zoom box (window in standard state)
- `inZoomOut = 8;` // in the zoom box (window in user state)

**Related Topic**

See also Chapter 33, “Windows.”
Processing Events

This section answers FAQs about general event handling. It answers how, when, and why activate, deactivate, resume, and suspend events are posted. It also discusses the event queue and shows how a program can send an event to itself.

Is there an event priority that determines the order in which events are returned by the Event Manager?

Yes, each type of event has a certain priority that determines the order in which events are returned by the Event Manager. The priority is as follows:

1. Activate events
2. Mouse-down, mouse-up, key-down, key-up, and disk-inserted events (in first-in, first-out order)
3. Auto-key events
4. Update events (in front-to-back order of the windows)
5. Operating system events (suspend, resume, mouse-moved)
6. High-level events
7. Null events

How can I tell how much time has elapsed between events?

Every event that gets retrieved by WaitNextEvent() and GetNextEvent() contains, in its event.where field, the time when the event was posted. This value represents the number of ticks since System startup.

You can also call TickCount() after WaitNextEvent() or GetNextEvent() to get the tick count when you read the event.
What is the purpose of the sleep parameter in 
WaitNextEvent()?

The sleep parameter in WaitNextEvent() specifies the length of time, in ticks, during which the application relinquishes the CPU if no events are pending for it. If 0 is passed in the sleep parameter, then the Event Manager will yield only a minimum amount of time to the background processes. If a nonzero value is passed in and no events are pending in the application’s event queue when WaitNextEvent() is called, the processor schedules other processes until an event becomes available or the specified sleep time expires. Then the Process Manager schedules your process to run at the next available chance. If, at this time, no events are pending for your application, the Process Manager sends your application a null event.

The frequency of any periodic tasks that your application may perform will determine the value you specify in the sleep parameter. The less frequent the task, the higher the value of sleep. Inside Macintosh states that a reasonable value for the sleep parameter is 60, which represents one second.

How can I tell what type of OS event is posted?

You can tell if an OS event is a mouse-moved event, suspend event, or resume event from the message field. If bit 24 and bit 0 are both set, then the event is a resume event. If bit 24 is set and bit 0 is not set, the event is a resume event. If the two high bytes are equal to 0xFA, then the event is a mouse-moved event. The following code examines the OS event:

```c
//******************************
switch ( eventPtr->what )
{
    case osEvt:
        if (eventPtr->message & 0x01000000 )
        {
            // suspend or resume
            if ( eventPtr->message & 0x00000001 )
            {
                // resume event
            }
            else
            {
                // mouse-moved event
            }
        }
```
// suspend event

// check for a mouse-moved event
if ((eventPtr->message & 0xFF000000)==0xFA000000)
{
    //mouse moved
    break;
}

Related FAQs

See also FAQ 7-17, "How can I tell when the mouse moves if there are no mouse-moved events?"

See also FAQ 7-24, "How can I tell when my application goes to the background and returns to the foreground?"

See also FAQ 7-31, "Why can't my GNE filter detect OS events (such as when an application comes to the foreground)?"

When my program receives an activate event, how can I tell if the window is activated or deactivated?

When a window should be activated or deactivated, an activateEvt is posted. The message field of this event contains a windowPtr if the window is being activated or deactivated. The first bit of the event’s modifiers flag is set if the window is becoming active. To determine if the window is activated or deactivated, test the modifiers field against the activeFlag mask as follows:

//*******************
void HandleEvent( EventRecord *eventPtr )
{
    WindowPtr   theWindow;
    Boolean    becomingActive;

    switch ( eventPtr->what )
{    
    case activateEvt:
        theWindow = (WindowPtr)eventPtr->message;
        becomingActive = (((eventPtr->modifiers & activeFlag) != 0));
        HandleActivate( theWindow, becomingActive);
        break;
    
}
**Why won't my application receive suspend or resume events?**

For an application to receive suspend and resume events, the `acceptSuspendResumeEvents` bit in the application’s ‘SIZE’ resource must be set.

Related Topic

- See also Chapter 20, “Processes: Process Manager.”

**How do I send my program an event?**

An application can send itself an event by calling the `PostEvent()` or `PPostEvent()` function, or by inserting an event in an event queue whose header can be retrieved by `GetEvQHdr()`.

Related FAQ

- See also FAQ 7-27, “Is it possible to post an event to the head of the event queue?”

**Is it possible to post an event to the head of the event queue?**

You can get the address of the event queue header by calling `GetEvQHdr()` . This returns the value 0x014A, which is the address of the global variable `EventQueue` . The event queue header is of type QHdr . The qHead field of the QHdr structure contains the address of the first queue element. The event queue is composed of elements of type EvQE1. You can create an EvQE1 element that represents an event and insert it at the front of the event queue.
**Why does my application get continuous update events?**

Make sure you are calling `BeginUpdate()` and `EndUpdate()` in response to an update event. `EndUpdate()` signals that you are finished updating the contents of the window pointed to in the update event.

**How do I delay my program for a time that will be consistent on all machines?**

Use the `Delay()` function. `Delay()` pauses the execution of your program for a specified number of ticks. This will be constant on all machines. You can get the number of ticks that have elapsed since the system started by calling `TickCount()`, as follows:

```c
long finalTick;

// pause for 2 seconds
Delay( 120, &finalTick);
```

**GNE Filters**

This section answers FAQs about GNE filters. A GNE filter provides special processing capabilities. It allows an application to intercept all events that are posted by the system, even those sent to other applications.

**How do I create and install a GNE filter?**

Creating and installing a GNE filter is not difficult on a Power Macintosh. The definition of a GNE filter function is as follows:

```pascal
pascal void myGNEFilter( EventRecord *theEvent, Boolean *bValue)
```
The filter function gets passed two parameters, a pointer to the event that will be
passed to the application through `WaitNextEvent()` or `GetNextEvent()`, and
the Boolean result.

To install the GNE filter, you must access the low-memory global variable at
address 0x29A that contains a pointer to the GNE filter. The universal headers
provide two functions, `LMGetGNEFilter()` and `LMSetGNEFilter()`, that
allow you to get and set this pointer. Before your filter function is installed, the
address of the current filter function must be retrieved with `LMGetGNEFilter()`
and stored. This value will be used later by the GNE filter function. Next, a
universal procedure pointer of your filter function is installed by
`LMSetGNEFilter()`.

The GNE filter can examine the event that is passed to it and perform some
appropriate processing, change the event and have the modified value passed to
the application, or intercept the event by changing it to a null event that is passed to
the application. When it has completed its processing, the filter function should call the
function that it replaced when it was installed. This is done by calling
`CallGetNextEventFilterProc()`, which takes three parameters: the UPP of
the old filter function, the event pointer, and the Boolean result value.

The following sample code shows an application installing a GNE filter. The GNE
filter beeps twice every time a mouseDown event occurs in any application. When
the application exits, the filter function is deinstalled. This routine was written
using the CodeWarrior compiler. The Symantec compiler has different names for
two routines used: `CallGetNextEventFilterProc()` becomes
`CallGNEFilterProc()`, and `NewGetNextEventFilterProc()` becomes
`NewGNEFilterProc()`.

```c
//********************
GNEFilterUPP oldGNE;
pascal void myGNEFilter( EventRecord *theEvent, Boolean *bVal);

main()
{
   DoProgramInit();

   oldGNE = LMGetGNEFilter();
   LMSetGNEFilter( NewGetNextEventFilterProc(myGNEFilter) );

   MainLoop();

   LMSetGNEFilter( oldGNE);
}
```
pascal void myGNEFilter( EventRecord *theEvent, Boolean *bVal)
{
    switch ( theEvent->what )
    {
        case mouseDown:
            SysBeep(10);
            SysBeep(10);
            break;
    }

    *bVal = true;
    CallGetNextEventFilterProc(oldGNE, theEvent, bVal);
}

Writing and installing a GNE filter for a 68K Macintosh is much more complicated and requires the use of inline assembly. The filter routine is defined as follows:

void myGNEFilter(void)

When it is called, A1 points to the event record and D0 contains the Boolean result. If the filter routine is going to access its own globals, it should set up the A4 register. After the event processing is completed, the previous GNE filter should be called. There are numerous examples online that show how to install a 68K GNE filter.

Deinstalling a filter function is dangerous. If another GNE filter function is installed after yours, it will most likely try to call your routine to complete the chaining of filter functions. If your function is no longer in memory, bad things will happen.

Related FAQs

See also FAQ 7-12, “How do I capture keystrokes in all applications?”

See also FAQ 30-21, “How can I examine key-down and mouse events in all applications?”

Why can’t my GNE filter detect OS events (such as when an application comes to the foreground)?

A GNE filter won’t see OS events. It sits below the Process Manager and consequently will see only those events that are generated by the Toolbox itself, rather than the Process Manager. An alternative approach would be to patch GetNextEvent() or WaitNextEvent().
Related FAQs

- See also FAQ 7-17, “How can I tell when the mouse moves if there are no mouse-moved events?”
- See also FAQ 7-22, “How can I tell what type of OS event is posted?”
- See also FAQ 7-24, “How can I tell when my application goes to the background and returns to the foreground?”

Topic-Related FAQs

- See also FAQ 3-9, “Will a hidden cursor report mouse-down and mouse-moved events?”
- See also FAQ 6-10, “Why can’t my modal dialog box detect disk-insert events?”
- See also FAQ 14-3, “How do I detect a double-click on a cell?”
- See also FAQ 23-16, “Why does my drawing trash the screen whenever I draw in response to an update event for a window that is not the frontmost window?”
- See also FAQ 30-21, “How can I examine key-down and mouse events in all applications?”
Applications in the Macintosh are able to play sounds and display beautiful graphics. In order to do this, they need to obtain, process, and save all types of data. This data is stored in files. Every application utilizes files, whether it is to retrieve a resource from its own resource fork or to read and write the data that it uses. The File Manager provides you with the functions to open, close, read, and modify your program’s files. As you explore this chapter, you’ll learn about the file system, volume reference numbers, directory IDs, file references, invisible files, path names, drive queue, and resource and data forks. You’ll also learn how to do the following:

- Recurse down through all the files and subfolders in a directory
- Delete one of a file’s data or resource forks without deleting the other
- Find the amount of free space on a volume
- Determine if there’s a floppy in the floppy drive, and if that floppy is locked
- Access the special System folders such as the Extensions, Control Panels, and Preferences folders
There are six subsections in this chapter:

- Definitions
- Referencing Files, Folders, and Volumes
- Disks
- Searching for Files, Folders, and Volumes
- Special System Folders
- Working with Files, Folders, and Volumes

Definitions

This section defines a number of terms that are used to reference files. These terms are used in almost all of the FAQs in this chapter.

What is an FSSpec?

An FSSpec, or file system specification, is a structure that is used to specify the name and location of a file or directory. The universal headers’ structure definition of the file system specification is as follows:

```c
struct FSSpec
{
    short vRefNum;
    long parID;
    Str63 name;
};
typedef struct FSSpec FSSpec;
typedef FSSpec *FSSpecPtr, **FSSpecHandle;
```

There are three components to an FSSpec structure. They are:

- `vRefNum` — the volume reference number
CHAPTER 8: Files: File Manager

- **parID** — the directory ID of the parent directory
- **name** — the name of the file or directory

To convert a file specification into a **FSSpec** structure, call `FSMakeFSSpec()`.

**What is the difference between the default directory and volume and the current directory and volume?**

The default directory and volume are maintained by the File Manager and are used in File Manager calls whenever you don't explicitly specify a directory. The current file and volume are maintained by the Standard File Package and are used with the Standard Dialogs.

An application maintains a default volume and default directory that are used by the File Manager routines whenever you don't explicitly specify a directory. The default volume is the volume containing the default directory. If you pass 0 as the volume reference number with routines that operate on a volume, the File Manager assumes that you want to perform the operation on the default volume. With routines that access files or directories, if you don't specify a directory and you pass a volume reference number of 0, the File Manager assumes that the file or directory is located in the default directory. Initially, the volume used to start up the application is set as the default volume and the folder containing your application is the default directory. These values can be modified by functions such as `PBHSetVol()`.

The current volume and directory can only be retrieved and set through the manipulation of low-memory global variables.

**Related FAQs**

- See also FAQ 8-10, “How can I get and set the default volume and folder?”
- See also FAQ 8-16, “Why don’t the current directory and volume used in the Standard File Package calls set the default directory and volume used in File Manager calls?”
- See also FAQ 9-8, “How do I set which folder should initially be shown in the Standard File Package calls?”
What are the creator ID and file type of a file?

An application's creator ID, otherwise known as signature, is a 4-byte field that uniquely identifies the application in the Finder's database. The application itself and documents created by it share the same signature. When you double-click on a document, the Finder looks in its database for the application with the same creator ID. When it finds one, it launches it by sending a `kAEOpenDocuments` Apple event. Any character (uppercase and lowercase are distinct), number, and symbol can be used in a creator ID. Apple reserves all signatures that are made up of four lowercase letters.

A file type is another 4-byte field that is assigned to a file. It is used to differentiate types of files and documents associated with an application.

Together, the creator ID and the file type are used as part of a 'BNDL' resource to, among other things, determine the icon that is displayed on the Desktop.

Both unique creator IDs and file types must be registered with Apple.

What is a working directory?

A working directory is a combined volume and directory specification used by the File Manager. A working directory reference number is an alias to a hard directory ID and a volume reference. The File Manager uses this reference number to look up the real volume reference and directory numbers in a system-wide table.

Apple developed the working directory to allow applications written for the old Macintosh file system (MFS) to work properly under the hierarchical file system (HFS). Since MFS didn't truly support directories, HFS needed to be able to encode both volume and directory information in the same place. This was done in `vRefNum`, creating the concept of a `WDRefNum`.

A volume reference number is a small negative word (such as `$FFFE`). A working directory is a large negative word (such as `$8033`).
What are resource and data forks of a file?

Every file consists of two parts: a data fork and a resource fork. The resource fork contains the resources of the file. The data fork contains everything else and can be in any format the creating application desires. Only resource data should be stored in the resource fork in order for the Resource Manager to understand its contents. A Mac file always contains both forks; however, either one or both can be empty. Each fork must be individually opened for access.

What is the drive queue?

The drive queue is a list of all disk drives connected to the computer that the File Manager creates at System startup time and then maintains. The File Manager maintains four flag bytes preceding each drive queue element. From these flags, you can get information about whether or not any of the following conditions exist:

- The volume is locked
- A disk is currently in the drive
- The drive is an ejectable drive
- The disk was ejected within the last 1.5 seconds

Referencing Files, Folders, and Volumes

This section answers FAQs about accessing files, folders, and volumes. For example, if you have 10 different files named “My Data,” how does your program know which one you mean?
You can refer to a specific file by using combinations of different types of information, including the file’s name, partial path, and full path, as well as the reference numbers of its folder and volume. This section shows the different ways to use this information.

**How do I specify a path name, and how do I include and separate the volume name, directory names, and file name?**

The colon character (‘:’) is used by the File Manager to separate the volume name, directory names, and file name. A full path name begins with the root directory (or volume) name, includes all directories that lead to the file, and ends with the file name, as follows:

```
VolumeName:Directory1:Directory2:FileName
```

A partial path begins with a colon (‘:) in a directory other than the root directory, as follows:

```
:Directory2:FileName
```

The partial directory is usually used in conjunction with either a volume reference ID, working directory reference number, or directory ID that acts as the relative starting point.

The parent directory is denoted by two colons (‘::’), as follows:

```
::Directory2 is equivalent to Directory1
```

**Related FAQ**

See also FAQ 8-8, “What is the maximum length of file, directory, and volume names?”

**What is the maximum length of file, directory, and volume names?**

The maximum size of a volume name is 27 characters. The maximum size of a file or directory name is 31 characters.
How do I get the full path of a file referenced by a vRefNum, parent directory ID, and file name?

Generating a full path name from a vRefNum, parent directory ID, and file name can be done by making successive calls to PBGetCatInfo() to travel up the directory tree, as shown in the following example. Each call to PBGetCatInfo() gets the name and directory ID of the parent directory of the given directory. This is continued until you reach the root directory that has a directory ID of 2. Note that you insert the separator colon character (':') between the volume name, all directory names, and the file name.

```c
//********************************************************
char *PathNameFromDirId( long dirId, short vRefNum, char *fullPath )
{
    CInfoPBRec myCPB;
    Str255 directoryName;
    OSERR err;

    *fullPath = 0;
    myCPB.dirInfo.ioNamePtr = directoryName;
    myCPB.dirInfo.ioDrParID = dirId;

    do
    {
        myCPB.dirInfo.ioVRefNum = vRefNum;
        myCPB.dirInfo.ioFDirIndex = -1;
        myCPB.dirInfo.ioDrDirID = myCPB.dirInfo.ioDrParID;
        err = PBGetCatInfo(&myCPB, false);

        pStrcat(directoryName, "\p:"
        pStrcat(directoryName, fullPath);
        pStrcpy(fullPath, directoryName);
    }
    ```
Macworld Mac Programming FAQs

```
while ( myCPB.dirInfo.ioDrDirID != fsRtDirID )
    return(fullPath);
```

How can I get and set the default volume and folder?

PBHSetVol(), PBSetVol(), and HSetVol() allow you to set the default volume and default directory. PBHGetVol(), PBGetVol(), and HGetVol() allow you to determine the default volume and default directory.

Related FAQs

- See also FAQ 8-2, “What is the difference between the default directory and volume and the current directory and volume?”
- See also FAQ 8-16, “Why don’t the current directory and volume used in the Standard File Package calls set the default directory and volume used in File Manager calls?”
- See also FAQ 9-8, “How do I set which folder should initially be shown in the Standard File Package calls?”
- See also FAQ 9-9, “How do I get the current folder and volume that are initially displayed by the Standard File Package?”

How do I find the DirID and VolRefNum of a file located in the same folder as my application?

To access a file located in the same folder as your application, you can use FSSpec with the vRefNum and parID field set to 0 and the name field set to the name of the file. Values of 0 passed in vRefNum and parID tell the File Manager to look in the current volume and directory. When your application starts up, they point to the volume and directory of your application. As long as you did not change them with SetVol() or PBHSetVol(), they will remain unchanged.
How can I get the directory ID of a folder from the full path name and from the parent folder's ID and directory name?

If you have a full path of a folder, and the file name is less than 255 characters, you can call PBGetCatlnfo() to get the directory ID. Set ioFDirIndex to zero, ioDrDirID to zero, and ioNamePtr to the path name. The directory is passed back in the ioDrDirID field. Call PBHGetVlnfo() to find its vRefNum. The following sample routine shows how to do this:

```c
//***************************
long GetDirIDFromPath( Str255 fullPath)
{
    CinfoPBRec myCPB;
    OSerr err;
    myCPB.dirInfo.ioNamePtr = fullPath;
    myCPB.dirInfo.ioVRefNum = 0;
    myCPB.dirInfo.ioFDirIndex = 0;
    myCPB.dirInfo.ioDrDirID = 0;
    err = PBGetCatlnfo(&myCPB, fullPath, false);
    return( myCPB.dirInfo.ioDrDirID );
}
```

If the full path name is longer than 255 characters, you will first have to break it up into pieces with less than 255 characters. Then you can employ the methods discussed previously to get the volume reference number and directory ID of a parent directory. With this data, you can use the following method to get the final directory information.

If you have a partial path, parent directory ID, and volume reference number, you can get the directory ID again by calling PBGetCatInfo(). Set iFDirIndex to zero, ioNamePtr to the path name, ioDrDirID to the parent directory ID, and ioVRefNum to the volume reference number. The directory is passed back in the ioDrDirID field. The following sample routine shows how to do this:

```c
//***************************
long GetDirIDFromPartialPath( short vRefNum,
                                long parID,
                                Str255 folderName)
{
```
CInfoPBRec myCPB;
OSErr err;

myCPB.dirInfo.ioNamePtr = folderName;
myCPB.dirInfo.ioVRefNum = vRefNum;
myCPB.dirInfo.ioFDirIndex = 0;
myCPB.dirInfo.ioDrDirID = parId;
err = PBGetCatInfo(&myCPB, false);

return( myCPB.dirInfo.ioDrDirID );
}

How do I find the name of a folder from dirID and vRefNum?

Call PBGetCatInfo(). Set ioNamePtr to point to an empty string, ioFDirIndex to -1, and ioDrDirID and ioVRefNum to the directory ID and volume reference number of the folder. Setting ioFDirIndex to -1 forces PBGetCatInfo() to get information about the vRefNum/dirID folder instead of the file and/or folder specified by vRefNum, parID and name. The folder name is returned in the ioNamePtr.

The following function retrieves the name of the folder identified by vRefNum and dirID in the strFolderName parameter:

//************************
void GetFolderName( Str63 strFolderName, short vRefNum, long dirId )
{
CInfoPBRec myCPB;
OSErr err;

strFolderName[0] = 0;
myCPB.dirInfo.ioNamePtr = strFolderName;
myCPB.dirInfo.ioDrDirID = dirId;
myCPB.dirInfo.ioVRefNum = vRefNum;
myCPB.dirInfo.ioFDirIndex = -1;

err = PBGetCatInfo(&myCPB, false);
}
**How do I get the vRefNum of the startup volume?**

The current File Manager always assigns the startup volume a vRefNum of -1. That's because the boot volume is mounted first, and volume reference numbers are allocated starting with -1. This is not something that is guaranteed to remain true in the future. The safe method of getting the startup volume is to use `FindFolder()` to find the System Folder and then use the vRefNum returned.

**Can WDRefNum and vRefNums be used interchangeably, or are working directories obsolete?**

You can use a working directory reference number whenever a volume reference number is needed. As a general rule, you should avoid using working directories when possible. When a function does return a working directory to your program, translate it immediately to a volume reference number and directory ID.

If both a directory ID and a working directory reference number are specified, the volume referenced by the working directory is used and the directory specified by the directory ID is used.

Related FAQ

See also FAQ 8-4, “What is a working directory?”

**Why don’t the current directory and volume used in the Standard File Package calls set the default directory and volume used in File Manager calls?**

The Standard File Package and the File Manager use different variables to store the current volume and directory.

Related FAQs

See also FAQ 8-2, “What is the difference between the default directory and volume and the current directory and volume?”
How do I access the top directory of a volume and on the Desktop?

The directory ID of the root directory of all volumes is 2.

The items shown at the Desktop level are the mounted volumes and all items inside the invisible directory named Desktop Folder at the root of the volume. To get all of the mounted volumes, call PBHGetVInfo(). To find all of the files on the Desktop, call PBGetCatInfo() to enumerate the items in the Desktop Folder directory.

Related Topic

See also Chapter 10, "Finder and Desktop."

How do I get the directory and volume of my application file?

To get the volume and directory of your application, call HGetVol() or PBHGetVol() when your application starts up. HGetVol() and PBHGetVol() return the default volume and directory, which are initially the ones in which the application is located.

The following function returns the current default volume and directory:

///**********************
CHAPTER 8: Files: File Manager

```c
void GetAppVolumeAndFolder( short *myVRefNum, long *myDirID )
{
    Str255 volName;
    HGetVol(volName, myVRefNum, myDirID);
}
```

**How do I convert a working directory to a volume reference number and directory ID?**

You can convert a working directory into a volume reference number and directory ID by calling GetWDInfo() or PBGetWDInfo().

**Searching for Files, Folders, and Volumes**

This section answers FAQs about searching for file-related information. It covers how to get all of the files and folders in a folder, how to get a list of all of the volumes that are currently mounted, and how to search for files and folders that match a set of criteria.

**How do I find invisible files and folders in a directory?**

Use PBGetCatInfo() to scan the directory or directories that you want to test. Invisible files have the fInvisible bit set in the fdFlags field of their FileInfo structure. Invisible directories have the fInvisible bit set in the frFlags field of their FileInfo structure. Both of these structures can be retrieved from the CInfoPBRec structure used by PBGetCatInfo(). To verify that these bits are set, test the fields against the fInvisible mask.

The following code accepts a pointer to a CInfoPBPtr structure from a PBGetCatInfo() call and tests to see if the file is invisible:

```c
Boolean IsFileInvisible( CInfoPBPtr pCPB )
{
    if ( pCPB->hFileInfo.ioFlFndrInfo.fdFlags & fInvisible )
        return ( true );
```
return false;

The following code accepts a pointer to a CInfoPBRec structure from a PBGetCatInfo() call and tests to see if the folder is invisible:

```c
Boolean IsFolderInvisible( CInfoPBPtr pCPB)
{
    if ( pCPB->dirInfo.ioDrUsrWds.frFlags & fInvisible )
        return true;
    return false;
}
```

Related FAQ

See also FAQ 8-34, “How do I create an invisible file?”

**How can I get all of the files in a directory?**

PBHGetFinfo() can be used to get all of the files in a directory. Subdirectories are not returned. If you also want to get the subdirectories in a directory, use PBGetCatInfo().

Related FAQ

See also FAQ 8-22, “How can I recurse through all files and folders in a directory?”

**How can I recurse through all files and folders in a directory?**

To recurse through a directory on an HFS volume, use PBGetCatInfo(). You pass, into the ioVRefNum and ioDirID field of the CInfoPBRec parameter, the volume reference number and directory ID you want to search. To start the search you pass 1 into the IOFDirIndex field. This index is used to reference a specific file in the specified folder. To get more files, keep incrementing this index until PBGetCatInfo() returns an error.
To recurse through an entire volume, set the ioDirID field to the directory ID of the volume's root directory, which is always 2 (the File Manager constant fsRtDirID). You can determine if PBGetCatInfo() returned a file or a directory by examining the ioFlAttrib field.

The following example recurses through all of the files and directories on the volume nVRefNum:

```
CInfoPBRec gMyCPB;

//****************
void TraverseDir( short nVRefNum )
{
  Str255 gItemName;

  gMyCPB.dirlnfo.ioCompletion = NULL;
  gMyCPB.dirlnfo.ioNamePtr = gItemName;
  gMyCPB.dirlnfo.ioVRefNum = nVRefNum;

  SearchDir( fsRtDirID );
}

//****************
static void SearchDir ( long dirIDToSearch )
{
  short index = 1;
  OSErr gErr;

  do
  {
    gMyCPB.hFileInfo.ioFDirIndex = index;
    gMyCPB.hFileInfo.ioDirID = dirIDToSearch;

    gErr = PBGetCatInfo(&gMyCPB,false);
    if (gErr == noErr)
    {
      /* directory */
      if ( (gMyCPB.hFileInfo.ioFlAttrib & ioDirMask) != 0 )
      {
        ...
      }
      else
      {
        /* file */
        ...
      }
    }
  }
  while (gErr != noErr);
}
```
How can I find all mounted volumes?

All of the mounted volumes can be retrieved by calling PBHGetVInfo() with a ioVolIndex field value of 1 and incrementing it until an error is returned, as follows:

```c
void GetMountedVolumes( void )
{
    Str255 volName;
    short nVolRefNum;
    short vindex;
   OSErr nErr = noErr;

    for (vindex = 1; nErr == noErr; vindex++)
    {
        nErr = GetIndVolume (vindex, (char *)volName, &nVolRefNum);
        if (nErr == noErr)
            WriteStuff(volName, 0);
    }
}
```
static OSerr GetIndVolume (short whichVol, char *volName, 
    short *volRefNum)
{
    HParamBlockRec HPBlock;
    OSerr error;

    HPBlock.volumeParam.ioNamePtr = (unsigned char *)volName;
    HPBlock.volumeParam.ioVRefNum = 0;
    HPBlock.volumeParam.ioVolIndex = whichVol;

    error = PBHGetVInfo(&HPBlock, false);
    if(error == noErr)
        *volRefNum = HPBlock.volumeParam.ioVRefNum;
    return (error);
}

How do I search a volume for files that match a set of search criteria?

Use PBCatSearch() to search a volume for files matching a certain criteria. It returns a list of FSSpec records containing the matching files or directories.

PBCatSearch is not available on all volumes or under all versions of the File Manager. Use Gestalt to see if PBCatSearch() is supported by your version of the File Manager. To determine if a volume supports PBCatSearch(), call PBHGetVolParms() and test the results against the bHasCatSearch bit. The following sample does this for the volume identified by the volume reference number passed in as the parameter, nVRefNum:

//***************
#define BTstQ(arg, bitnbr) (arg & (1 << bitnbr))

Boolean PBCatSearch_Avail ( short nVRefNum )
{
    long gesResponse;
    GetVolParmsInfoBuffer GVPBuffer;
    HParamBlockRec HPBlock;

    if ( !(!Gestalt(gestaltFSAttr, &gesResponse) 
        && BTstQ(gesResponse, gestaltFullExtFSDispatching)) )
        return FALSE;
Macworld Mac Programming FAQs

HPBlock.ioParam.ioCompletion = nil;
HPBlock.ioParam.ioNamePtr = nil;
HPBlock.ioParam.ioVRefNum = nVRefNum;
HPBlock.ioParam.ioBuffer = (char *)&GVPBuffer;
HPBlock.ioParam.ioReqCount = sizeof(GVPBuffer);
if (PBHGetVolParms(&HPBlock, FALSE) != noErr)
    return FALSE;
if BTstQ(GVPBuffer.vMAttrib, bHasCatSearch)
    return TRUE;
else
    return FALSE;
}

PBCatSearch() can search for files that match a number of different types of search criteria. These include full and partial names, size ranges, date ranges, and Finder attributes. To search for files and/or directories that do not match the search criteria, include fssSBNegate in the ioSearchBits field.

PBCatSearch() can be used to collect a list of FSSpec records to all items on a volume by setting ioSearchBits in the parameter block to 0.

The following example searches the volume identified by vRefNum for all files that contain the string “FAQ”:

    //***************
    void Do_CatSearch( short vRefNum )
    {
        OSerr gErr;
        HParamBlockRec gPb;
        FSSpec gTheResults[kMaxMatches];
        CInfoPBRec gSpec1, gSpec2;
        // search criteria, part 2
        char gBuffer[kSearchBufferSize];
        // search cache
        Str255 gFileName = "\pFAQ";
        short matchCount;
        long dirID;
gPb.csParam.ioCompletion = nil;
// no volume name; use vRefNum

// no volume name; use vRefNum

// points to results buffer

// number of matches

// search partial name

// search partial name

// point to string to find

// clear bit 4 to ask for files

// set to nil

// set mask for bit 4

do {

gErr = PBCatSearch(&gPb.csParam, FALSE);

if ( ((gErr == noErr) || (gErr == eofErr)) &&

   (gPb.csParam.ioActMatchCount > 0) )
   {

for ( matchCount = 0; matchCount <
gPb.csParam.ioActMatchCount; matchCount++)
{
    // do action on found matches
    WriteStuff(gTheResults[matchCount].name, 0);
}
}
while ( gErr != eofErr );
}

If PBCatSearch() is not available, you can use PBGetCatlnfo() and check
your search criteria for each file and directory.

Special System Folders

This section answers FAQs about the special folders in the file system, such as the
System folder, the Preferences folder, and even the Trash.

How do I access the System (Blessed) folder?

To access the System folder, call FindFolder() to get the volume reference
number and directory ID of the System folder. If FindFolder() is not available,
use SysEnvirons(), as follows:

//**************************************************************************************
OSErr FindSysFolder(short *foundVRefNum, long *foundDirID)
{
    long gesResponse;
    SysEnvRec envRec;
    WDPBRec myWDPB;
    unsigned char volName[34];
    OSErr err;

    *foundVRefNum = 0;
    *foundDirID = 0;
    if ( !Gestalt(gestaltFindFolderAttr, &gesResponse) &&
        BTstQ(gesResponse, gestaltFindFolderPresent) )
    {
CHAPTER 8: Files: File Manager

/* Does Folder Manager exist? */
err = FindFolder (kOnSystemDisk, kSystemFolderType,
kDontCreateFolder, foundVRefNum, foundDirID);
}
else
{
    /* use SysEnviron */
    if (!(err = SysEnviron (curSysEnvVers, &envRec)))
    {
        myWDPB.ioVRefNum = envRec.sysVRefNum;
        volName[0] = '\000';
        myWDPB.ioNamePtr = volName;
        myWDPB.ioWDIndex = 0;
        myWDPB.ioWDPRecID = 0;
        if (!(err = PBGetWDInfo (&myWDPB, 0)))
        {
            *foundVRefNum = myWDPB.ioWDVRefNum;
            *foundDirID = myWDPB.ioWDDirID;
        }
    }
    return (err);
}

Where should my application store temporary files?

There is an invisible folder called "Temporary Items" at the root level of the volume that is meant to hold temporary files created by an application. It can be located with FindFolder(), which returns its volume reference number and directory ID. However, when you store temporary files here, make sure the names will be unique so they won't conflict with other temporary files from other applications, and remove all temporary files when your application closes.

The following example stores temporary files:

//************************
OSErr GetTemporaryFolder(short *foundVRefNum, long *foundDirID)
{
    OSErr err;
*foundVRefNum = 0;
*foundDirID = 0;

err = FindFolder (kOnSystemDisk, kTemporaryFolderType,
                 kDontCreateFolder, foundVRefNum, foundDirID);

return (err);

How do I find the special folders such as the System folder and the Preferences folder?

FindFolder() passes back the volume reference number and directory ID of the special folder specified by the following constants ('CONST'):

- kAppleMenuFolderType = 'amnu'; {Apple Menu Items}
- kControlPanelFolderType = 'ctrl'; {Control panels}
- kDesktopFolderType = 'desk'; {Desktop folder}
- kExtensionFolderType = 'extn'; {Extensions}
- kFontsFolderType = 'font'; {Fonts folder}
- kPreferencesFolderType = 'pref'; {Preferences}
- kPrintMonitorDocsFolderType = 'prnt'; {PrintMonitor documents}
- kStartupFolderType = 'strt'; {Startup items}
- kSystemFolderType = 'macs'; {System Folder}
- kTemporaryFolderType = 'temp'; {Temporary items}
- kTrashFolderType = 'trsh'; {single-user Trash}
- kWhereToEmptyTrashFolderType = 'empt'; {shared Trash on a network}
To search on the startup disk, pass the constant kOnSystemDisk in the vRefNum parameter. The File Manager can automatically create the desired directory if it does not exist if you pass the constant kCreateFolder in the third parameter. To prevent it from creating the folder, pass in kDontCreateFolder.

Call Gestalt() to test if FindFolder() is available. If FindFolder isn't available, call SysEnviron to find the System folder's working directory reference number and call PBGetWDInfo or GetWDInfo to convert that number to a true vRefNum and dirID, as follows:

```c
//*******************************
OSErr FindSysFolder()
{
    short foundVRefNum;
    long foundDirID;
    long gesResponse;
    OSErr err;

    foundVRefNum = 0;
    foundDirID = 0;
    if (!Gestalt (gestaltFindFolderAttr, &gesResponse) &&
        BTstQ (gesResponse, gestaltFindFolderPresent))
    {
        /* Does Folder Manager exist? */
        err = FindFolder (kOnSystemDisk, kSystemFolderType,
                          kDontCreateFolder, &foundVRefNum,
                          &foundDirID);
    }
}
```

Working with Files, Folders, and Volumes

This section answers FAQs on the use of the File Manager. It describes the differences between resource forks and data forks and the methods required to access them. It covers operations such as opening, reading from, and writing to a file. It also answers FAQs on how to delete only a single fork of a file as well as how to create an invisible file.
How do I read data from and write data to a file?

To read and write to a file, you must get a valid file reference number to indicate to the reading and writing routines which file they will be accessing. If the file does not exist, it must be created. If you are only interested in creating a resource file, you can call one of the resource-specific Create routines — such as FspCreateResFile() — to create a resource fork and the resource map in it. If you want to store data in the data fork of a file, you can call one of the routines that will create both forks, such as FSpCreate(). Note that FSpCreate() creates an empty resource fork. Before you open a resource fork of a file created with this call, you will have to create a resource map in the file by calling FSpCreateResFile() or one of the other resource-specific Create routines.

Once the file is created, you need to open the fork you want to access before you can read or write to it. You can open the date fork by calling FSpOpenDF(). You can open the resource fork by calling FSpOpenRF(). Both of these routines return a file reference number. Note that you cannot use the file reference number returned by FSpOpenDF() to access the resource fork or vice versa.

Now you are ready to read and write to the file. The point in the file that you read from and write to is determined by the location of the file mark. You can get the file mark's current position in the file by calling GetFPos(). You can set the position of the file mark by calling SetFPos(). The position of the file mark is given in terms of the number of characters it follows. When the file is opened, the file mark initially points to the beginning of the file (file position = 0). You can determine the current end of the file, which will be your limit for positioning the file mark, by calling GetEOF().

To read data from a data file, you can call FSRead() or PBRead(). To write data to the data file, you can call FSWrite() or PBWrite().

Reading from and writing to a resource fork is handled a bit differently. All of the resource data is maintained in the resource map. To write a resource to a resource fork, you must call AddResource(). AddResource() adds the handle, resource type, resource ID, and resource name to the resource file's resource map in memory. To force the new data to be written to the resource fork on disk, call WriteResource(). To read data from a resource fork, you can call GetResource(), which will return a handle to the resource of the specified type and resource ID.

After you have completed your read and write operations, you must close the file. This can be done with FSClose() or PBClose().
Can I interchange the file reference numbers that I get back from opening a data fork of a file and those that I get from opening a resource fork of a file?

No. A file’s resource fork file reference number is different from its data fork file reference number.

How can I tell if a file is open, and if it is open, how can I tell which fork of the file is open?

PBGetCatInfo(), PBHGetFinfo(), and PBGetFinfo() can all be used to determine whether or not a file is open and if so, which forks are open. To select the desired file, fill in the ioVRefNum, ioFDiридIndex, ioDirID, and ioNamePtr fields for PBGetCatInfo() and PBHGetFinfo() and the ioVRefNum, ioFDiридIndex, and ioNamePtr fields for PBGetFinfo(). After calling the function, check the value passed back in the ioF1Attrib. If bit 2 is set, then the resource fork is open. If bit 3 is set, then the data fork is open.

The following example shows how to check for an open file and its open forks:

```c
//*******************************************
OSErr IsFileOpen(short vRefNum, long dirId, Str255 strFileName )
{
    HParamBlockRec    HPBlock;
    OSErr nErr;
```
How do I delete only the resource fork or the data fork of a file?

To delete only one fork, open the file fork with the File Manager, set the fork’s EOF to zero, and then close the file and update the volume, as follows:

```c
void TruncateResFork( FSSpec fileSpec)
{
    OSErr err;
    short fRefNum;

    err = FSpOpenRF(&fileSpec, fsRdWrPerm, &fRefNum);
    err = SetEOF(fRefNum, 0);
}```
How do I delete a folder?

You can delete a folder with the same calls that you use to delete a file — PBHDelete(), PBDelete(), FSDelete(), and Hdelete(). The directory must be empty of all files. To recursively delete all files and subdirectories in a folder, call PBGetCatInfo() to recurse through all files and delete them. Remember, you cannot delete a subdirectory when PBGetCatInfo() initially encounters it. You must recurse into it, delete all of the files and subdirectories in it, and then delete it.

How do I alter a file without automatically changing the time stamp?

Basically, you make your modifications and save the file, which will change the date and time of the last modification, and then you restore the date and time back to what it was before the modification. Before you open the file, call PBHGetFInfo() and get the last modification date and time from the ioFlMdDat field in the HParamBlockRec union. Make the modifications you want to the file. Then reset the last modification date and time by calling PBHSetFInfo() with the original time stored in the ioFlMdDat field in the HParamBlockRec union.

The following example shows how to restore a file's date and time:

```c
void ModWithOldDate( short vRefNum;
        long dirId;
        Str255 strFileName )
{
    HParamBlockRec HPBlock;
    OSErr nErr;
    unsigned long originalDate;

    HPBlock.fileParam.ioCompletion = NULL;
    HPBlock.fileParam.ioNamePtr = strFileName;
    HPBlock.fileParam.ioVRefNum = vRefNum;
    HPBlock.fileParam.ioDirID = dirId;
    HPBlock.fileParam.ioFVersNum = 0;
```

Macworld Mac Programming FAQs

```c
HPBlock.fileParam.ioFDirIndex = 0;
nErr = PBHGetFInfo( &HPBlock, FALSE);

// get the current last modification date
originalDate = HPBlock.fileParam.ioFlMdDat;

/**
// Make your file modifications here
/**

HPBlock.fileParam.ioCompletion = NULL;
HPBlock.fileParam.ioNamePtr = strFileName;
HPBlock.fileParam.ioVRefNum = vRefNum;
HPBlock.fileParam.ioDirID = dirId;
HPBlock.fileParam.ioFVersNum = 0;
HPBlock.fileParam.ioFDirlndex = 0;
HPBlock.fileParam.ioFlMdDat = originalDate;
nErr = PBHSetFinfo( &HPBlock, FALSE);
}

Related Topic

See also Chapter 10, “Finder and Desktop.”

How do I create an invisible file?

To make a file invisible, get the file information with PBHGetFInfo(), set the invisible bit in the ioFlFdInfo.fdFlags field, and then reset the file information with PBHSetFInfo().

The following example creates an invisible file:

```c
/****************************
void CreateInvisibleFile( long dirId, short vRefNum,
                        Str255 strFileName )
{
    HParamBlockRec       HPBlock;
    OSerr                nErr;

    HPBlock.fileParam.ioCompletion = NULL;
    HPBlock.fileParam.ioNamePtr = strFileName;
    HPBlock.fileParam.ioVRefNum = vRefNum;
```
CHAPTER 8: Files: File Manager

HPBlock.fileParam.ioFVersNum = 0;
HPBlock.fileParam.ioDirID = dirId;

nErr = PBHCreate(&HPBlock, FALSE);

// get file info
HPBlock.fileParam.ioCompletion = NULL;
HPBlock.fileParam.ioNamePtr = strFileName;
HPBlock.fileParam.ioVRefNum = vRefNum;
HPBlock.fileParam.ioDirID = dirId;
HPBlock.fileParam.ioFVersNum = 0;
HPBlock.fileParam.ioFDirIndex = 0;

nErr = PBHGetFinfo( &HPBlock, FALSE);

// set file to be invisible
HPBlock.fileParam.ioFlFndrinfo.fdFlags |= fInvisible;

// set file info
HPBlock.fileParam.ioCompletion = NULL;
HPBlock.fileParam.ioVRefNum = vRefNum;
HPBlock.fileParam.ioFVersNum = 0;
HPBlock.fileParam.ioFDirIndex = 0;
HPBlock.fileParam.ioNamePtr = strFileName;
HPBlock.fileParam.ioDirID = dirId;

nErr = PBHSetFinfo( &HPBlock, FALSE);

Related FAQs

☐ See also FAQ 8-20, “How do I find invisible files and folders in a directory?”

☐ See also FAQ 10-10, “Why is my file displaying the standard icon and not my bundled icon?”
Why can’t I create a file smaller than 4K on my hard drive, and why does the size of the same file increase when I move it to a larger drive?

The File Manager always allocates disk space to a file in units called “allocation blocks,” which are groups of consecutive 512-byte chunks called “logical blocks.” A maximum of 65,535 of these allocation blocks can be accessed by the File Manager. To determine the minimum size of a file on a particular volume, divide the hard disk size by 65,535 and round up to the nearest 512 bytes. For example, a drive of about 32 MB can support allocation blocks of the minimum size, a single logical block, or 512 bytes. Drives ranging from about 32 MB to 64 MB can support minimum allocation blocks of 1024 bytes.

How do I open files whose names start with a period like driver names do?

During an Open call, if the file name starts with a period (‘.’), the Open code calls the Device Manager to open a driver, instead of the File Manager. This is because the Open trap is shared between the Device Manager and the File Manager. When Open is called, it checks first to see whether you’re trying to open a driver. Driver names always start with a period. To open those files, use the Open Data Fork functions — FSpOpenDF, HOpenDF, and PBHOpenDF. These functions bypass the driver name check and go right to the File Manager.

How do I open a file in the current directory with FSOpen()?

To open a file in the current directory, just pass 0 as the volume reference number. This specifies the current default directory set by SetVol() or PBHSetVol().
Why do I get the compiler error, "Error: ‘ioNameptr’ is not a struct/union/class member," when I try to access the ioNameptr field of an HParamBlockRec structure?

You get this error because ‘ioNameptr’ is not a struct/union/class member of HParamBlockRec and HPBlock.ioNameptr is not a valid reference. HParamBlockRec is defined as a union of ten structures. These structures are the ones that contain the ioNameptr field. The following example shows the definition of HParamBlockRec as defined in <Files.h> along with a legal reference to ioNamePtr:

```c
union HParamBlockRec
{
    HIOParam ioParam;
    HFileParam fileParam;
    HVVolumeParam volumeParam;
    AccessParam accessParam;
    ObjParam objParam;
    CopyParam copyParam;
    WDPParam wcParam;
    FIDParam fidParam;
    CSParam csParam;
    ForeignPrivParam foreignPrivParam;
};

HParamBlockRec HPBlock;

HPBlock.ioParam.ioNamePtr = nil;
```

Related Topic

See also Chapter 4, “Development Environment and Language Issues.”
How can I get the Finder label menu color and label string of a file?

The Label menu in the Finder displays eight different possible labels that can be attached to a file. The first designation represents no label or color. The next seven represent a color and label string that can be customized through the Labels control panel. The user can assign a color to an icon on the Desktop (file, folder, or volume) by selecting the icon and then selecting the appropriate menu item from the Label menu (see Figure 8-1). This information is stored in bits 1 through 3 of fdFlags in a file’s FileInfo structure. The color and label string associated with this value can be retrieved by the GetLabel() function.

The following example returns the color and label string of the file specified by an FSSpec:

```c
//*****************
OSErr GetFilesColorAndLabel( FSSpec *myFSSpec,
    RGBColor *labelColor,
    Str255 labelString )
```
CHAPTER 8: Files: File Manager

FInfo fileInfo;
short labelNumber;

err = FSpGetFinfo( myFSSpec, &fileInfo);

if (err == noErr)
{
    labelNumber = ((fileInfo.fdFlags >> 1) & 0x0007);
    GetLabel(labelNumber, labelColor, labelString);
}

return(err);

Related FAQ

See also FAQ 13-4, “How can I draw an icon as it appears in the Finder in the different Finder states?”

Related Topic

See also Chapter 10, “Finder and Desktop.”

Disks

This section answers FAQs about disks. It covers how to determine the type of a disk, determine if a disk is locked, and test to see if there is a diskette in your machine’s diskette drive.

How can I tell if there is a diskette in my diskette drive?

Determining whether or not there is a diskette in the diskette drive requires you to check the data stored in the drive queue that is maintained by the File Manager. First, you need to get the right drive queue element, as shown in the following example. GetDrvQHd() returns the head element of the drive queue. Compare the d0Drive field with the drive number of the diskette drive. If they are not the same, move to the next queue element that is stored in the qLink field. Repeat this procedure until the proper queue element is found. There are four byte flags that
precede each queue element in the drive queue. If the second byte equals 0, then there is no diskette in the drive. If it is equal to 1 or 2, then there is a removable disk in the drive. If it is equal to 8 or 0x48, then a non-ejectable disk is in the drive.

```c
//**********************************************************************
Boolean IsDiskInDrive( short driveNumber)
{
    DrvQElPtr   elementPtr;
    QHdrPtr     queueHeader;
    Ptr         p;

    queueHeader = GetDrvQHdr();
    elementPtr = (DrvQElPtr)queueHeader->qHead;

    while (elementPtr != nil)
    {
        if (elementPtr->dODrive == driveNumber)
        {
            p = (Ptr)elementPtr;
            /* to get to byte with eject info */
            p -= 3;
            if (*p == 0)
                return false; /* no disk in drive */
            else
                return true;
        }
        elementPtr = (DrvQElPtr)elementPtr->qLink;
    }

    return false;
}
```

Related FAQ

See also FAQ 8-42, “How do I tell if a Macintosh volume is a floppy or hard disk?”

How can I tell if a disk is locked?

To test if a disk is locked, you need to test for both hardware and software locks. This is done by calling PBHGetVInfo() and examining the volume attributes
field, ioVAtrb, as shown in the following example. Bit 7 is set for a hardware lock, and bit 15 is set for a software lock. A CD-ROM and a locked diskette will not have a software lock, only a hardware lock, so bit 7 is set and bit 15 is clear.

```c
/**********************************************
#define HARDWARE_LOCK_MASK 128
#define SOFTWARE_LOCK_MASK 32768

Boolean IsVolumeLocked( short vRefNum )
{
    HParamBlockRec    HPBlock;
    Str255             volName;
    long               lSize;

    HPBlock.volumeParam.ioCompletion = NULL;
    HPBlock.volumeParam.ioNamePtr = volName;
    HPBlock.volumeParam.ioVRefNum = vRefNum;
    HPBlock.volumeParam.ioVolIndex = 0;

    if ( PBHGetVInfo(&HPBlock, false) != noErr )
        return FALSE;

    if ( HPBlock.volumeParam.ioVAtrb & HARDWARE_LOCK_MASK )
        return TRUE;

    if ( HPBlock.volumeParam.ioVAtrb & SOFTWARE_LOCK_MASK )
        return TRUE;

    return FALSE;
}
```

How do I tell if a Macintosh volume is a floppy or hard disk?

There are two pieces of information related to determining whether or not a volume is a floppy or a hard disk. The first is used to determine if the drive is an ejectable drive. The second is used to determine if the size of the drive is consistent with being a floppy.
Determining whether or not a drive is an ejectable drive requires you to check the data stored in the drive queue that is maintained by the File Manager. First, you need to get the right drive queue element. `GetDrvQHdr()` returns the head element of the drive queue. Compare the `dQDrive` field with the drive number that you are looking for (you can get the drive number from `PBHGetVInfo()`). If they are not the same, move to the next queue element that is stored in the `qLink` field. Repeat this procedure until the proper queue element is found. There are four byte flags that precede each drive queue element in the drive queue. If the second byte equals 8 or 0x48, then the disk is not ejectable.

The following example tests to see if the volume `vRefNum` is ejectable:

```c
//****************
static Boolean IsEjectable( short vRefNum)
{
    DrvQElPtr elementPtr;
    QHdrPtr queueHeader;
    Ptr p;
    Str255 volName;
    short driveNumber;
    HParamBlockRec HPBlock;
    HPBlock.volumeParam.ioCompletion = NULL;
    HPBlock.volumeParam.ioNamePtr = volName;
    HPBlock.volumeParam.ioVRefNum = vRefNum;
    HPBlock.volumeParam.ioVolIndex = 0;
    if ( PBHGetVInfo(&HPBlock, FALSE) != noErr )
        return FALSE;
    driveNumber = HPBlock.volumeParam.ioVDrvInfo;
    queueHeader = GetDrvQHdr();
    elementPtr = (DrvQElPtr)queueHeader->qHead;
    /* find the appropriate drive # */
    while (elementPtr != nil)
    {
        if (elementPtr->dQDrive == driveNumber)
        {
            p = (Ptr)elementPtr;
```
/* to get to the byte with eject info */
    p -= 3;
/* non ejectable disk in drive */
    if (*p == 8 || *p == 0x48)
        return false;
    else
        return true;
}
elementPtr = (DrvQE1Ptr)elementPtr->qLink;
}

return false;

To determine if the size of the drive is consistent with being a floppy, call
PBHGetVInfo() and multiply the ioVNmALBlks and ioVAlBlkSiz fields of
the HPBlock.volumeParam structure, as shown in the following example. If the
size is less than or equal to 1440K, then the volume is a floppy.

/*****************
Boolean IsVolumeADiskette( short vRefNum )
{
    Str255    volName;
    HParamBlockRec    HPBlock;
    long    lSize;

    HPBlock.volumeParam.ioCompletion    = NULL;
    HPBlock.volumeParam.ioNamePtr    = volName;
    HPBlock.volumeParam.ioVRefNum    = vRefNum;
    HPBlock.volumeParam.ioVolIndex    = 0;

    if ( PBHGetVInfo(&HPBlock, FALSE) != noErr )
        return FALSE;

    lSize = HPBlock.volumeParam.ioVNmALBlks *
           HPBlock.volumeParam.ioVAlBlkSiz;
    // see if size is less than 14.4 floppy
    if ( lSize <= ((long)1440 * 1024) )
        return TRUE;

    return FALSE;
}
How can I determine the amount of free space on a volume?

PBHGetVInfo() returns the number of free allocation blocks on a volume in the ioVFrBlk field and the number of bytes in the allocation blocks on that volume in the ioVA1B1kSz field. These two numbers multiplied together will result in the amount of free space available on the volume.

Topic-Related FAQs

- See also FAQ 10-4, "How do I force the Macintosh Finder to update its windows after my application has changed a file’s FndrInfo?"
- See also FAQ 20-7, "How can I get the name and location of my application?"
- See also FAQ 24-18, "What is the format of a PICT file?"
- See also FAQ 24-19, "How do I write a picture to a PICT file?"
- See also FAQ 24-20, "How do I read a picture from a PICT file?"
- See also FAQ 25-12, "What does the ‘1’ stand for in the Resource Manager routines?"
- See also FAQ 25-14, "How do I save a resource to a file?"
- See also FAQ 25-15, "How do I copy resources from one resource file to another?"
- See also FAQ 25-16, "How can I set the current resource file?"
- See also FAQ 25-18, "How can an application get the file reference number of its own resource file?"
See also FAQ 25-19, “Given a handle to a resource, how can I determine which resource file it came from?”

See also FAQ 25-20, “If I have multiple resource files open, what is the order in which their resources are searched for?”

See also FAQ 25-21, “How can I get the number of resources of a particular type in a resource file?”

See also FAQ 25-22, “How can I limit the searching for a resource to check only a single file?”

See also FAQ 25-23, “Is there a maximum number of items for Macintosh resources in a resource file?”

See also FAQ 25-24, “How does an application access the resources in its own resource fork and in the System File?”

See also FAQ 26-13, “When does the scrap get copied to a file?”

See also FAQ 26-14, “How can I tell if the scrap is in memory or has been written to a file?”

See also FAQ 27-24, “How do I create an AIFF file from sampled data?”
The Standard File Package provides the interface you see when a Mac application asks you to either open or save a file. In this chapter, you'll learn how to customize that interface, how to get and set the folder that initially appears when a standard file dialog box comes up, how to enable or disable the display of invisible files in a standard file list, and more.

There are three subsections in this chapter:

- Definitions
- Using the Standard File Package
- Customizing the Standard File Package
Definitions

This section describes dialog hook functions and filter functions, two of the many concepts that are discussed in this chapter.

What is the difference between the dialog hook function and the filter function in CustomGetFile() and CustomPutFile()?  

The dialog hook function handles item selections in the dialog box. It is called by CustomGetFile() and CustomPutFile() immediately after they call ModalDialog(). Each time the user selects one of the dialog items, the dialog hook function is passed the dialog record and an item number. The modal dialog filter function is called by the ModalDialog() function internally to CustomGetFile() and CustomPutFile() before an event is sent to the dialog hook function.

Related Topic

See also Chapter 6, “Dialogs and Controls.”

Using the Standard File Package

This section answers FAQs about the general use of the Standard File Package. It describes how you can select which files are displayed in the file selection list of the StandardGetFile() dialog. It also shows you how to specify which directory the Standard File Package dialog boxes initially display.

How do I open a file the user has selected using StandardGetFile()?  

StandardGetFile() returns an FSSpec of the selected file in the StandardFileReply parameter's sFile field. You can use this FSSpec to open a file.

The following example allows a user to select a file with StandardGetFile(). If the user did not select the Cancel button, it attempts to open the file with FspOpenDF().
```c
void DoOpenAFile()
{
    SFTypelist fileTypes;
    StandardFileReply reply;
    OSErr err;
    short fRefNum;

    StandardGetFile(nil, -1, fileTypes, &reply);
    if ( !reply.sfGood )
        return;

    err = FSpOpenDF( &reply.sfFile, fsRdWrPerm, &fRefNum);
    if ( err != noErr )
        return;

    // do file processing
}
```

**How can I tell if the Cancel button in a standard file dialog box gets hit?**

When the user selects the Cancel button in a Standard File Package call, the sfGood field in the reply structure returns false, as follows:

```c
void CheckForCancel()
{
    SFTypelist fileTypes;
    StandardFileReply reply;

    StandardGetFile(nil, -1, fileTypes, &reply);

    // did user hit cancel
    if ( !reply.sfGood )
        return;

    // files selected, do your processing
    ...
}
```
How can I select which files are displayed in the file list of the Standard Get File routines?

The numTypes and typeList parameters of StandardGetFile() and CustomGetFile() give you limited control over which files are visible in the displayed list. They allow you to filter out files based on a list of specified file types. If you want complete control over which files are displayed, you can add a file filter function. The file filter function is called by the Standard File package each time it identifies a file of an acceptable type. It receives a pointer to the file's catalog information record that it can evaluate to determine if the file is filtered. If the file name is to be displayed, the file filter should return false. If the file name is to be suppressed, the function should return false. To have StandardGetFile() and CustomGetFile() call a file filter function, pass a pointer to the function in the first parameter. Note that StandardGetFile() and CustomGetFile() use different filter functions. The two functions are defined as follows:

```pascal
Boolean MyStandardFileFilter(CInfoPBPtr pb)

Boolean MyCustomFileFilter(CInfoPBPtr pb, Ptr myData)
```

The following sample code puts up StandardGetFile() and CustomGetFile() dialog boxes that use file filter programs:

```c
void UseFileFilters()
{
    SFileTypeList fileTypes;
    StandardFileReply reply;
    Point pt = {-1, -1};
    FileFilterYDUPP uppCustomFileFilter;
    FileFilterUPP uppStandardFileFilter;

    uppStandardFileFilter = NewFileFilterProc(MyStandardFileFilter);
    StandardGetFile(uppStandardFileFilter, -1, fileTypes, &reply);

    uppCustomFileFilter = NewFileFilterYDProc(MyCustomFileFilter);
    CustomGetFile(uppCustomFileFilter, -1, fileTypes, &reply, 0, pt, nil, nil, nil, nil, &reply);
}
```
I have a custom dialog box that I pass to CustomGetFile() in which I allow the user to select a file type to look for. How can I get the Standard File Package to rescan the current directory, calling my file filter procedure if necessary?

To force the Standard File Package to rescan a directory and display new contents in the file list, add a dialog hook function to the CustomGetFile() call, as shown in the following example. In the hook function, test the item parameter for a value equal to the ID of your dialog box item that changes the selected file type. Do the processing necessary to determine if the selected file type has changed. If it has changed, return the pseudo-item sHookRebuildList to force the Standard File Package to rescan the files.

```pascal
//******************
DoSelectFiles()
{
   SFTypelist       fileTypes;
   StandardFileReply reply;
   Point             dlogLocation :
      CustomGetFile(nil, -1, fileTypes, &reply, 128,
         dlogLocation, NewDlgHookYDProc(myDlogHook),
         nil, nil, nil, &reply);
}

//******************
pascal short myDlogHook( short item,
               DialogPtr dlogPtr,
               Ptr dataPtr )
{
   short retItem;
   retItem = item;
   switch( item )
   {
```
case kItemId:
    // Do your processing after item is selected.
    // Check if new type selected.
    // If so tell Standard File
    // Package to rebuild list
    retItem = sfHookRebuildList;
    break;

    default:
    break;
}

return(retItem);

---

**How do I display only visible files and folders in a standard file dialog box?**

To display only visible files and folders in a standard file dialog box, write a file filter function to filter out invisible files.

**Related FAQ**

See also FAQ 9-4, "How can I select which files are displayed in the file list of the Standard Get File routines?"

---

**How do I tell the StandardGetFile() dialog to display every type of file?**

The second parameter, numTypes, is used to specify the number of file types to be displayed. The types are specified in the third parameter, typeList. To view all types of files, pass -1 in the numTypes parameter.
How do I set which folder should initially be shown in the Standard File Package calls?

To set the current volume and folder for the Standard File Package (the volume and folder that is initially displayed when the standard file dialog box appears), you can call CustomGetFile() or CustomPutFile() and specify your own dialog hook function that will allow you to specify the folder that will initially appear.

To do this, first create a dialog hook function. The dialog hook function is defined as follows:

```pascal
typedef pascal short myDlgHook(short item, DialogPtr theDialog, Ptr myDataPtr);
```

- `item` is the number of the item selected.
- `theDialog` is a pointer to the custom standard file dialog box.
- `myDataPtr` is a pointer to optional data whose address is passed to CustomGetFile() or CustomPutFile().

In the hook function, test for a value `sfHookFirstCall` passed into the first parameter, `item`. `sfHookFirstCall` is a pseudo-item that is passed to the dialog hook function immediately before it displays the dialog. When this value is passed in, take the `dataptr` value, cast it to a pointer to a `StandardFileReply` structure, and set the `replyPtr`, `sfFile.parID`, and `replyPtr->sfFile.vRefNum` fields to point to the folder you want to be initially displayed. Then have your function return the `sfHookChangeSelection` psuedo-item to signify that the reply record has been changed.

When you call CustomGetFile(), pass as the last parameter a pointer to a `StandardFileReply` structure variable (the same variable that you passed in the reply parameter).

The following example calls CustomGetFile() and makes the current folder the System Folder:

```c
//********************
void PutupCustomGetFile()
{
    SFTypelist fileTypes;
    StandardFileReply reply;
    Point pt = {-1, -1};

    CustomGetFile(nil, -1, fileTypes, &reply, 0, pt,
                  NewDlgHookYDProc(myDlgHook), nil, nil, nil,
                  &reply);
}
```
```pascal
pascal short myDlogHook ( short DialogPtr, short dlogPtr, short dataPtr )
{
    StandardFileReply *replyPtr;
    long sysDirId;
    short sysVRefNum;
    short retItem;
    OSErr err;

    retItem = item;

    switch ( item )
    {
        case sfHookFirstCall:
            err = FindFolder(kOnSystemDisk,
                             kSystemFolderType,
                             kDontCreateFolder,
                             &sysVRefNum, &sysDirId);
            if (err != noErr)
                break;
            replyPtr = (StandardFileReply *)dataPtr;
            replyPtr->sffile.parID = sysDirId;
            replyPtr->sffile.vRefNum = sysVRefNum;

            retItem = sfHookChangeSelection;
            break;

        default:
            break;
    }

    return (retItem);
}
```

Related FAQs

- See also FAQ 8-2, "What is the difference between the default directory and volume and the current directory and volume?"
- See also FAQ 8-10, "How can I get and set the default volume and folder?"
How do I get the current folder and volume that are initially displayed by the Standard File Package?

The current folder and volume are stored in the low-memory globals SFSaveDisk at location Ox0214 and CurDirStore at location Ox0398. To get the current folder and volume, read the data stored at these addresses, as follows:

```c
//********************
void GetCurrentVolumeAndFolder( long *curDirId,
                                short *curVRefNum )
{
    long       curDirId, *dirId;  
    short      curVRefNum, *vRefNum;

    dirId = (long *)Ox0398;
    vRefNum     = (short *)Ox0214;

    *curDirId = *dirId;
    *curVRefNum = *vRefNum;
}
```

You can also use the low memory global routines:

```c
//********************
void GetCurrentVolumeAndFolder( long *curDirId,
                                short *curVRefNum )
{
    *curDirId = LMGetCurDirStore();
    *curVRefNum = LMGetSFSaveDisk();
}
```
Related FAQs

- See also FAQ 8-2, "What is the difference between the default directory and volume and the current directory and volume?"

- See also FAQ 8-10, "How can I get and set the default volume and folder?"

- See also FAQ 8-16, "Why don't the current directory and volume used in the Standard File Package calls set the default directory and volume used in File MavAger calls?"

- See also FAQ 9-8, "How do I set which folder should initially be shown in the Standard File Package calls?"

Customizing the Standard File Package

This section answers FAQs on how to customize the Standard File Package dialog boxes.

**How can I customize StandardGetFile() and StandardPutFile() to display and process my own dialog box?**

The Standard File Package provides you with two routines, CustomGetFile() and CustomPutFile(), that allow you to specify the ID of a dialog box to display in place of the standard dialog boxes and dialog hook function. You can have a dialog box item in your custom dialog box mirror the functionality of the corresponding dialog box item in the standard dialog box by assigning it the same item ID. *Inside Macintosh: Files* lists the item IDs of the items in the standard dialog boxes.

If you want to eliminate one of the standard dialog box items, leave it in your dialog box’s item list but set its location outside the visible area of the dialog box. CustomGetFile() and CustomPutFile() expect these items to be there.

If you add new items to your dialog box that are not in the standard dialog boxes, you must handle them in your own dialog hook function. Make sure that the item
IDs of your new items do not duplicate the IDs of any of the items in the standard dialog box.

See the “Standard File Package” chapter in *Inside Macintosh: Files* for detailed information on customizing the Standard File Package dialog boxes.

**Why doesn’t my dialog hook function for CustomGetFile() recognize the sfHookFirstCall and sfHookLastCall pseudo-item numbers?**

Make sure that the first parameter (‘item’) in your dialog hook function is defined as ‘short’ and not as ‘int’. `sfHookFirstCall` is defined as -1, `sfHookLastCall` is defined as -2.

The declaration of the hook function should be as follows:

```pascal
pascal short MyDlogHook( short item, DialogPtr dlogPtr, Ptr dataPtr );
```

It should *not* be the following:

```pascal
pascal short MyDlogHook( int item, ...
```

**How do I select a folder with the Standard File Package?**

If you want to allow the user to select a folder with the Standard File Package, you must call `CustomGetFile()` to display your own custom dialog box and use a custom dialog hook function. *Inside Macintosh: Files* describes how you can do this in the “Standard File Package” chapter.

**Related FAQs**

- See also FAQ 8-2, “What is the difference between the default directory and volume and the current directory and volume?”

- See also FAQ 8-16, “Why don’t the current directory and volume used in the Standard File Package calls set the default directory and volume used in File Manager calls?”
The Finder is the most widely used Macintosh application. The Desktop database is where the Finder stores information linking applications and their related documents. In this chapter, you'll learn how to access the Desktop file, add an icon to your application, create a bundle, and other Finder related issues.

There are three subsections in this chapter:

- Definitions
- The Finder
- The Desktop

Definitions

This section defines the Desktop database, one of the many concepts that are discussed in this chapter.
What is the Desktop database?

The Desktop database contains information about files and directories on a volume. It contains all icon definitions and their associated file types. It also stores the location of all applications on the disk, any comments that pertain to an application, and a list of all file types an application can open. Each volume has its own copy of the Desktop database file.

The Finder

This section answers FAQs about the Finder and describes how to update the Finder to properly display up-to-date file information.

How do I get my application to respond to the Finder for actions such as printing a document or opening a document double-clicked on by the user?

These functions are dealt with via the following four Apple events:

- Open Application
- Open Documents
- Print Documents
- Quit Application

Your application must be equipped to handle these Apple events.

Related FAQ

- See also FAQ 1-1, “What are the four required Apple events?”

Related Topic

- See also Chapter 1, “Apple Event Manager.”
How do I make the Finder see a new file that I created, or if I change the file type, how do I display a new icon for it?

To force the Finder to update the folder, you need to set the last modification date and time of the file's parent folder to the current date and time, as shown in the following example. Call PBGetCatInfo() to get the current directory information, get the current date and time by calling GetDateTime() and store it in the ioDrMdDat field of the CInfoPBRec structure, and then call PBSetCatInfo() to set the folder's last modification date and time.

```c
//***************************************************
OSErr UpdateFolder ( short vRefNum, long parID )
{
    CInfoPBRec myCPB;
    Str63 strFolderName;
    short err;

    strFolderName[0] = 0;
    myCPB.hFileInfo.ioNamePtr = strFolderName;
    myCPB.hFileInfo.ioVRefNum = vRefNum;
    myCPB.hFileInfo.ioDirID = parID;
    myCPB.hFileInfo.ioFDirIndex = -1;
    myCPB.hFileInfo.ioFVersNum = 0;
    err = PBGetCatInfo( &myCPB, false );
    if ( err )
        return err;

    GetDateTime( &myCPB.dirInfo.ioDrMdDat );

    myCPB.hFileInfo.ioVRefNum = vRefNum;
    myCPB.hFileInfo.ioDirID = parID;
    myCPB.hFileInfo.ioFDirIndex = -1;
    myCPB.hFileInfo.ioFVersNum = 0;
    myCPB.hFileInfo.ioNamePtr[0] = 0;
    err = PBSetCatInfo( &myCPB, false );

    return err;
}
```
How do I force the Macintosh Finder to update its windows after my application has changed a file's FndrInfo?

The Finder updates the appearance of the Desktop after it detects that the volume has changed, which is indicated by a change in the volume’s modification date. This occurs when a file is created, deleted, or moved to another folder. When the Finder notices that the volume’s modification date has changed, it scans all of the open folders for changes. This scanning process takes place about once every 10 seconds.

Changing a file’s FndrInfo does not change the modification date. You can force the Finder to update the Desktop by manually setting the folder’s last modification date, as follows:

```c
//**************************
OSErr UpdateFolder ( short vRefNum , long dirID )
{
    CInfoPBRec myCPB ;
    Str63 strFolderName ;
    short err ;

    strFolderName[0] = 0 ;
    myCPB.hFilelnfo.ioNamePtr = strFolderName ;
    myCPB.hFilelnfo.ioVRefNum = vRefNum ;
    myCPB.hFilelnfo.ioDirID = dirID ;
    myCPB.hFilelnfo.ioFDirlndex = -1 ;
    myCPB.hFilelnfo.ioFVersNum = 0 ;
    err = PBGetCatlnfo( &myCPB, false ) ;
    if ( err )
        return err ;

    // Get the Current date and time
    GetDateTime ( &myCPB.dirInfo.ioDrMdDat ) ;

    myCPB.hFilelnfo.ioVRefNum = vRefNum ;
    myCPB.hFilelnfo.ioDirID = dirID ;
    myCPB.hFilelnfo.ioFDirlndex = -1 ;
    myCPB.hFilelnfo.ioFVersNum = 0 ;
    myCPB.hFilelnfo.ioNamePtr [ 0 ] = 0 ;
    err = PBSetCatlnfo( &myCPB, false ) ;

    return err ;
}
```
Related Topic

See also Chapter 8, “Files: File Manager.”

The Desktop

This section answers FAQs about the Desktop, including how to access the Desktop file. It also describes the steps necessary to attach an icon to an application and the application’s files along with other aspects of icons in the Finder.

How do I access the Desktop file?

You can get the file reference number of the Desktop database by calling PTDTGetPatch() or PBDTOpenInform(). First, you must check if the volume supports all of the Desktop functions. This is done by checking the bHasDesktopMgr bit in the vMAtrib field of the GetVolParmsInfoBuffer structure. This is bit number 12. If the Desktop functions are supported, you can then call PTDTGetPatch() or PBDTOpenInform() to get the Desktop database. You cannot use the file reference number of the Desktop file in any File Manager routine.

The following example gets the reference number of the Desktop database and also detects if the file was empty when opened:

```c
//***************************
OSErr GetDesktopDB( short vRefNum,
                    short *dtRefNum,
                    Boolean *isNew)
{
    OSErr error;
    HParamBlockRec hpBlock;
    GetVolParmsInfoBuffer volParmsInfo;
    DTPBRec dtpb;

    // check if Desktop Manager calls are supported
    hpBlock.volumeParam.ioCompletion   = nil;
    hpBlock.volumeParam.ioNamePtr     = nil;
    hpBlock.volumeParam.ioVRefNum     = vRefNum;
    hpBlock.ioParam.ioBuffer          = (Ptr)&volParmsInfo;
    hpBlock.ioParam.ioReqCount        =
```
sizeof(GetVolParmsInfoBuffer);
error = PBHGetVolParms( &hpBlock, FALSE );

if ( error == noErr )
{
    // check bHasDesktopMgr bit in vMAtrib
    if ( (volParmsInfo.vMAtrib >> bHasDesktopMgr) & 0x01 )
    {
        dtpb.ioNamePtr = nil;
        dtpb.ioVRefNum = vRefNum;
        error = PBDTOpenInform(&dtpb);

        // check if PBDTOpenInform just created a new
        // Desktop db also meaning it is empty
        *isNew = ((dtpb.ioTagInfo & ll) == 0);

        // if PBDTOpenInform returns paramErr use
        // PBDTGetPath
        if ( error == paramErr )
        {
            error = PBDTGetPath(&dtpb);

            // don't know if new, assume not
            *isNew = false;
        }
        *dtRefNum = dtpb.ioDTRefNum;
    }
    else
    {
        error = paramErr;
    }
}
return ( error );

How can I get the application that opens a file with a given creator?

Call the Desktop Manager function PBDTGetApp().
Why is my application's old icon still displayed on the Desktop after I change the icon?

The Desktop file is a resource file maintained by the Finder in which icons, file types, applications, and other types of information are stored. Even after an icon is changed, the old icon may still be in the Desktop file and the Finder may not know that it has been changed. To get the Finder to use the new icon, you need to rebuild the Desktop file. To do this, hold down the Option and Command keys on startup (or on insertion of the disk in question if the disk is removable and not in the drive during startup).

How do I attach an icon to an application?

To attach an icon, you create a bundle. A bundle is a resource ('BNDL') that groups together a collection of resources used by the Finder to associate an application and its documents with their icons. A bundle uses three other types of resources: an ‘ICN#’, an ‘FREF’, and an ‘sign’.

The ‘ICN#’ resource type is an icon list. Each ‘ICN#’ resource contains one or more icons of an icon family, representing different sizes and color depths.

An ‘FREF’ is a file type reference. It ties a file type to a local icon resource ID. This will be mapped by the ‘BNDL’ onto an actual resource ID number of an ‘ICN#’ resource. You need one ‘FREF’ for each file type that has an icon.

Every application (or other file with a bundle) has a unique four-character signature ('sign'). The Finder uses this to identify an application.

To associate an icon family to an application and its documents, follow these steps:

1. Create an icon list ('ICN#') resource and the other members of the icon family.
2. Create a ‘BNDL’ resource.
3. In the ‘BNDL’ resource, list the resource ID of the application’s ‘ICN#’ resource and assign it a local ID of 0.
4. Create an ‘FREF’ resource and assign the application a file type of ‘APPL’ with a local ID of 0.
5. In the ‘BNDL’ resource, list the ‘FREF’ resource ID for the application file and assign it a local ID of 0.
6. Create another icon family for your application's documents.

7. In the 'BNDL' resource, list the resource ID of each document's 'ICN#' resource and assign it a unique local ID.

8. Create an 'FREF' resource for each document and assign it the file type of the document and a unique local ID.

9. In the 'BNDL' resource, add each document's 'FREF' resource ID and assign it a unique local ID.

Resource editors group all of these resources together in the 'BNDL' resource editing screen.

Figure 10-1 illustrates what a typical bundle resource looks like.

![Figure 10-1: The 'BNDL' resource editing screen from Resorcerer](image)

Related Topic

See also Chapter 13, "Icons."

**How do I get the icon of a file with a given type and creator?**

The Desktop Manager function `PBDTGetIcon()` returns the bitmap for an icon associated with a file of a given type and creator.
Why is my file displaying the standard icon and not my bundled icon?

The file's Finder flags contain a hasBundle flag. This bundle bit specifies that the file has a 'BNDL' resource that associates the file with its own icons. If the bundle bit is not set, the Finder displays the default icons (see Figure 10-2). If the bundle bit and the file's hasBeenInitialized flag are both set, the Finder displays a file's icons from information in the Desktop database. If the hasBeenInitialized flag is not set, the Finder installs the bundle resource information into the Desktop database.

Figure 10-2: The default application and document icons

Related FAQ

See also FAQ 8-34, "How do I create an invisible file?"

Related Topic

See also Chapter 13, "Icons."

Topic-Related FAQs

See also FAQ 1-5, "How does the Finder tell my application that one of its documents was double-clicked?"
See also FAQ 8-17, "How do I access the top directory of a volume and on the Desktop?"

See also FAQ 8-33, "How do I alter a file without automatically changing the time stamp?"

See also FAQ 8-39, "How can I get the Finder label menu color and label string of a file?"

See also FAQ 13-3, "What types of icons can the Finder display?"

See also FAQ 13-4, "How can I draw an icon as it appears in the Finder in the different Finder states?"

See also FAQ 16-25, "How do I hide the menu bar of an application?"
The Gestalt Manager tells you just about everything you'd want to know about the machine your application is running on. You can find out if your Mac is 680x0 or PowerPC-driven, how much memory it has, or even what type of keyboard is attached to it.

The FAQs in this chapter deal with how you use and extend the Gestalt mechanism. (Appendix B contains a complete listing of all the Gestalt codes available as this book went to press.)

How can I tell if Gestalt Manager is available?

You can determine whether or not the Gestalt Manager is available by testing for the existence of the Gestalt trap. Call NGetTrapAddress() with the trap number for _Gestalt, as follows:

```c
Boolean IsGestaltAvailable()
{
    return ( NGetTrapAddress(_Gestalt, OSTrap) !=
             GetToolboxTrapAddress(_Unimplemented) );
}
```
Note that `Gestalt()` is an Operating System trap.

**Related FAQs**

- See also FAQ 30-2, "What is the Unimplemented() routine?"
- See also FAQ 30-15, "How can I tell if a System routine is available in my system?"

---

**How do I use Gestalt to get information?**

`Gestalt()` is used to get information about the operating environment. It is defined as follows:

```
err = Gestalt(OSType selectorCode, long &response);
```

To use `Gestalt()`, you pass it a selector code in the first parameter. This selector code specifies what information you are requesting. In the second parameter, `Gestalt()` returns a response that contains the information.

Selector codes have different suffixes, each of which signals a different type of response. These suffixes and their responses are listed in Table 11-1.

---

**Table 11-1:**

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attr</td>
<td>32-bit flags that must be compared against constants defined in <code>Gestalt.h</code></td>
</tr>
<tr>
<td>Count</td>
<td>The total of the kind of type selected</td>
</tr>
<tr>
<td>Table</td>
<td>The base address of a table</td>
</tr>
<tr>
<td>Type</td>
<td>An index describing a particular type of feature</td>
</tr>
<tr>
<td>Version</td>
<td>A version number</td>
</tr>
</tbody>
</table>

---

**Related FAQ**

- See also FAQ 25-5, "How do I load a resource into the System heap?"
What are the available Gestalt values?

See Appendix B.

How do I register new selectors in Gestalt?

You can add a new selector code by calling the NewGestalt() function. NewGestalt() takes two parameters: a four character Gestalt selector code that is to be registered with the Gestalt Manager and the address of a new selector function that is executed when Gestalt is called with that selector code. The selector function must have the following prototype:

```pascal
OSErr MySelectorFunction(OSType selector, long *response);
```

The code in the selector function will test the value passed in the selector parameter and then return the proper value in the response field. The following example tests for Gestalt values 'FAQ1' and 'FAQ2' and passes back 0x0001 and 0x0002, respectively:

```pascal
OSErr MyGestaltFunc(OSType selector, long *reply) {
    if (selector == 'FAQ1')
        *reply = 0x0001;
    if (selector == 'FAQ2')
        *reply = 0x0002;
    return noErr;
}
```

The code that installs the new selector code must load the selector function into the System heap (most likely by loading a 'CODE' resource) and then call NewGestalt() with the selector type as the first parameter and a pointer to the function in System memory as the second parameter. In the universal headers, this procedure pointer is defined as type SelectorFunctionUPP. The function NewSelectorFunctionProc() creates a universal procedure pointer to a Gestalt selector function. The following code creates a universal procedure pointer for the new customized Gestalt selector function and passes it to NewGestalt():

```pascal
SelectorFunctionUPP myGestaltUPP;
myGestaltUPP = NewSelectorFunctionProc(MyGestaltFunc);
```
NewGestalt('FAQS', myGestaltUPP)

Related FAQs

- See also FAQ 11-6, "How can I use Gestalt to communicate between processes?"
- See also FAQ 30-9, "How do I get my extension and control panel to communicate or share data?"

Can my Gestalt selector function have global variables?

As with other types of standalone code, Gestalt functions can access their own global variables by setting up the A4 register in 68K machines.

In CodeWarrior, the A4 register is set up as follows:

```pascal
OSErr myGestalt(OSType selector, long *response) {
    long oldA4;

    // Set up A4, so we can access our globals.
    oldA4 = SetCurrentA4();

    // Do your stuff

    // Restore the value of A4 on the way out.
    SetA4(oldA4);
}
```

In Think C, the A4 register is set up as follows:

```pascal
OSErr myGestalt(OSType selector, long *response) {
    // Set up A4, so we can access our globals.
    RememberAo();
    SetUpA4();

    // Do your stuff

    // Restore the value of A4 on the way out.
```
CHAPTER 11: Gestalt Manager

This A4 register setup is unnecessary in native PowerPC applications.

Related FAQ
- See also FAQ 15-20, “What is the A4 register used for in code resources?”

How can I use Gestalt to communicate between processes?

You can use Gestalt() to allow programs to have access to common data. To do this, you must first write your Gestalt selector routine and register it with NewGestalt(). In response to a selector type, the Gestalt function can pass back the handle or address of a global variable that it maintains.

Related FAQs
- See also FAQ 11-4, “How do I register new selectors in Gestalt?”
- See also FAQ 30-9, “How do I get my extension and control panel to communicate or share data?”

Related Topic
- See also Chapter 20, “Processes: Process Manager.”
Macworld Mac Programming FAQs
In general, monitors, off-screen memory areas, and printers can all be represented as graphics devices. Graphics devices and their associated data structures were created to link output devices with their associated device drivers. In this chapter, you'll learn about Gamma tables and the techniques you can use to dim a monitor to black. You'll learn how to get and set the characteristics of a graphics device, including the pixel depth, color versus grayscale, and which monitor is the main monitor (the monitor on which the menu bar appears at startup). You'll also learn how to determine how many monitors are attached to a machine and how to capture an image of your entire screen, including your Desktop.

There are four subsections in this chapter:

- Definitions
- Getting Device Information
- Manipulating Graphics Devices
- Working with Graphics Devices
Definitions

This section answers FAQs about two important terms that are discussed in this chapter: the graphics device (the basis of the chapter) and Gamma tables (used in specialized functions).

What is a Gamma table?

A Gamma table is a set of values used by a video card’s software driver to compensate for the fact that the intensity of each color is not proportional to the intensity of the display’s electronic beam. It does this by modifying the value loaded in the color lookup table through the use of a single correction table. By adjusting the values in this table up or down, the screen can be made lighter or darker.

Related FAQs

- See also FAQ 12-9, “How can I fade my display to black and back again or to another color?”
- See also FAQ 12-10, “How can I fade the screen of a direct color device to black?”

What is the main graphics device?

The main device is the monitor that contains the menu bar. Unless the user has set this value in the parameter RAM to another device (through the monitors control panel, as shown in Figure 12-1), the main device is the screen whose video card is in the lowest-numbered slot.

Figure 12-1: The Monitors control panel displays the settings and positioning of all of the monitors connected to your Macintosh. It allows you to set the color mode, the pixel depth, and the position of the monitors.
CHAPTER 12: Graphics Devices

Getting Device Information

This section answers FAQs on retrieving information about video devices and off-screen graphics worlds. It describes how to determine the maximum pixel depth that a device can support and provides examples of getting the current pixel depth and color mode.

It is often necessary to determine how many monitors are attached to your computer and the size of each one, which is also discussed in this chapter.

**How can I determine the maximum pixel depth that a screen can support?**

The maximum pixel depth that a screen can support can be determined by calling HasSetDepth() on the graphics device handle of the monitor. If your machine only has one monitor, call GetMainDevice() to return a handle to a graphics device. If you have more than one screen, walk through the device list and get the proper graphics device handle.

To determine the maximum supported depth, call HasDepth() with the different pixel depths. The highest successful depth is the maximum depth that is supported.

The following sample routine gets the maximum supported pixel depth of a graphics device:

```c
//********************
short GetSupportedDepth( GDHandle hGD )
{
    if ( !HasDepth(hGD, 2, 0, 0) )
        return( 1 );

    if ( !HasDepth(hGD, 4, 0, 0) )
        return( 2 );

    if ( !HasDepth(hGD, 8, 0, 0) )
        return( 4 );

    if ( !HasDepth(hGD, 16, 0, 0) )
        return( 8 );

    return( 16 );
}
```
**How can I get the screen with the greatest color depth?**

GetMaxDevice() returns a handle to the video device with the greatest color depth.

**How can I determine which monitor is the main monitor?**

GetMainDevice() returns a handle to the main screen (the screen with the menu bar).

**How can I determine how many monitors are attached to my computer?**

You can determine the number of monitors attached to a computer by walking through the device list and testing for active screen devices. To get the first device in the device list, call GetDeviceList(). To get the next device in the list, call GetNextDevice(). When GetNextDevice() returns nil, you have iterated through the entire list. For each returned device, call TestDeviceAttribute() to test if the device is a screen device and if it is active.

The following example returns the number of active monitors attached to a computer:

```c
//********************
short GetNumGDevices( void )
{
    short count = 0;
    GDHandle hDevice;
    for ( hDevice = GetDeviceList(); hDevice != nil;
    hDevice = GetNextDevice(hDevice) )
    {
        if ( (TestDeviceAttribute(hDevice, screenDevice))
            &&(TestDeviceAttribute(hDevice, screenActive)))
            count += 1;
```
How do I find the current pixel depth and color mode of the screen?

To get the current pixel depth and color mode of your screen, you must first get its graphics device handle. If your machine only has one monitor, call GetMainDevice() to return a handle to a graphics device. If you have more than one screen, walk through the device list and get the proper graphics device handle.

The color mode of the monitor is stored in the gdFlags field of the GDevice structure referenced by the graphics device handle. If the first bit is 1, then the monitor is set for color mode. If it is not set, then the monitor is in black-and-white mode. The current pixel depth setting is stored in the gdMode field of the GDevice structure. gdMode is set to 128 for 1-bit depth, 129 for 2-bit depth, 130 for 4-bit depth, 131 for 8-bit depth, and 132 for 16-bit depth.

The following sample routine passes back the pixel depth and whether or not the monitor is set for color:

```c
#define kIsColorMask 0x0001
#define kPixelDepth1 128
#define kPixelDepth2 129
#define kPixelDepth4 130
#define kPixelDepth8 131
#define kPixelDepth16 132

void GetMonitorMode( GDHandle hGD, short *pixelDepth, Boolean *isColor )
{
    HLock( (Handle)hGD );
    if ( ((GDPtr)*hGD)->gdFlags & kIsColorMask )
        *isColor = TRUE;
    else
        *isColor = FALSE;
    *pixelDepth = GDMode( (GDPtr)*hGD );
    return count;
}
```
*isColor = FALSE;

switch ( ((GDPtr)*hGD)->gdMode )
{
    case kPixelDepth1:
        *pixelDepth = 1;
        break;
    case kPixelDepth2:
        *pixelDepth = 2;
        break;
    case kPixelDepth4:
        *pixelDepth = 4;
        break;
    case kPixelDepth8:
        *pixelDepth = 8;
        break;
    case kPixelDepth16:
        *pixelDepth = 16;
        break;
    default:
        *pixelDepth = -1;
        break;
}

HUnlock( (Handle)hGD );

The pixel depth can also be obtained by examining the pixmap of the graphics device referenced by the PixMapHandle stored in its gdPMap field as shown in the following example. The pixmap's pixelSize field contains the current pixel depth of the graphics device.

//**************************
short GetMonitorPixMapDepth( GDHandle hGD )
{
    PixMapHandle hPixMap;
    short pixelDepth;

    HLock( (Handle)hGD );
How do I find the pixel size of a monitor?

The pixel size of a monitor can be found in the gdRect field of the graphics device structure.

The following example gets the size of the main graphics device:

```c
//************************
void GetMainGDSize( short *height,
  short *width )
{
    GDHandle hGD;

    hGD = GetMainDevice();

    HLock((Handle)hGD);
    *height = ((GDPtr)*hGD)->gdRect.bottom -
              ((GDPtr)*hGD)->gdRect.top;
    *width = ((GDPtr)*hGD)->gdRect.right -
             ((GDPtr)*hGD)->gdRect.left;
    HUnlock((Handle)hGD);
}
```

Manipulating Graphics Devices

This section answers FAQs on manipulating graphics devices, particularly video screens. Two different approaches on fading the screen are presented. One, the infamous gamma fading, works with direct devices. The other method uses palettes and color tables and works with indexed devices. This section also covers how to set the pixel depth and color of a monitor.
How can I fade my display to black and back again or to another color?

A screen can be faded to black by slowly reducing the RGB values stored in the device's color table to \((0,0,0)\). To fade the screen back to its original state, the colors are increased until they match the original RGB values. (This will only work on indexed video devices.)

First, get the GDevice structure of the screen that you want to fade. To get the GDevice of the main screen, call GetMainDevice(). You can get the color table of the device from the device's pixmap, gdPMap, in the pmTable field. Keep a copy of the original value of the color table to restore later. Reduce the values of the red, green, and blue components of each entry in the color table. After all of the entries have been modified, call SetEntries() to set the new color table entries for the gDevice. When you are done fading, set all of the entries to black. To fade the screen back to its original state, incrementally add to the RGB components of the blacked-out color table until you reach the original values.

The screen can be dimmed to any RGB value by reducing the colors to the specified RGB values instead of to \((0,0,0)\).

The following example fades the main screen to black, waits for a second, and then restores the monitor back to its original state:

```c
//**********************
#define kMaxColors 0xFF
#define kFadeIn 0
#define kFadeOut 1

CTabHandle pFadeCTable;
CTabHandle pOrigCTable;

void FadeDevice( GDHandle screen, short fadeCmd );
void FadeOutDevice( RGBColor *fadeValues );
void FadeInDevice( RGBColor *fadeValues );
void BlackoutDevice( void );

//**********************
void DoFade( void )
{
    long finalTicks;
```
FadeDevice( GetMainDevice(), kFadeOut );
Delay(60, &finalTicks);
FadeDevice( GetMainDevice(), kFadeIn );

//******************************
void FadeDevice( GDHandle fadeGDevice,
                 short    fadeState )
{
    GDHandle    oldDev;
    RGBColor    fadeValues[kMaxColors];
    short index;

    // must be an indexed device
    if ( ( **fadeGDevice).gdType == directType )
        return;

    oldDev = GetGDevice();
    SetGDevice( fadeGDevice );

    // copy color table to manipulate and restore later
    pFadeCTable = ( (*fadeGDevice)->gdPMap )->pmTable;
    pOrigCTable = ( (*fadeGDevice)->gdPMap )->pmTable;
    HandToHand( ( Handle* )&pOrigCTable );

    HLock( ( Handle )pFadeCTable );
    HLock( ( Handle )pOrigCTable );

    // calculate the fade values for the RGB components for
    // each entry in the color table
    for (index = 0; index < ( *pFadeCTable )->ctSize; index++)
    {
        fadeValues[index].red =
        (*pFadeCTable)->ctTable[index].rgb.red / 100;
        fadeValues[index].green =
        (*pFadeCTable)->ctTable[index].rgb.green / 100;
        fadeValues[index].blue =
        (*pFadeCTable)->ctTable[index].rgb.blue / 100;
    }

    if ( fadeState == kFadeOut )
        FadeOutDevice( fadeValues );
else
    FadeInDevice( fadeValues );

    // restore the clut to original value
    for ( index = 0; index < ( *pFadeCTable )->ctSize;
        index++ )
    {
        (*pFadeCTable)->ctTable[index].rgb.red =
            ( *pOrigCTable )->ctTable[index].rgb.red;

        (*pFadeCTable)->ctTable[index].rgb.green =
            ( *pOrigCTable )->ctTable[index].rgb.green;

        (*pFadeCTable)->ctTable[index].rgb.blue =
            ( *pOrigCTable )->ctTable[index].rgb.blue;
    }

    // color table changed, must reset ctseed
    (*pFadeCTable )->ctSeed = GetCTSeed();
    MakeITable( nil, nil, 0 );
    if ( noErr != QDError() )
        return;

    HUnlock( ( Handle )pFadeCTable );
    HUnlock( ( Handle )pOrigCTable );

    SetGDevice( oldDev );
    DisposeHandle( ( Handle )pOrigCTable );

    //***********************
    void FadeOutDevice( RGBColor *fadeValues)
    {
        short index, percentFade;

        for ( percentFade = 100; percentFade >= 0; percentFade-- )
        {
            for ( index = 0; index < ( *pFadeCTable )->ctSize;
                index++ )
            {
                if (((*pFadeCTable)->ctTable[index].rgb.red
                    > fadeValues[index].red)
(*pFadeCTable)->ctTable[index].rgb.red -= fadeValues[index].red;

if ((*pFadeCTable)->ctTable[index].rgb.green > fadeValues[index].green)
(*pFadeCTable)->ctTable[index].rgb.green -= fadeValues[index].green;

if ((*pFadeCTable)->ctTable[index].rgb.blue > fadeValues[index].blue)
(*pFadeCTable)->ctTable[index].rgb.blue -= fadeValues[index].blue;

SetEntries( 0, (*pFadeCTable)->ctSize - 1, (*pFadeCTable)->ctTable);

BlackoutDevice();

//******************************
void FadeinDevice( RGBColor *fadeValues )
{
    short index, speedIndex;

    BlackoutDevice();

    for (speedIndex = 0; speedIndex < 100; speedIndex++)
    {
        for (index = 0; index < (*pFadeCTable)->ctSize;
            index++)
        {
            if ((*pFadeCTable)->ctTable[index].rgb.red < (*pOrigCTable)->ctTable[index].rgb.red)
                (*pFadeCTable)->ctTable[index].rgb.red += fadeValues[index].red;

            if ((*pFadeCTable)->ctTable[index].rgb.green < (*pOrigCTable)->ctTable[index].rgb.green)
                (*pFadeCTable)->ctTable[index].rgb.green += fadeValues[index].green;
        }
    }
}
if ((*pFadeCTable)->ctTable[index].rgb.blue <
(*pOrigCTable)->ctTable[index].rgb.blue)
(*pFadeCTable)->ctTable[index].rgb.blue
+= fadeValues[index].blue;
}
SetEntries( 0, ( *pFadeCTable )->ctSize - 1,
( *pFadeCTable )->ctTable );
}

//***********************
void BlackoutDevice(void)
{
    short index;

    for ( index = 0; index < ( *pFadeCTable )->ctSize; index++ )
    {
    ( *pFadeCTable )->ctTable[index].rgb.red   = 0;
    ( *pFadeCTable )->ctTable[index].rgb.green = 0;
    ( *pFadeCTable )->ctTable[index].rgb.blue  = 0;
    }

    SetEntries( 0, ( *pFadeCTable )->ctSize - 1,
( *pFadeCTable )->ctTable );
}

Related FAQs

☑ See also FAQ 12-1, "What is a Gamma table?"

☑ See also FAQ 12-10, "How can I fade the screen of a direct color device to black?"

Related Topic

☑ See also Chapter 2, "Color."
How can I fade the screen of a direct color device to black?

To fade a direct color device, you have to modify the Gamma table values for that screen. The Gamma table contains data that is used to correct the intensity levels of the electronic beam projected onto the face of the monitor. The purpose is to adjust for the non-linear aspect of each color on a video display with respect to the intensity of the electronic beam. By adjusting the values of the table data, the screen can be dimmed or brightened.

The Gamma table structure is as follows:

```c
struct GammaTbl
{
    short gVersion;    /* gamma version number */
    short gType;       /* gamma data type */
    short gFormulaSize; /* Formula data size */
    short gChanCnt;    /* number of channels of data */
    short gDataCnt;    /* number of values/channel */
    short gDataWidth;  /* bits/corrected value (data packed to next larger byte size) */
    short gFormulaData[1]; /* data for formulas followed by gamma values */
};
```

The Gamma table values can be retrieved by calling PBSstatus() with the csCode of the CntrlParam structure set to cscSetGamma and the ioCRefNum field set to the reference number of the display device driver (found in the gdRefNum field of a GDevice). The actual Gamma values used to correct the colors are located at an offset of gFormulaSize bytes from the gFormulaData field. The number of these values is the number of lookup tables stored (gChanCnt) times the number of values in each table (gDataCnt). These values can be modified and then stored in the GDevice by the PBControl() call with the csCode set to cscSetGamma, the reference number of the display device set in ioCRefNum, and the pointer to the modified Gamma table structure in the cParam field. If the device being faded is an indexed device, the color table must also be reset after the Gamma table is modified.

The following example uses gamma fading to fade the main screen in and then fade out, delay for two seconds, and fade the screen in and out again at a slower rate. The sample function SetupGammaTables() gets a pointer to the device’s Gamma...
table and makes two copies of it: one to keep as the original and the other on which to perform the fading. GetDeviceGammaTable() calls the Toolbox function PBStatus() to get the device's Gamma table. GammaFadeDevice() sets the values in the Gamma table to the specified percent of their original value. It then gets a pointer to the gamma data by getting the address of the gFormulaData field and incrementing by the formula data size, and gets the number of gamma data entries by multiplying the number of channel correction tables by the number of values in each channel correction table, adjusting the values in each entry by the specified percent. SetDeviceGammaTable() installs the faded Gamma table into the graphics device by calling PBControl(). For indexed devices, it sets the color table to the modified values.

```c
//********************************
GammaTblHandle hOrigGammaTable, hFadeGammaTable;
short gammaTableSize;

//******
void DoGammaFade()
{
    short i;
    long finalTicks;
    GDHandle mainGDevice;

    mainGDevice = GetMainDevice();

    if ( !TestDeviceAttribute(mainGDevice, screenDevice) ||
        TestDeviceAttribute(mainGDevice, noDriver))
        return;

    SetupGammaTables(mainGDevice);

    for(i=100; i >= 0; i-=5)
        GammaFadeDevice(mainGDevice, i);

    for(i=0; i <= 100; i+=5)
        GammaFadeDevice(mainGDevice, i);

    Delay(120, &finalTicks);

    for(i=100; i >= 0; i-=2)
        GammaFadeDevice(mainGDevice, i);
```
for (i = 0; i <= 100; i += 2) 
    GammaFadeDevice(mainGDevice, i);

DisposeGammaTables();

//********
OSErr SetupGammaTables(GDHandle theGDevice) 
{
    short err = noErr;
    GammaTblPtr currentGTable;

    if (err = GetDeviceGammaTable(theGDevice, &currentGTable)) 
        return(err);

    gammaTableSize = sizeof(GammaTbl) +
                      currentGTable->gFormulaSize +
                      (currentGTable->gChanCnt *
                      currentGTable->gDataCnt *
                      currentGTable->gDataWidth / 8);

    hOrigGammaTable =
                      (GammaTblHandle) NewHandle(gammaTableSize);
    hFadeGammaTable =
                      (GammaTblHandle) NewHandle(gammaTableSize);

    BlockMove((Ptr) currentGTable, (Ptr) *(hOrigGammaTable),
              gammaTableSize);

    return(0);
}

//********
OSErr GammaFadeDevice(GDHandle theGDevice, short percent) 
{
    short err = noErr;
    register long numEntries, i, theNum;
    unsigned char *dataPtr;

    BlockMove((Ptr) *hOrigGammaTable,
              (Ptr) *(hFadeGammaTable), gammaTableSize);

    dataPtr = (unsigned char *)

```c
    (*hFadeGammaTable)->gFormulaData +
    (*hFadeGammaTable)->gFormulaSize;

    numEntries = (*hFadeGammaTable)->gChanCnt *
    (*hFadeGammaTable)->gDataCnt;

for(i=0; i < numEntries; i++)
{
    theNum = dataPtr[i];
    theNum = (theNum * percent) / 100;
    dataPtr[i] = theNum;
}

err = SetDeviceGammaTable(theGDevice, hFadeGammaTable);
return(err);
}

//********
void DisposeGammaTables()
{
    DisposeHandle((Handle) hOrigGammaTable);
    DisposeHandle((Handle) hFadeGammaTable);
}

//********
OSErr GetDeviceGammaTable(GDHandle theGDevice, GammaTblPtr *theTable)
{
    short err = noErr;
    CntrlParam theCPB:

    theCPB.csCode = cscGetGamma;
    theCPB.ioRefNum = (*theGDevice)->gdRefNum;
    *(GammaTblPtr **)theCPB.csParam = theTable;
    err = PBStatus((ParmBlkPtr)&theCPB, 0);

    return(err);
}

//********
OSErr SetDeviceGammaTable(GDHandle theGDevice, GammaTblPtr *theTable)
```
{ 
    CTabHandle cTab;
    GDHandle saveGDevice;
    CntrlParam theCPB;
    short err = noErr;

    theCPB.csCode = cscSetGamma;
    theCPB.ioCRefNum = (*theGDevice)->gdRefNum;
    *(GammaTblPtr **)theCPB.csParam = theTable;
    err = PBControl((ParmBlkPtr *)&theCPB, 0);

    if (err == noErr) {
        saveGDevice = GetGDevice();
        SetGDevice(theGDevice);
        cTab = (*((theGDevice)->gdPMap)->pmTable;
        SetEntries (0, (*cTab)->ctSize, (*cTab)->ctTable);
        SetGDevice(saveGDevice);
    }

    return(err);
}

Related FAQs

- See also FAQ 12-1, "What is a Gamma table?"
- See also FAQ 12-9, "How can I fade my display to black and back again or to another color?"

**How can I get the image of a screen (as in a screen capture)?**

To create a PICT of the screen image requires getting the Window Manager port, creating a picture, and using `CopyBits()` to copy the pixmap of the Window Manager port.

`GetCWMgrPort()` and `GetWMgrPort()` are used to get the GrafPort of the Window Manager port on Color QuickDraw and on the original monochrome QuickDraw, respectively. The Window Manager port must be made current port. Next, you set the clipping area to the size of the Desktop. (Don’t forget to get the
current clipping area before setting it so you can restore it to the original value.) To
create a picture of the image, call OpenPicture() (or OpenCPicture()). Then
call CopyBits() to copy the contents of the pixmap of the Window Manager port
(the portBits field) to itself, specifying the area of the screen (the portRect field)
as the source and destination area. Stop recording the picture with ClosePicture()
and restore the original clipping region and port.

The following example creates a picture of the Desktop on the main screen. You
could copy a portion of the screen instead by specifying its rectangle in the srcRect
and destRect fields of CopyBits(). Or, if you wanted to manipulate the image
of the Desktop before creating a picture from it, you could create a GWorld and
CopyBits() to the Desktop first.

```c
//******************************************************************************
PicHandle GetDesktopPict( )
{
GrafPtr wMgrPort, oldPort;
PicHandle hPict;
RgnHandle curClip;

GetDeskTopGrafPort( &wMgrPort );

GetPort( &oldPort );
SetPort( wMgrPort );
curClip = NewRgn();
GetClip( curClip );
ClipRect( &wMgrPort->portRect );

// start recording
hPict = OpenPicture( &wMgrPort->portRect );

CopyBits( &wMgrPort->portBits, &wMgrPort->portBits,
&wMgrPort->portRect, &wMgrPort->portRect, srcCopy,
OL );

ClosePicture(); // stop recording
SetClip( curClip );
DisposeRgn( curClip );
SetPort(oldPort);

return( hPict );
}

/******************************************************************************
```
void GetDeskTopGrafPort( GrafPtr *wMgrPort )
{
    OSErr err;
    long response;

    err = Gestalt( gestaltQuickdrawVersion, &response);
    if ( (err == noErr) && (response >= gestalt8BitQD) )
        GetCWMgrPort((CGrafPtr *)wMgrPort);
    else
        GetWMgrPort(wMgrPort);
}

This next sample copies the image of the desktop to a GWorld, and then
copies it from the Gworld to the current window. This code uses the
GetDeskTopGrafPort() routine from the sample above.

raries

#define mTopLeft(r)       (*((Point *)&(r))[0])
#define mBotRight(r)      (*((Point *)&(r))[1])

void GWorldScreenCapture( )
{
    RGBColor blackRGB = {0, 0, 0};
    RGBColor whiteRGB = {65535, 65535, 65535};
    CGrafPtr curPort;
    GDHandle curDevice;
    Rect winRect;
    PixMapHandle offPix;
    GrafPtr wMgrPort;
    GWorldPtr gOurGWorld = nil;

    GetGWorld(&curPort, &curDevice);

    // get rectangle defining current port's interior,
    // translate it into global coordinates.
    winRect = curPort->portRect;
    LocalToGlobal(&mTopLeft(winRect));
    LocalToGlobal(&mBotRight(winRect));

    NewGWorld( &gOurGWorld, 0, &winRect, nil, curDevice, 0);
    if ( !gOurGWorld )
        return;
SetGWorld(curPort, curDevice);

// copy desktop image to GWorld
offPix = GetGWorldPixMap(gOurGWorld);

if(LockPixels(offPix))
{
    SetGWorld(gOurGWorld, NULL);
    GetDeskTopGrafPort(&wMgrPort);
    CopyBits(&wMgrPort->portBits,
             (BitMap *)offPix, &wMgrPort->portRect, 
             &curPort->portRect, srcCopy, nil);
    UnlockPixels(offPix);
}

// blit offscreen desktop image to onscreen
if(LockPixels(offPix))
{
    RGBForeColor(&blackRGB);
    RGBBackColor(&whiteRGB);

    CopyBits((BitMap *)offPix, 
             ((GrafPtr)curPort)->portBits, 
             &curPort->portRect, &curPort->portRect, srcCopy, nil);
    UnlockPixels(offPix);
}

// wait for button press to admire your work
while (!Button());

DisposeGWorld(gOurGWorld);

Figure 12-2 shows an example of a window with a screen image in it.
Figure 12-2: The window in the center of the screen in this figure contains a snapshot of the desktop.

Here's a little bonus for you that Pete, my technical reviewer, thought was super cool. Have you ever seen a picture of a picture of a picture? ... Well, what would happen if we copy the contents of the screen to a window on the screen using the following routine? (Have fun . . .)

```c
void CopyScreenToWindow( WindowPtr myWindow)
{
    GrafPtr wMgrPort;

    GetDeskTopGrafPort( &wMgrPort);
    CopyBits( &wMgrPort->portBits, &myWindow->portBits,
              &wMgrPort->portRect,
              &myWindow->portRect, srcCopy, nil);
}
```
And as a final side note about creating screen images, now that we’ve done all this work: Apple provides a nifty little built-in screen capture. Just press Command-Shift-3. This produces a picture of the screen image and stores it on the top level of the main volume.

**Related FAQs**

- See also FAQ 23-38, “Is there a way to draw outside a window on the desktop?”
- See also FAQ 33-17, “How do I get the GrafPort of the Desktop?”

**Related Topics**

- See also Chapter 23, “QuickDraw: Drawing.”
- See also Chapter 33, “Windows.”

---

**How can I set the pixel depth and color mode of a monitor?**

`SetDepth()` is used set the pixel depth of a monitor and set the monitor to color mode or black-and-white mode. It takes four parameters:

- `gdh` — the handle of the graphics device whose pixels are to be set
- `depth` — the pixel depth of the monitor
- `whichFlags` — the graphics device type (color or black-and-white) that will be set by this call
- `flags` — the device type value to switch to.

`SetDepth()` always sets the pixel depth of the device to the value specified by the depth parameter. Whether or not it affects the mode of the graphics device depends upon the value stored in `whichFlags` (the third parameter). When `whichFlags` is set to 1, the graphics device is set to the color mode specified in the flags parameter: 1 for color or 0 for black-and-white. If the `whichFlags` parameter is 0, then the flags parameter is ignored and the color mode remains untouched.

Before you actually call `SetDepth()`, call `HasDepth()` with the same parameters to see if the device supports the given pixel depth and type.
The following example sets the graphics device, hGD, to 256 colors (8-bit pixel depth) and the device type to color mode:

```c
if ( HasDepth(hGD, 8, 1, 1) )
    SetDepth( hGD, 8, 1, 1 );
```

The following example sets the graphics device, hGD, to 16 colors and leaves the device type untouched:

```c
if ( HasDepth(hGD, 4, 0, 1) )
    SetDepth( hGD, 4, 0, 1 );
```

Related FAQ

See also FAQ 12-14, “Why does SetDepth() from B/W produce grays instead of colors?”

**How can I set a monitor’s setting from gray to color?**

See FAQ 12-12, “How can I set the pixel depth and color mode of a monitor?”

**Why does SetDepth() from B/W produce grays instead of colors?**

Make sure that the last two parameters in SetDepth(), whichFlags and flags, are set properly. When the whichFlags parameter is set to 1, the graphics device is set to the color mode specified in the flags parameter: 1 for color or 0 for black-and-white. If the whichFlags parameter is 0, then the flags parameter is ignored and the color mode remains untouched.

Related FAQ

See also FAQ 12-12, “How can I set the pixel depth and color mode of a monitor?”

**Working with Graphic Devices**

This section answers FAQs about updating windows that exist on multiple screens.
If my window overlaps multiple screens with different depths, how can I draw the contents of the window properly on all screens?

You can use DeviceLoop() to have an application-defined drawing procedure called for each monitor that intersects your window's drawing region, as follows:

```
DeviceLoop( drawingRgn, drawingProc, userData, flags );
```

The first parameter is the region of the window that is to be redrawn. The second parameter is a pointer to your drawing procedure. The third parameter contains user-defined data that is passed directly to the drawing procedure and usually holds a pointer to the window to be redrawn. The flags field defines how devices with the same display characteristics are handled.

The drawing procedure called by DeviceLoop() is defined as follows:

```
pascal void myDrawingProc( short screenDepth, short deviceFlags, GDHandle hDevice, long userData )
```

The screenDepth parameter holds the pixel size of the target device. deviceFlags contains the gdFlags of the target device's graphic device record. hDevice is the target device.

Each time your drawing procedure is called, the current port's visRgn will have been set to the intersection of the original port's visRgn and the intersecting portion of the target device — so you can just draw.

Related Topics

- See also Chapter 23, "QuickDraw: Drawing."
- See also Chapter 33, "Windows."

How can I determine which screens my window overlaps and which screen the majority of my window exists on?

You can iterate through the device list and check the intersection of the window's rectangle with the graphics device's global bounds. To get the first device in the device list, call GetDeviceList(). To get the next device in the list, call
GetNextDevice(). When GetNextDevice() returns nil, you have iterated through the entire list. For each returned device, call TestDeviceAttribute() to test if the device is a screen device and if it is active. The rectangle of the entire window can be obtained from the rgnBBox field of the windows structure region. The graphics device's global bounds are stored in the gdRect field of the device.

The following example gets the rectangle of the window and finds all graphics devices that the window intersects. The graphics device with the greatest intersection of the window is passed back to the main routine.

```c
void GetWindowDevices( WindowPtr pWin )
{
    GDHandle hDevice;
    Rect rWindow;

    rWindow = (**((WindowPeek)pWin)->strucRgn)->rgnBBox);
    hDevice = GetIntersectingDevices(&rWindow);
}

GDHandle GetIntersectingDevices(Rect *testRect)
{
    long area, maxArea;
    GDHandle hDevice, hMaxIntersectingDevice;
    Rect intersection;

    maxArea = 0;

    for (hDevice = GetDeviceList(); hDevice != nil; hDevice =
        GetNextDevice(hDevice))
    {
        HLock( (Handle)hDevice );
        // test for active monitors that intersect the rect
        if (((TestDeviceAttribute(hDevice, screenDevice)
            && TestDeviceAttribute(hDevice, screenActive)
            && SectRect(testRect, &intersection))
        {
            area = ((long)(intersection.right -
                intersection.left)) *
(((long)(intersection.bottom - intersection.top));
if (area > maxArea)
{
    hMaxIntersectingDevice = hDevice;
    maxArea = area;
}
}
HUnlock((Handle)hDevice);
return( hMaxIntersectingDevice );

Related Topic

See also Chapter 33, "Windows."

Topic-Related FAQs

See also FAQ 22-12, "How do I determine the slot number of a screen?"
See also FAQ 24-10, "How do I create a picture of the contents of a window?"
Icons

Your application can make use of icons in many different ways. Not only does it use an icon to represent itself in the Finder, but your application can also use icons in windows, dialogs, and menus (both as menu titles and to accompany menu items). In this chapter, you'll learn about icon families, icon suites, and how to draw icons so that they have the same appearance as your application's Finder icon. You'll also learn about color icons, how you can speed up the drawing of 'cicn' icons, and how you can create icon masks for icons of different depths.

What are an icon family and icon suite?

An icon family is a collection of different icon resources that share the same resource ID and represent different size and bit depths. There are two sizes of icons and three bit depths. The resource icon types of an icon family are as follows:

- 'ICN#' = 32 x 32 pixel 1-bit depth
- 'icl4' = 32 x 32 pixel 4-bit depth
- 'icl8' = 32 x 32 pixel 8-bit depth
- 'ics#' = 16 x 16 pixel 1-bit depth
An icon family does not have to contain all of the possible types. If a requested type is not available, then the routines that use icon families will select the best available match.

An icon suite is a collection of handles to a subset of icons from an icon family.

**Which icon types can I put in dialogs and menus?**

The Menu Manager allows you to display icons of resource type ‘ICON’, ‘cicn’, and ‘SICN’. To display other types of icons, you will have to write your own MDEF.

The Dialog Manager allow you to display icons of resource type ‘ICON’ and ‘cicn’. If you want to display other types of icons, define a user item and draw the icon with a draw procedure.

**Related FAQ**

- See also FAQ 16-7, “How do I put an icon in my menu next to the menu item text like the Apple menu?”

**Related Topics**

- See also Chapter 6, “Dialogs and Controls.”
- See also Chapter 16, “Menus.”

**What types of icons can the Finder display?**

The Finder can only display icons in an icon family. It does not display resources of type ‘ICON’, ‘cicn’, or ‘SICN’.

**Related Topic**

- See also Chapter 10, “Finder and Desktop.”
How can I draw an icon as it appears in the Finder in the different Finder states?

`PlotIconID()`, `PlotIconMethod()`, `PlotIconHandle()`, and `PlotCIconHandle()` take a transform parameter. The transform parameter allows you to transform the icon in a manner analogous to certain Finder states for icons. The valid transform values are as follows:

- `kTransformNone = 0x0`
- `kTransformDisabled = 0x1`
- `kTransformoffline = 0x2`
- `kTransformOpen = 0x3`
- `kTransformSelected = 0x4000`
- `kTransformSelectedDisabled = kTransformSelected | kTransformDisabled`
- `kTransformSelectedoffline = kTransformSelected | kTransformoffline`
- `kTransformSelectedOpen = kTransformSelected | kTransformOpen`

Another group of constants allow you to assign a color to an icon that simulates the Finder label colors. They are:

- `kTransformLabel1 = 0x0100`
- `kTransformLabel2 = 0x0200`
- `kTransformLabel3 = 0x0300`
- `kTransformLabel4 = 0x0400`
- `kTransformLabel5 = 0x0500`
- `kTransformLabel6 = 0x0600`
- `kTransformLabel7 = 0x0700`
To determine the appropriate label for a file's icon, you can check bits 1 through 3 of the fflags field in the file's file information record. A value of 0 indicates no label.

Related FAQ

See also FAQ 8-42, "How can I get the Finder label menu color and label string of a file?"

Related Topic

See also Chapter 10, "Finder and Desktop."

Why does only a part of my icon draw?

It may be a problem with your icon's mask. Make sure the mask completely covers the area occupied by your icon.

How can I create different masks for the icons in an icon family that are the same size but of different bit depths?

You can't. The 1-bit-per-pixel icon of each size contains the mask data for all icons of that size.

Why doesn't PlotCIcon() get recorded as part of a picture by OpenPicture()/ClosePicture()?

PlotCIcon() uses the CopyMask() procedure, which bypasses the QuickDraw bottleneck procedures. If you want to include an icon in a picture, draw the icon with the PlotCIconHandle() call.

Related Topic

See also Chapter 24, "QuickDraw: Pictures."
How can I speed up displaying a ‘cicn’ icon for animation (PlotCIcon() is too slow)?

Don’t draw your ‘cicn’ resources with PlotCIcon() each time. Load them into an off-screen GWorld and use CopyBits() to draw onto the screen.

Topic-Related FAQs

- See also FAQ 10-8, “How do I attach an icon to an application?”
- See also FAQ 10-9, “How do I get the icon of a file with a given type and creator?”
- See also FAQ 10-10, “Why is my file displaying the standard icon and not my bundled icon?”
- See also FAQ 16-7, “How do I put an icon in my menu next to the menu item text like the Apple menu?”
- See also FAQ 16-26, “How do I create a menu that has an icon as its title?”
The List Manager provides an easy way to create and manage a scrolling list of pictures, icons, and/or text. In this chapter, you'll learn about LDEFs, custom list management procedures you write that get called by the List Manager. Using an LDEF, you can customize your lists, allowing them to display more than just text. You'll also learn how to add lists to a dialog window, how to add a click loop to a list, and a host of other list-related tips and tricks.

There are five subsections in this chapter:

- Definitions
- Working with Lists
- Displaying Lists
- List Information
- Click Loops
Definitions

This section answers FAQs about two important terms that are discussed in this chapter: LDEF and click loop. Both the LDEF and the click loop are used for customized list processing.

What is an LDEF?

An LDEF, or list definition procedure, is a code resource that defines the appearance of a list. The default list definition procedure only supports the display of unstyled text. If you want to display a list of icons or pictures, or display different types of data in the same list, you can write your own customized list definition procedure.

What is a list’s click loop?

A click loop routine in a list is a callback routine that gets called repeatedly when the list is processing a click and drag with the LClick() routine. This allows you to have actions performed while the mouse button is being pressed. The click loop routine is stored in the list’s ClickLoop field. It is not executed at interrupt time and can therefore access the application’s global variables and call routines that move memory.

Related FAQs

- See also FAQ 14-19, “How do I install a click loop in my list?”
- See also FAQ 14-20, “Why won’t my list’s click loop work?”
- See also FAQ 14-21, “How can I update a text field to display information about a cell in a list when the mouse drags over it during LClick()?”

Working with Lists

This section answers FAQs about the basic operation of lists. It discusses detecting double-clicks in a list and presents solutions to a few common problems.
How do I detect a double-click on a cell?

The function LClick(), which is used to process all mouse tracking and clicking in a list, returns true if the user double-clicked in the same cell.

Related Topic

See also Chapter 7, "Events."

Why do the strings in my list have unrecognizable characters in front of them?

If you store data in a cell using LSetCell() or LAddToCell(), you must set the dataPtr parameter to point to the text data only, and not include the length byte of a Pascal string. The following example sets the cell, theCell, to the data stored in the Pascal string listData:

```pascal
Str255 listData;
LAddToCell( &(listData[1]), listData[0], theCell, theList );
```

Is there a limit to the amount of data a list can contain?

Lists cannot contain more than 32K of data. You can get around this limit by writing your own LDEF.

I make a series of changes to a list, it flashes. How can I prevent this?

Before making a series of changes to a list, disable the automatic drawing mode for the list by calling LSetDrawingMode() with the first parameter set to false. After all of the changes are made, enable the automatic drawing mode for the list by calling LSetDrawingMode() with the first parameter set to true. To make the changes visible, force the list to be redrawn by calling InvalRect(), passing the display rectangle of the list as a parameter.
The following code shows how to disable the automatic drawing mode, set the first ten items in the list to the string passed to the function, and then reenable the drawing mode and force the list to redraw:

```c
//*******************
void SetCellsToString( ListHandle hList,
                      Str255 newString )
{
    Cell theCell;
    short row;

    LSetDrawingMode( FALSE, hList);
    // make changes to the list

    for ( row = 0; row < 10; row++ )
    {
        SetPt( &theCell, 0, row);
        LSetCell( &(newString[1]), newString[0], theCell,
                  hList );
    }

    LSetDrawingMode(TRUE, hList);
    HLock( hList );
    InvalRect( (*hList)->rView );
    HUnlock( hList );
}
```

How can I have my list prevent the selection of multiple cells?

By default, the List Manager allows the selection of a range of cells or non-contiguous ranges of cells with the Shift and Command keys. You can change this cell highlighting behavior by modifying the 8-bit selFlags field of a list. When a list is created, the List Manager clears all of the bits in this field. The bit values of selFlags and the flags used to set them are listed in Table 14-1. These flags can be used in combination by adding their values together.
Table 14-1:
Selection Flags

<table>
<thead>
<tr>
<th>Bit Value</th>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NoNilHilite</td>
<td>Disables highlighting of empty cells</td>
</tr>
<tr>
<td>2</td>
<td>UseSense</td>
<td>Allows the use of the Shift key to deselect items</td>
</tr>
<tr>
<td>3</td>
<td>NoRect</td>
<td>Does not draw a Shift-drag selection as a rectangle</td>
</tr>
<tr>
<td>4</td>
<td>NoExtend</td>
<td>Deselects all items before responding to a Shift-click</td>
</tr>
<tr>
<td>5</td>
<td>NoDisjoint</td>
<td>Does not allow non-contiguous selections using the Command key</td>
</tr>
<tr>
<td>6</td>
<td>ExtendDrag</td>
<td>Enables multiple selections by dragging without the Shift key</td>
</tr>
<tr>
<td>7</td>
<td>OnlyOne</td>
<td>Allows only one item to be selected at a time</td>
</tr>
</tbody>
</table>

Use the following code to set the list selection so only one cell can be selected at a time:

(*hList)->selFlags = OnlyOne;

Why won't my list draw or reflect changes?

Try calling LDoDraw(TRUE, theList) to turn on list drawing. You can turn the drawing mode on and off to prevent the updating of the list on the screen while making modifications to it. List drawing can be turned off by specifying false in the drawIt parameter (the sixth parameter) of LNew() or by calling LDoDraw(FALSE, theList).

How do I put a list in a Macintosh modal dialog box?

The Control Manager does not define a control for lists. To put a list in a dialog, you define a user item and pass a draw procedure to the modal dialog for the list. You must also write an event filter procedure to handle events for the list.

The following example creates a dialog and a list and displays the dialog in the list. The dialog contains an OK button, a Cancel button, and a user item. The user item is assigned the draw procedure, DrawMyList(), that draws the list and the frame
Macworld Mac Programming FAQs

around it. ModalDialog() is called with the filter procedure,
MyListProcedure(), that tests for mouse-down events in the list.

***
Handle hList;

void PutListInDlog( )
{
    DialogPtr myDialog = nil;
    Rect listBounds = { 0, 0, 0, 1 };
    Cell theCell = { 0, 0 };
    short hitItem = 0;
    Rect tempRect;
    short tempItem;
    Handle tempHandle;
    ModalFilterUPP myModalFilterProc;

    myModalFilterProc = NewModalFilterProc(MyListFilter);
    myDialog = GetNewDialog(128, nil, (WindowPtr)-1);
    SetPort(myDialog);

    // Get size of user item and assign the list updating
    // routine as the draw procedure
    GetDItem(myDialog, 3, &tempItem, &tempHandle, &tempRect);
    SetDItem(myDialog, 3, tempItem,
             (Handle)NewUserItemProc(DrawMyList),
             &tempRect);

    // allow room for scrollbar
    tempRect.right -= 16;

    hList = LNew(&tempRect, &listBounds, theCell, nil,
                 myDialog, true, false, false, true);

    // fill list with your dat
    FillListWithData( );

    // loop until ModalDialog is done
    do
    {
        ModalDialog(myModalFilterProc, &hitItem);
    }
    while (hitItem != ok && hitItem != cancel);
DisposeDialog(myDialog);

//********************
void pascal DrawMyList( DialogPtr myDialog,
    short itemNum )
{
    Rect tempR;

    // Update the list.
    LUpdate(myDialog->visRgn, hList);

    // frame the list
    tempR = (*hList)->rView;
    InsetRect( &tempR, -1, -1 );
    FrameRect( &tempR );
}

//********************
pascal Boolean MylistFilter( DialogPtr theDialog,
    EventRecord *theEvent,
    short *itemHit )
{
    WindowPtr oldPort;
    Rect tempR;
    Point thePt;
    Boolean returnValue = false, isDoubleClick;

    // get the current port, set the port to this dialog
    GetPort(&oldPort);
    SetPort(theDialog);

    if (theEvent->what == mouseDown)
    {
        GetMouse(&thePt);

        // get the list rectangle
        tempR = (*hList)->rView;

        // add the scroll bar back in for hit testing
        tempR.right += 16;
    }
// See if the list was clicked in
if (PtInRect(thePt, &tempR))
{
    isDoubleClick = LClick(thePt, nil, hList);
    if (isDoubleClick)
        *itemHit = 1;
    else
        *itemHit = kListItem;
    returnValue = true;
}
}

SetPort(oldPort);
return(returnValue);

Related Topic

See also Chapter 6, “Dialogs and Controls.”

Displaying Lists

This section answers FAQs about the appearance of lists. It describes how to draw a border around a list and how to change the size and style of text in a list. It also discusses using an ‘LDEF’ to display a list of items other than text.

Can different columns in a list have different widths?

All the cells in a list must be the same size. The ListRec structure stores the size of the cells in the list as a single point in the cellSize field. You even have this restriction if you write your own list definition (LDEF) procedure. You can create the appearance of a single two-dimensional list with columns of different widths by creating a separate list for each column or by creating an LDEF that supports one column and draws a line through each cell, simulating a cell dividing line.
Can I have a list of more than one type of data (such as strings and icons)?

You must use a customized list definition procedure.

When I create a list, it has no border around it. Is there a way to make a border come up automatically, and if not, how do I draw one?

You have to draw the border around a list box yourself. To draw the border, use the following routine to get the bounding rectangle of the list from the rView field in the list structure, increase the rectangle by a pixel in each direction with the InsetRect() call (which prevents LClick() from erasing your frame), and draw the rectangle with FrameRect():

```c
//****************************
void DrawListBorder( ListHandle myListHandle )
{
    ListPtr myListPtr;
    Rect listBorder;

    HLock( (Handle)myListHandle );
    myListPtr = *myListHandle;

    PenSize(1,1);
    listBorder = myListPtr->rView;
    InsetRect( &listBorder, -1, -1);
    FrameRect(&listBorder);

    HUnlock( (Handle)myListHandle );
}
```

How can I change the size and style of text in my lists?

By default, the list definition procedure only supports the drawing of unstyled text in a list. To put styled text in a list, you should write a customized LDEF.
The list draws its text in the font, size, and style of the current graphics port. You can change the values of these font characteristics in the current graphics port and have the list draw all of its items with them. In order to get the strings to draw properly in each cell, you may have to adjust the size of the cells in the list in the cellSize field and the horizontal and vertical offsets of the text in the indent field.

You can set the style of the text in individual cells by replacing the standard low-level text drawing procedure using the SetStdProcs() call. Each time the text in a cell is drawn, your text drawing procedure will be called. You can determine which cell, if any, is being drawn by getting the location of the pen of the graphics port and calling LRect() and PtInRect(). If the pen location is in a cell, then that is the cell about to be drawn. Change the text to the desired style and then call the default standard drawing routine to draw the text.

Related FAQs

See also FAQ 6-28, “How do you deactivate (gray-out) static text items and edit text items in a dialog box?”

See also FAQ 23-3, “What are the QuickDraw ‘bottleneck’ functions?”

See also FAQ 23-39, “How can I replace the standard QuickDraw low-level or ‘bottleneck’ functions?”

See also FAQ 24-16, “How can my application process picture comments when drawing a picture?”

**How do I make a list of icons or pictures?**

The default list definition procedure only allows text to be displayed in a list. To display icons or pictures in a list, you must write your own list definition procedure.

**List Information**

This section answers FAQs about obtaining various pieces of information about lists. It discusses how to determine the pixel size of the list’s display area and the number of rows and columns that make up a list. It also describes how to determine which cells are selected and more.
How do I get the size of the current display area of a list?

The size of a list can be determined by the rView field of the ListRec structure. rView represents the visible rectangle of the list in coordinates local to the graphics port in which it exists. This rectangle does not include the height of a horizontal scroll bar or the width of a vertical scroll bar that may be attached to the list (both of which are 15 pixels).

How can I test if a cell is selected, and how do I get all of the currently selected cells?

You can test if a single cell is selected by calling the following function with the advanceIt parameter set to false:

```c
LG etSelect( Boolean advanceIt, Cell *theCell, ListHandle hList)
```

LGetSelect() will return true if the cell defined by theCell is selected.

LGetSelect() can also be used to return the first selected cell that is equal to or greater than the cell defined by theCell. If theCell is selected, then LGetSelect() returns true. If it is not selected, then LGetSelect() traverses through the list until it finds the next cell that is selected. If it finds one, it returns true and passes the cell back in the theCell parameter. If no selected cell is found, then LGetSelect() returns false. When a selected cell is found, LGetSelect() must be passed the next cell in the list.

LGetSelect() checks all of the cells in the row and then moves on to the next row.

The following sample routine finds all of the selected cells in hList. It starts at cell (0, 0) and traverses the entire list, getting all selected cells.

```c
//***************
short CheckForAllSelectedCells()
{
    short numSelected = 0;
    Cell theCell = {0, 0};

    while ( LGetSelect( TRUE, &theCell, hList ) )
    {
        // do your processing on selected cell
```
MySelectedCellFunction( theCell);

// advance to next cell
LNextCell( TRUE, TRUE, &theCell, hList );

numSelected++;
}

return( numSelected );

How can I tell which cells in my list are visible?

The cell that are visible in the list's visible rectangle, rView, are stored in the visible field of the list structure. The values in the right and bottom fields of the visible field are one greater than the dimension of the last coordinate. For example, if the first 5 columns and first 10 rows of a list are visible, the visible field is set to (0, 0, 5, 10), while the last visible coordinate is (4, 9).

How can I tell how many rows and columns are in a list?

The cell dimensions of a list are stored in the dataBounds field of the list structure. The top and left fields are set to 0, and the bottom and right fields are set to the number of rows and columns, respectively. The values in these two fields are one greater than the dimension of the last coordinate. For example, if a list has 5 columns and 10 rows, the dataBounds field is (0, 0, 5, 10), while the coordinate of the last cell is (4, 9).

Click Loops

This section answers FAQs about the implementation of click loops in lists. It describes how to install a click loop and how to use it to update a text field to display information about the currently selected list.
How do I install a click loop in my list?

You install a click loop routine into a list by manually assigning a pointer to it in the lClickLoop field of the list structure. A universal procedure pointer for the click loop routine is returned by NewListClickLoopProc(). The click loop routine should be defined as follows:

```pascal
Boolean MyClickLoop()
```

The click loop is not called at interrupt time and can therefore access application globals and call routines that move memory without any extra work on your part.

The following sample routine creates a list with 1 column and 100 rows and assigns the MyClickLoop() procedure as its click loop. MyClickLoop() gets the current mouse position each time it is called by examining the contents of the mouseLoc field of the list. If the cursor is in the list cell (1,0), then it beeps.

```
// ****************
#define kScrollBarWidth 15

ListHandle hList;

pascal Boolean MyClickLoop();

void CreateList( void )
{
  Rect myRect = {20, 20, 200, 200};
  Rect myDataBounds = {0, 0, 100, 1};
  Point myCellSize = {0, 0};

  // adjust the rectangle to leave room for the scroll bar
  myRect.right = myRect.right - kScrollBarWidth;

  hList = LNew(&myRect, &myDataBounds, myCellSize, 0,
               pWindow, TRUE, FALSE, FALSE, TRUE);

  // assign click loop to list
  (*hList)->lClickLoop = NewListClickLoopProc(MyClickLoop);

  // put data in the list
  FillList();
}
```
pascal Boolean MyClickLoop()
{
    Point mousePt;
    Cell theCell = {1, 0};
    Rect cellRect;

    mousePt = (*hList)->mouseLoc;
    LRect(&cellRect, theCell, hList);
    if ( PtInRect(mousePt, &cellRect) )
        SysBeep( 20);

    return TRUE;
}

Related FAQs

- See also FAQ 14-2, "What is a list's click loop?"
- See also FAQ 14-20, "Why won't my list's click loop work?"
- See also FAQ 14-21, "How can I update a text field to display information about a cell in a list when the mouse drags over it during LClick()?"

Why won't my list's click loop work?

The universal headers define the click loop function of a list as procedure returning a Boolean value. Inside Macintosh defines the click loop as a procedure not returning any value. The click loop must be defined as follows and it must return a value of true:

pascal Boolean MyClickLoop()

Related FAQs

- See also FAQ 14-2, "What is a list's click loop?"
- See also FAQ 14-19, "How do I install a click loop in my list?"
- See also FAQ 14-21, "How can I update a text field to display information about a cell in a list when the mouse drags over it during LClick()?"
How can I update a text field to display information about a cell in a list when the mouse drags over it during LClick()?

You can install a click loop routine. You can determine which cell the cursor is currently over by getting the current mouse location (in local coordinates) from the mouseLoc field of the list and using LRect() and PtInRect(). Once you have determined the cell, you can update the text field from within the click loop routine since it is not called at interrupt time.

Related FAQs

- See also FAQ 14-2, “What is a list’s click loop?”
- See also FAQ 14-19, “How do I install a click loop in my list?”
- See also FAQ 14-20, “Why won’t my list’s click loop work?”
Proper memory management is a vital concern to any application developer. In this chapter, you'll learn the difference between the System and application heaps and how to work with both. You'll learn how to work with temporary memory and the advantages temporary memory can offer. You'll also learn what to do when you just can't get enough memory, along with some tips for working with memory more effectively (like creating a block of memory that is initialized to all zeros and the best way to copy data from one handle to another).

There are five subsections in this chapter:

- Definitions
- Handles and Pointers
- Using Memory
- The A5 World
- Temporary Memory
Definitions

This section answers FAQs about some important terms that are used to describe the memory environment of the Macintosh, including how a Mac's memory is organized and the different elements available to your program.

What is a handle?

A handle is a pointer to a master pointer, which points to a block of relocatable memory (see Figure 15-1). The master pointer keeps track of the location of a relocatable block as it gets moved within the heap. This way, the system can move allocated blocks of memory around to optimize the available memory by preventing fragmentation. Many Toolbox functions take handles to specific types of data as parameters. You can also directly access the data referenced by a handle by dereferencing it. One level of indirection from a handle gives you a pointer to the data. Two levels of indirection from the handle give you the data in the block of memory.

The optimization provided by handles does have its pitfalls. A handle can be moved by the system at any time. If your program dereferences a handle, stores the pointer value, and accesses the handle's data through this pointer, it is possible that the pointer could become invalid if its data is relocated. To avoid this problem, you can lock a handle by calling HLock(), which prevents the memory block from moving. Make sure you unlock the handle with HUnlock() when you are done.
A handle is created by the Memory Manager. You cannot take a pointer that you created and pass its address to a function that requires a handle.

**What is temporary memory?**

Temporary memory is unused memory outside an application heap that can be allocated and temporarily used by an application.

**What is the System heap?**

The System heap is a block of memory used by the system for its own memory allocation. If you want your application to allocate memory that remains after your application quits, then you must allocate the memory in the System heap.

**What is the A5 world?**

The A5 world, shown in Figure 15-2, is an area of memory in a 68K Macintosh application’s partition that contains the application’s global variables, parameters, jump table, and QuickDraw global variables.

![Figure 15-2: The components of the A5 world in a 68K Macintosh application](image)

The Operating System uses the A5 register to point to the boundary between an application’s parameters and its global variables. This value is also stored in the System global variable, CurrentA5. The application’s own global variables are found as negative offsets from the value of CurrentA5.
Handles and Pointers

This section answers FAQs about using handles and pointers. It discusses when, why, and how to lock handles. It describes how to allocate blocks of zeroed-out memory and more.

**FAQ 15-5**

If I lock a locked handle and then call HUnlock(), will the handle be unlocked, or do I need to balance HLocks() with HUnlocks()?

Calling HLock() on a locked handle does nothing, so calling a HUnlock() will unlock a handle that has been locked multiple times with HLock().

**FAQ 15-6**

Can I just pass a pointer to a memory block pointer where I need a handle?

No. Passing the address of a pointer is not the same as passing a handle.

Related FAQ

See also FAQ 15-1, "What is a handle?"
**How can I tell whether a handle was previously locked before I locked it, so I know whether or not to unlock it when I’m done with it?**

Before you lock a handle, call `HGetState()` to get the current value of the relocatable block’s tag byte. This will tell you whether or not the handle was locked. When you are done with the handle, call `HSetState()` to return the handle to the state it was in before you locked it. If it was unlocked, `HSetState()` will unlock it. If it was locked `HSetState()` will leave it locked. Use the following routine:

```c
/***************
char handleState;
handleState = HGetState(myHandle);
HLock( myHandle);
// perform your actions
HSetState( myHandle, handleState );
```

**How can I tell if a purgeable block is still valid?**

If a handle’s block has been purged, its master pointer is set to nil.

**When should I lock a handle?**

You only need to lock a handle if there is a chance that, after it is dereferenced, it may move while you are accessing the data in its block. This could happen if you dereference the handle to get a pointer to the data and then call a routine that moves memory. When memory is allocated, your handle may move in memory making your pointer point to an invalid block of memory. To avoid this problem, you should call `HLock()` to lock the handle. When you are done, call `HUnlock()` to unlock it.

You should avoid unnecessarily locking handles because this will slow down the Memory Manager.
Can I change the size of my handle?

The function SetHandleSize(theHandle, newSize) will shrink or expand the handle (theHandle) to a size of newSize. Make sure that the handle is not locked. In order to expand a relocatable memory block, the entire block may need to be moved in memory. If the handle has been locked with HLock(), then the operation will fail.

What is the best way to copy data from one handle to another handle?

HandToHand() copies the contents of the passed-in handle to a newly created handle and passes back the new handle, as follows:

```c
//*************
void DoCopyHandle()
{
    OSErr err;
    Rect rPict = {50, 50, 150, 150};
    PicHandle hPict, hPictCopy;
    hPict = GetPicture( PICT_ID );
    hPictCopy = hPict;
    err = HandToHand( &Handle)hPictCopy );
    if (!err)
        DrawPicture( hPictCopy, &rPict);
}
```

Why should I use handles instead of pointers?

Handles should be used instead of pointers when possible to avoid heap fragmentation (see Figure 15-4). A handle references a relocatable block of memory. This block can move within its heap to allow the remaining available memory to form a large contiguous block as opposed to two smaller ones. This allows new blocks to be allocated that may otherwise not have enough memory. Pointers created with
NewPtr() point to non-relocatable blocks of memory. The allocated memory cannot be repositioned in the heap to allow the individual blocks of available memory to form larger contiguous blocks.

**Figure 15-4:** Using handles to reference moveable blocks of memory prevents heap fragmentation. Figure 15-4(a) shows fragmented memory. Figure 15-4(b) shows memory after it has been compacted. The free memory is now in a single block, allowing for a larger block of memory to be allocated contiguously.

**How can I get a handle to a block of zeroed-out memory?**

NewHandleClear() returns a handle to a relocatable block of memory that has been zeroed out. NewPtrClear() returns a pointer to a non-relocatable block of memory that has been zeroed out.

**How can I get the size of the data referenced by a handle or a pointer?**

To get the size of the data referenced by a handle, call GetHandleSize(). To get the size of the data pointed to by a pointer, call GetPointerSize().
Using Memory

This section answers FAQs about an application’s use of memory. It describes the steps necessary to allocate memory in the System heap and how to determine how much memory is available for your application’s use.

How can I tell how much memory is available to my application?

The function FreeMem() returns the amount of free space currently in the heap zone.

PurgeSpace() returns the total number of bytes and the size of the largest contiguous block of memory that would be available in the application heap if all purgeable blocks were purged by the call PurgeMem().

MaxBlock() returns the size of the largest block of memory that would be available if heap compaction were performed by a call to CompactMem(). MaxBlockSys() gets this information for the System heap zone.

How do I allocate memory in the System heap instead of the application heap?

To allocate a relocatable block of memory in the System heap zone, you can call NewHandleSys(). NewPtrSys() allocates a non-relocatable block in the System heap. NewHandle() and NewPtr() allocate memory from the application heap of the application that called them.

Another approach is to set the current heap zone to the System zone and then use ordinary Memory Manager calls. The following sample calls set the current heap zone to the System zone returned by SystemZone(), allocate memory, and then reset the current zone to the application zone returned by ApplicationZone():

```c
SetZone( SystemZone() );

// allocate memory
SetZone( ApplicationZone() );
```
You may want to allocate memory in the System zone to store information needed by a System extension or store an interrupt task that you want to continue when the installing application is not the current application.

**When my program ends, will it free up all memory that it has allocated?**

All memory that an application allocates in the application heap is freed when the application exits. Memory allocated in System heap must be manually deallocated.

**The A5 World**

This section answers FAQs about that magical place in 68K Macintoshes called the "A5 world." It discusses how to access the information stored in the A5 world during interrupt time.

**How do I access the A5 world of my application from a function called at interrupt time?**

On 68K Macintosh computers, the location of the global variables of an application are stored in the A5 register. When the context of one process is switched with the context of another process, the A5 register and CurrentA5 are set to point to the A5 world of the new process. When a function is called during interrupt time, such as a sound callback procedure or a notification response procedure, it must have access to its application's A5 world before it can manipulate any of its application's globals.

There are two functions that help do this: SetCurrentA5() and SetA5(). SetCurrentA5() is called in the application to get the current value of A5. This value is then stored in an unused field of a structure that is passed to the code to be executed at interrupt. This code then retrieves the passed-in value of A5 and calls SetA5() to set A5 to its application's A5. SetA5() returns the current value of A5. When it is done accessing its application's globals, the interrupt function must reset A5 to its original value.
The following sample callback function is assigned to a sound channel that sets the application global variable `gSoundComplete`. The calling function stores the current value of A5 in the unused `userInfo` field of the sound channel structure.

```c
/******************
void CheckByCallBack( )
{
    mySndChan->userInfo = SetCurrentA5();
    err = SndStartFilePlay( mySndChan, 0, 200, 65536, NULL, NULL, NULL, TRUE );
}
/******************
SndCallBackUPP MyCallBackProc(SndChannelPtr mySndChan, SndCommand *pCmd )
{
    long myA5;
    myA5 = SetA5(mySndChan->userInfo);
    gSoundComplete = TRUE;
    myA5 = SetA5(myA5);
}
```

In native PowerPC applications, an application’s global variables are always accessible. Therefore, `SetupA5()` and `SetCurrentA5()` are unnecessary.

**When do I need to explicitly set up and restore the A5 world?**

First of all, you never need to set up and restore the A5 world in native PowerPC applications.

In 680x0 applications, you need to set up the A5 world to do the following:

- Allow a piece of detached code installed by an application (such as a Time Manager task or a VBL task) to access the application’s global variables or QuickDraw global variables.
Create a 680x0 context for a stand-alone code module (such as an MDEF)

What is the A4 register used for in code resources?

Code resources use the A4 register to access their global variables. The code resource must set up the A4 register before it attempts to access any globals.

In CodeWarrior the A4 register is set up like this:

```c
long oldA4;

// Set up A4, so we can access our globals.
oldA4 = SetCurrentA4();

// Do your stuff

// Restore the value of A4 on the way out.
SetA4(oldA4);
```

In Think C the A4 register is set up like this:

```c
// Set up A4, so we can access our globals.
RememberAO();
SetUpA4();

// Do your stuff

// Restore the value of A4 on the way out.
RestoreA4();
```

This A4 register set up is unnecessary in native PowerPC applications.

Related FAQ

See also FAQ 11-5, "Can my Gestalt selector function have global variables?"
Temporary Memory

This section answers FAQs about temporary memory. It describes how an application can obtain more memory after it has exhausted its supply in its own application heap.

How do I get temporary memory?

A relocatable block of temporary memory can be allocated by the call TempNewHandle(). TempHLock() and TempHUnlock() are used to lock and unlock a block of temporary memory. To release the block of temporary memory, call TempDisposeHandle().

What can I do when I need memory but can't get it?

You can call CompactMem() and PurgeMem() to shift relocatable blocks and free unlocked, relocatable blocks in your heap. If there still is not enough memory in your heap, you can try to get memory outside your heap, called temporary memory, for temporary use.

Can I access more memory than is allocated in my application heap?

An application can access available memory outside of its application heap by using temporary memory.

Topic-Related FAQs

☐ See also FAQ 17-11, “Do I have to do all of the A5 work on PowerMacs to access globals like I do on 680x0 machines?”

☐ See also FAQ 25-5, “How do I load a resource into the System heap?”
Menus

Though it is actually pretty easy to add menus to an application, there are a surprising amount of menu-related questions out there. For example, in this chapter, you'll learn to tell how many items are in a menu and whether a specific item, or the entire menu, is disabled. You'll learn the proper ellipsis etiquette (when it is proper to place '...' following a menu item). You'll also learn about MenuHooks and MBarHooks, and how to create a menu definition procedure (and MDEF). There's something here for everyone.

There are three subsections in this chapter:

- Definitions
- Using Menus
- Special Menu Processing
Definitions

This section answers FAQs about some important terms that are discussed in this chapter: hierarchical menus and hook functions. Hierarchical menus extend the basic menu interface and the two hook functions defined here are used for special programming techniques.

What is a hierarchical menu?

A hierarchical, or cascading, menu allow a programmer to attach an entire menu to a single menu item. When the user highlights a hierarchical menu item, the item’s associated submenu appears to its right. Hierarchical menu items have small, right-pointing triangles next to the item, as shown in Figure 16-1.

![Figure 16-1: A hierarchical menu](image)

Related FAQs

- See also FAQ 16-23, "Why won’t my hierarchical menus show up in my program?"
- See also FAQ 16-24, "How do I put hierarchical menus in my program?"

What are MenuHook and MBBarHook?

MenuHook and MBBarHook are procedure pointers that are called during menu operations. MBBarHook is executed by MenuSelect() before drawing the selected menu. MenuHook is called repeatedly during the MenuSelect() operation.
Using Menus

This section answers FAQs about using menus in an application. It discusses disabling and enabling menus and menu items, putting marks and icons in menu items, changing the title of a menu in the menu bar, and a host of other general menu manipulations.

How do I add menus to a program?

With a resource file editor, create an MBAR resource and add the resource IDs of the menus you want displayed in the menu bar. Create a menu resource with the appropriate menu items for each menu resource ID listed in the MBAR resource. In your program, load the menu bar and then draw it. You must have already run the standard initialization routines (InitGraf(), InitFonts(), InitWindows(), InitMenus(), and so forth) before you do this.

The following sample routine loads the menu bar with resource ID 128:

```c
//**********
void MenuBarInit( void )
{
    Handle menuBar;

    menuBar = GetNewMBar( 128 );
    SetMenuBar( menuBar );
    DrawMenuBar();
}
```

The processing of menu selections is done in the event loop. First, test for a mouseDown event. Then determine where the mouse was clicked with FindWindow(). If the part of the window returned is inMenuBar, the user clicked in the menu bar. To allow the menu to display and the user to select an item, call MenuSelect(). The return value of this function contains the menu ID in the high word and the item number in the low word. MenuSelect() returns a 0 if no item was selected. After the menu selection is processed, call HiliteMenu(0) to unhighlight the selected menu.

The following sample routine handles the user menu selection after a mouseDown event was detected. It takes the event pointer as a parameter. Note that the item number of the first item is 1 not 0.
/*************/
void HandleMouseDown( EventRecord *eventPtr )
{
  WindowPtr whichWindow;
  short thePart, menu, item;
  long menuChoice;

  thePart = FindWindow( eventPtr->where, &whichWindow );
  switch ( thePart )
  {
    case inMenuBar:
      menuChoice = MenuSelect( eventPtr->where );
      if ( menuChoice != 0 )
      {
        menu = HiWord( menuChoice );
        item = LoWord( menuChoice );

        switch ( menu )
        {
          case 128:
            switch( item )
            {
              case 1:
                // do function for first
                // item break;
              case 2:
                // do function for second
                // item break;
            }
            break;
          
          case 129:
            switch( item )
            {
              case 1:
                // do function for first
                // item break;
              case 2:
                // do function for second
                // item break;
            }
            break;
          
        } // end switch ( menu )
    } // end case inMenuBar
  } // end switch ( thePart )
} // end void HandleMouseDown
How do I put the Apple menu in my application?

The Apple menu is part of the standard set of menus that your application should display. By convention, the first item is used to display information about the application via an About box dialog. The remaining menu items list the files found inside the Apple Menu Items folder inside the System folder.

To create the Apple menu, go into your resource editor and create a new menu, as shown in Figure 16-2. (Both Resorcerer and ResEdit allow you to set the title of the menu to the Apple symbol.)

Figure 16-2: The 'MENU' resource editor from Resorcerer. You can set the title of a menu as the apple by pressing the button at the top of the dialog.
Add "About <your application>" as the first menu item and append a menu item separator line. In your program, get a handle to the Apple menu by calling GetMenuHandle(). To load the menu items into the Apple menu, call AddResMenu(), passing 'DRVR' into the second parameter.

The following routine loads the Apple menu with the files inside the Apple Menu Items folder:

```c
#define mApple 128
#define iAbout 1

void FillAppleMenu()
{
    menu = GetMHandle( mApple );
    AddResMenu( menu, 'DRVR' );
}

When the user selects an item from the Apple menu, you must first determine if your About item was selected, as shown in the following sample routine. If one of the items from the Apple Menu Items folder was selected, call GetItem() to get the name of the menu item. Then call OpenDeskAcc(), passing it the menu item's name, to open the menu item.

```c
void HandleAppleChoice( short item )
{
    MenuHandle appleMenu;
    Str255 accName;
    short accNumber;

    switch ( item )
    {
        case iAbout:
            DoAboutBox();
            break;
        default:
            appleMenu = GetMHandle( mApple );
            GetItem( appleMenu, item, accName );
            accNumber = OpenDeskAcc( accName );
            break;
    }
}
How do I add the application menu to my application's menu bar?

The application menu is automatically appended to an application's menu bar by the Menu Manager when the application includes the Apple menu.

How do I put a divider in my menu?

The divider is the dimmed line the extends across the width of the menu and is used to separate one group of menu items from another. To add a divider in your menu, add a menu item with a hyphen character ('-') as its title.

ResEdit and Resorcerer also have radio buttons that allow you to designate an item as a divider.

How do I put an icon in my menu next to the menu item text like the Apple menu?

You can add an 'ICON', 'cicn', reduced 'ICON', or 'SICN' to be displayed at the left side of a menu item with the SetItemIcon() command. The first parameter is a handle to the menu. The second parameter is the item number. The third parameter is the resource ID of the 'ICON', 'cicn', or 'SICN' minus 256. If you add an 'ICON' or a 'cicn' resource, just calling SetItemIcon() is sufficient. If you want to add a reduced 'ICON', you must call SetItemCmd() with the third parameter of 0x1D. If you want to add an 'SICN', you must call SetItemCmd() with the third parameter of 0x1E. Resorcerer allows you to attach an icon, reduced icon, or small icon to a menu item in its 'MENU' editing screen.

The following sample routine adds, in a menu with ID 129, an 'ICON' or 'cicn' of resource ID 259 to the first item, a reduced 'ICON' of ID 258 to the second menu item, and an 'SICN' to the third item:

```c
//***************
void AddMenuItemsIcons( )
{
    MenuHandle hMenu;
    long menuFlags;
    Boolean isEnabled;
```
hMenu = GetMHandle( 129 );
HLock((Handle)hMenu);

// add ICON or cicn resource of id 259
SetItemIcon( hMenu, 1, 3);

// add a reduced ICON of ID 258
SetItemCmd(hMenu, 2, 0x1D);
SetItemIcon( hMenu, 2, 2);

// add SICN of id 257
SetItemCmd(hMenu, 2, 0x1E);
SetItemIcon( hMenu, 3, 1);

HUnlock((Handle)hMenu);

Related FAQ

See also FAQ 13-2, "Which icon types can I put in dialog boxes and menus?"

Related Topic

See also Chapter 13, "Icons."

**Why can't a menu item have a small icon and a Command-key equivalent?**

The cmdChr field of a menu item is used to contain a Command-key equivalent. To attach a small icon to a menu item, the resource ID of the 'SICN' minus 256 is stored in the itemIcon field and the cmdChr field is set to 0x1E. To store a reduced icon, the cmdChr field is set to 0x1D. When an 'ICON' or 'cicn' is attached to a menu item, the cmdChr field is not used.

Figure 16-3 shows menu items with both Command-key equivalents and icons attached to them.
How do I dim or disable a menu item or an entire menu?

To dim or disable an item, you call `DisableItem()`, passing the menu ID of the item’s menu as the first parameter and the item number as the second parameter. You can disable the first 31 items of your menu. To enable the item, call `EnableItem()` with the same parameters.

To dim or disable a menu, call `DisableItem()`, passing the menu ID of the menu and a 0 as the item. This will dim the menu’s title and all of the items in the menu.

How can I tell if a menu or menu item is disabled?

You can get the enabled/disabled state of a menu and its menu items by examining the `enableFlags` field of the menu’s `MenuInfo` structure. A `MenuHandle` is a handle to a `MenuInfo` structure. The `enableFlags` field is of type “long.” The first bit, bit 0, reflects the state of the menu itself. The next bit, bit 1, reflects the state of the first menu item. Bit 2 reflects the state of the second item. The last bit reflects the state of item number 31. If the bit is set, the menu or corresponding item is enabled. If the first bit is 0, then the menu title and all of its menu items are disabled.

If you have more than 31 items, individual disabling and enabling of items can only be done from a customized MDEF that enables your program to determine the state of each item.

The following sample routine takes a menu ID and item number and returns true if the menu item is enabled and false if it is disabled:
//*************************
Boolean IsItemEnabled( short theMenu,
                      short theItem)
{
    MenuHandle  hMenu;
    long       menuFlags;
    Boolean    isEnabled;

    hMenu = GetMHandle( theMenu );
    HLock((Handle)hMenu);

    menuFlags =(*hMenu)->enableFlags;
    isEnabled = (menuFlags >> theItem) & 0x00000001;

    HUnlock((Handle)hMenu);

    return( isEnabled );
}

**When should I put ellipsis (...) at the end of a menu item?**

Put an ellipsis (...) at the end of any menu item whose actions bring up a dialog in order to complete its function. There is no trick to this — simply append the ellipsis to your menu item title.

**How do I put a check mark next to a menu item?**

To place or remove a check mark to the left of a menu item, you can call CheckItem(). CheckItem() takes three parameters. The first parameter is the handle of the menu of the item to check. The second parameter is the item ID of the item. The third parameter is true if the item is to be checked or false if the check mark is to be removed.

SetItemMark() allows you place or remove any character as the item mark. To remove a character as the item mark, pass 0 in the last parameter. Some character values of interest are listed in Table 16-1.
You can determine the current value of the item's mark by calling `GetItemMark()`.

### How many menu items can I have in a menu?

The standard MDEF can handle up to 255 items. Individual enabling and disabling is limited to the first 31 menu items. To handle more than this number, you will have to write your own MDEF.

### Why can't I individually enable or disable any menu past menu item position 31?

The enabled state of a menu item is maintained in a long field, `enableFlags`, in the menu structure. The state of the menu itself is maintained in the first bit. This leaves 31 bits to store the state of the menu items. `DisableItem()` and `EnableItem()` will allow you to individually enable and disable menu items 1 to 31, but no items past that. If you call `DisableItem()` with a 0 as the item parameter, the menu title and all of the menu items are disabled, including those past position 31. `EnableItem()` with a 0 as the item parameter will enable the menu title and all of the items. If you want to individually enable or disable items past position 31, you will have to write an MDEF for the menu.

---

**Table 16-1:**  
Menu Item Marks

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No mark</td>
<td></td>
</tr>
<tr>
<td>commandMark</td>
<td>17</td>
<td>Command mark</td>
</tr>
<tr>
<td>checkMark</td>
<td>18</td>
<td>Check mark</td>
</tr>
<tr>
<td>diamondMark</td>
<td>19</td>
<td>Small diamond</td>
</tr>
<tr>
<td>appleMark</td>
<td>20</td>
<td>Apple</td>
</tr>
</tbody>
</table>

---
How can I tell how many items are in a menu?

To count the number of items in a menu, call CountMItems(). CountMItems() takes the menu ID as a parameter and returns the number of items. This function is particularly useful for determining the number of items in menus that have been filled by AddResMenu() or InsertResMenu().

How can I dynamically create a menu with menu items that include metacharacters without having them treated as metacharacters?

Both AppendMenu() and InsertMenuItem() use metacharacters to specify additional information about a menu item that is about to be added to a menu. The valid metacharacters are ‘;’, ‘^’, ‘!’, ‘<’, ‘!’, and ‘(‘. To use any of these metacharacters in the text of a menu item, first use AppendMenu to specify at least one character as the item’s text, and then call SetMenuItemText() to set the item’s text to the desired string.

How can I change the style, size, and font of a menu?

The standard menu-definition routine draws all menu items and titles in the System font. This is normally 12-point Chicago. To use a different font or font size, you must write your own menu-definition procedure.

You can alter the style of a menu item by calling SetItemStyle(). When you add a menu item with AppendMenu() or InsertMenuItem(), you can define the menu style with the ‘<’ metacharacter followed by B, I, U, O, or S for boldface, italics, underlined, outlined, or shadowed text, respectively. If you use Resorcerer, you can specify these options in your resource file.

Related Topic

See also Chapter 31, “Text: Fonts and Drawing.”
How do I make a menu of available fonts?

To make a menu of available fonts, use AddResMenu(), specifying the ‘FONT’ resource to add all fonts available to the system. The menu to which the font names are being added must have been previously created. The fonts are appended to the menu in alphabetical order.

The following routine adds the font names to the passed-in menu:

```c
void AddFontsToMenu(MenuHandle fontMenu)
{
    AddResMenu(fontMenu,'FONT');
}
```

Related Topic

See also Chapter 31, “Text: Fonts and Drawing.”

How do I change the title of a menu in the menu bar after it has been installed?

The menu title is stored in the menuData field (the last field) of the MenuInfo structure. Even though this field is defined as a Str255, the Menu Manager only allocates enough memory necessary to store the number of characters in the menu title plus one character for the size byte. Following the menu data is a variable-length block of data used to store information about the menu’s items. You can modify the name of the menu, but do not modify the size of the string. After you modify the name, call DrawMenuBar() to redraw the menu bar and display the changes.

The following sample routine modifies a menu name and redraws the menu bar:

```c
void ChangeMenuName(short theMenu,
                     Str255 newName)
{
    MenuHandle hMenu;
    long menuFlags;
    int i;
```
hMenu = GetMHandle( theMenu );
HLock((Handle)hMenu);

// add characters of new name up to original
// length of the menu
for ( i = 1; i <= (*hMenu)->menuData[0]; i++)
{
    (*hMenu)->menuData[i] = newName[i];
}

// pad the rest of the menu title with spaces
for ( i = newName[0] + 1; i <= (*hMenu)->menuData[0]; i++)
{
    (*hMenu)->menuData[i] = ' ';
}

HUnlock((Handle)hMenu);
DrawMenuBar();
}

Another solution is to have a separate menu for each title. The menus can be created dynamically by the application when needed, or they can all be specified as initial resources and loaded at the start of your program. InsertMenu() and DeleteMenu() can be called to add and remove the menus from your application's menu bar.

How can I tell when a menu is about to be opened?

You can install an MBarHook procedure that gets executed before any menus are drawn. The MBarHook procedure receives as a parameter a rectangle defining the dimensions of the menu. This can be used to determine which menu is about to be opened.

Related FAQ

See also FAQ 16-28, "How do I install MBarHook and MenuHook procedures?"
When my menus are selected, why don't I get the menu IDs that I expect?

There are two identifiers for a menu: the menu resource ID and the menu ID. The menu resource ID is used to attach a menu to the menu bar with `GetMenu()`. The menu ID is the number that is passed to the program when a menu item selection is made. You can edit both of these values with a resource file editor. To simplify things, make sure the menu resource ID and the menu ID are the same.

When I use `GetMenu()` to get the handle of a menu in a menu bar, I get memory leaks. Why don't modifications I make affect the menu?

`GetMenu()` is meant to be used to get a handle to a menu from a ‘MENU’ resource, not to get the handle to a menu that is already loaded into memory. To get the handle of a menu already loaded into memory, call `GetMHandle()`.

Why won't my hierarchical menus show up in my application?

Hierarchical menus must be manually loaded with `GetMenu()` and then put in the menu with `InsertMenu()`. The second parameter of `InsertMenu()` must have a value of -1 to place the menu in the hierarchical portion of the menu list.

Related FAQs

- Check FAQ 16-1, “What is a hierarchical menu?”
- Check FAQ 16-24, “How do I put hierarchical menus in my program?”

How do I put hierarchical menus in my program?

Placing a hierarchical menu in an application is not a difficult procedure at all. The first step involves creating the parent menu and menu items. In your resource editor, begin creating a menu item that you want to display in a hierarchical menu when
selected, as you would any other menu item. You must then use the menu item editing dialog box to signify that this menu item will have a hierarchical menu attached to it. If you are using ResEdit, check the check box titled “has Submenu.” When you do this, a text entry field appears for you to enter the submenu ID. Enter the ID of the menu that you want to be displayed as a submenu. If you are using Resorcerer, set the radio button at the bottom of the dialog box to “sub-menu id.” Then enter the ID of the menu that you want displayed as a submenu in the TextEdit field to the right. The final step is to create the hierarchical menu itself. Create this menu as you would a normal menu. Set the menu ID to the value that you specified as the submenu ID when defining the parent menu item. Note that submenu IDs are limited to values between 0 and 255.

You need to add a couple of lines into your code to complete the installation. First, you must call GetMenu() to obtain a handle to your submenu. Then you must call InsertMenu() to insert the submenu into the menu list. The second parameter of InsertMenu() must be set to -1.

The following code shows the calls necessary to place a hierarchical menu in your application:

```
//********************
#define idFont 100

void InitMyMenus()
{
    Handle hMenuBar;
    MenuHandle hMenu;

    hMenuBar = GetNewMenuBar(128);
    SetMenuBar();

    // load and install submenu
    menu = GetMenu( idFont );
    InsertMenu( menu, -1);
}
```

Related FAQs

- See also FAQ 16-1, “What is a hierarchical menu?”
- See also FAQ 16-23, “Why won’t my hierarchical menus show up in my program?”
Special Menu Processing

This section answers FAQs on the more interesting aspects of menus. This is where we have a little fun. The section starts off with answering the age-old question, “How do I hide the menu bar?” It then provides code that shows how to use the MenuHook and MBARHook procedures to perform actions while the user has the mouse button pressed over the menu bar or a menu. It also discusses how to put up a menu of pictures and an icon as a menu title.

How do I hide the menu bar of an application?

Hiding the menu bar involves two general steps: setting the gray region of the main screen to include the area of the menu bar and setting the height of the menu bar to 0. To make the menu bar appear again, reset the gray region and menu bar height to their original values.

To hide the menu bar, as well as get rid of the rounded rectangles that normally make up the Desktop, create a region consisting of a union between the original gray area and the rectangular size of the screen found in the grRect field of the screen’s GDevice structure. Store this new region in the GrayRgn() global by calling LMSetGrayRgn(). Then redraw the Desktop by calling PaintOne(), specifying nil and the new gray region as parameters. Next, you have to redraw all of the windows so that a part of any window that was obscured by the menu bar but is now visible is redrawn. Recalculate the visible region for the windows. Finally, set the menu bar height to 0 by calling LMSetMBARHeight().

To make a hidden menu bar visible, call LMSetMBARHeight() to reset the menu bar to its original height, call LMSetGrayRgn() to restore the original gray region, and then redraw the menu bar by calling DrawMenuBar().

It is good practice to make the menu bar available when the user clicks in the region that the menu bar normally occupies. This area can be retrieved by calling LMGetMBARHeight(). It consists of a rectangle at the top of the main screen that spans the width of the screen and is as tall as the menu bar height. When the user clicks in this area, display the menu, process the menu commands, and then hide the menu bar again if desired.

The following sample routines hide the menu bar, wait for the user to press the mouse button, and then display the menu bar again:
```c
//*************************
Rect menuBarRect;
short menuBarHeight;
RgnHandle originalGrayRgn, newGrayRgn, underBarRgn;

void HideShowMenuBar( )
{
    GDHandle mainScreen;

    // store current gray region that displays menu bar and the current height of the menu bar
    originalGrayRgn   = LMGetGrayRgn();
    menuBarHeight     = LMGetMBarHeight();

    // calculate the rect of the menu bar to test
    // if mouse down is in if desired
    mainScreen         = GetMainDevice();
    menuBarRect        = (**mainScreen).gdRect;
    menuBarRect.bottom = menuBarHeight;

    HideMenuBar();

    while( !Button() );

    ShowMenuBar();

    // restore the original gray region to
    // make the menu bar visible
    LMSetGrayRgn( originalGrayRgn );
}

//*************************
void HideMenuBar( void )
{
    GDHandle mainScreen;
    Rect mainScreenBounds;
    RgnHandle mainScreenRgn;
    GrafPtr windowPort;
    GrafPtr oldPort;
    WindowPtr frontWindow;
}
get the gdhandle of the main screen which
is the screen with the menubar
mainScreen = GetMainDevice();
mainScreenBounds = (*mainScreen)->gdRect;

get new region encompassing entire screen
including area under menu bar and corners
mainScreenRgn = NewRgn();
newGrayRgn = NewRgn();
underBarRgn = NewRgn();
RectRgn( mainScreenRgn, &mainScreenBounds);
UnionRgn( mainScreenRgn, originalGrayRgn, newGrayRgn);
DiffRgn( newGrayRgn, originalGrayRgn, underBarRgn);
DisposeRgn( mainScreenRgn );

Set gray region to entire screen
LMSetGrayRgn( newGrayRgn );

GetPort( &oldPort );
GetWMgrPort( &windowPort );
SetPort( windowPort );
SetClip( newGrayRgn );

redraw the desktop to draw over the menu bar
PaintOne( nil, newGrayRgn );

in case any part of a window is covered
redraw it and recalculate the visible region
frontWindow = FrontWindow();
PaintOne( (WindowRef)frontWindow, underBarRgn );
PaintBehind((WindowRef)frontWindow, underBarRgn);
CalcVis((WindowRef)frontWindow);
CalcVisBehind((WindowRef)frontWindow, underBarRgn);

set menu bar height to 0
LMSetMBarHeight( 0 );

Restore the port
SetPort( oldPort );

} // HideMenuBar
void ShowMenuBar( void )
{
    WindowPtr    frontWindow;
    GrafPtr      windowPort;
    GrafPtr      oldPort;
    CGrafPtr     windowCPort;

    // Reset the menu bar height
    LMSetMBarHeight( menuBarHeight );

    // Restore the original gray region
    LMSetGrayRgn( originalGrayRgn );

    frontWindow = FrontWindow();
    CalcVis( (GrafPort *)frontWindow );
    CalcVisBehind( (WindowRef *)frontWindow, newGrayRgn );

    // Reset the clipping regions of the window mgr port
    GetPort( &oldPort );
    GetWMgrPort( &windowPort );
    SetPort( windowPort );
    SetClip( newGrayRgn );
    SetPort( oldPort );

    // Redraw the menu bar
    HiliteMenu( 0 );
    DrawMenuBar();
}

Related Topic

See also Chapter 10, "Finder and Desktop."

How do I create a menu that has an icon as its title?

To display an icon as a title of a menu, you need to set the menu title to a special value. The title of a menu is stored in the menuData field of the menu's MenuInfo structure. It must be a 6-byte Pascal string (one length byte, five character bytes). The first character (the second byte) must be set to 0x01, and the remaining four bytes must be set to a handle to an icon family. This signifies to the standard menu bar definition that the icon is to be drawn in the menu bar as the menu title.
The following example installs the menu with ID menuID into the menu using the icon with the ID iconID as its title. The title of the menu in the resource file should be a dummy string of five characters. The menu is inserted at location menuloc.

```
//********************************************************************
MenuHandle InstallMenuIcon( short menuID,
                           short iconID,
                           short menuloc )
{
    MenuHandle theMenu;
    Handle hIcons;
    Ptr tmpPtr;

    theMenu = GetMenu(menuID);
    GetIconSuite(&hIcons,iconID,svAllSmallData);
    ((*theMenu)->menuData)[1] = 0x01;

    tmpPtr = (Ptr)&(((*theMenu)->menuData)[2]);
    *((long *)tmpPtr) = (long)hIcons;

    InsertMenu(theMenu,menuloc);
    DrawMenuBar();

    return (theMenu);
}
```

Related Topic

See also Chapter 13, "Icons."

How do I create a customized menu definition procedure ('MDEF')?

The Menu Manager uses the standard menu definition procedure to accomplish the following:

- Calculate the dimensions of a menu
- Draw the menu items of a menu
- Perform the highlighting and unhighlighting in response to the user’s cursor movements
Determine which item the user selects from the menu

A customized menu definition must also perform these tasks.

A customized menu definition procedure enables you to display a menu of any format, including menus of pictures, icons, text of any size and font, or any combination of these. You can also specify multi-column menus.

The menu definition procedure has the following prototype:

```pascal
void main( short message, MenuHandle theMenu,
            Rect *menuRectPtr, Point hitPt,
            short *whichItemPtr )
```

When the Menu Manager calls the menu definition procedure to perform an action on a menu, it specifies the type of action in the message parameter and passes a handle to the menu record in the `theMenu` parameter. The messages that are sent include the following:

- **mSizeMsg** — calculates the dimensions of the menu and return the result in the `menuRect` parameter.

- **mDrawMsg** — draws the menu items within the menu rectangle specified by the `menuRectPtr` parameter.

- If the item is disabled it should be drawn using the colors returned by the `GetGray()` function.

- **mChooseMsg** — determines if the cursor is on a menu item and performs the necessary highlighting or unhighlighting.

- The mouse location is specified in the `hitPt` parameter, and the `whichItemPtr` parameter contains the currently highlighted item (that was set by a previous menu definition procedure in response to `mChooseMsg`). If the mouse location is in an active menu item that is different from the current item specified by `whichItemPtr`, the current menu item should be unhighlighted, the new item should be highlighted, and the `whichItemPtr` should be set to the newly-selected menu item. If the mouse location is not in an active menu, the current menu item should be unhighlighted and 0 should be returned in `whichItemPtr`. 

Once the menu definition procedure code is written, you use it to build an ‘MDEF’ resource. You then copy this resource to the resource file of your application.

To associate a menu with a custom menu definition procedure, you set the menuProc field of the menu’s MenuInfo structure to the resource ID of the ‘MDEF’ resource. (ResEdit and Resorcerer allow you to assign this ID by setting the ProcID field in the menu resource.) This will cause the Menu Manager to automatically call the procedure whenever an action needs to be performed on the menu.

The following is a sample menu definition procedure from Dave Mark’s book, The Macintosh Programming Primer, Volume II, Second Edition. It puts up a menu of pictures. The number of pictures and the resource IDs of the pictures are passed to the procedure in the enable flags field of the menu. The first two bytes specify the resource ID of the first PICT in the menu. The second two bytes specify the number of pictures in the menu. (This makes the procedure more generic but does not allow for the disabling of menu items.) The number of pictures and the resource ID of the first picture in the menu are determined by the GetNumPICTs() routine. The menu height and width are determined by the CalcitemHeightAndWidth() procedure: The height of the menu is determined by getting the maximum height of a picture and multiplying it by the number of menus; and the width of the menu is determined by the maximum width of all of the pictures. Items are highlighted and unhighlighted by the InvertItem() procedure, which calls InvertRect() on the ‘rect’ of the item.

```c
//*********************************************
#define kTopMargin 1
#define kLeftMargin 2

void DoSizeMessage( MenuHandle menu, Rect *menuRectPtr );
void DoDrawMessage( MenuHandle menu, Rect *menuRectPtr );
void DoChooseMessage( MenuHandle menu, Rect *menuRectPtr,
                      Point hitPt, short *whichItemPtr );
void InvertItem( short itemNumber, short itemHeight, Rect
                 *menuRectPtr );
void DrawCenteredPict( PicHandle pic, Rect *rectPtr );
void CalcitemHeightAndWidth( short basePICTid, short numPICTs,
                             short *widthPtr, short *heightPtr );
void GetNumPICTs( MenuHandle menu, short *baseIDPtr,
                 short *numPICTsPtr );
```
/****************************** main */

pascal void main( short message,
    MenuHandle menu,
    Rect *menuRectPtr,
    Point hitPt,
    short *whichItemPtr )
{
    switch( message )
    {
        case mDrawMsg:
            DoDrawMessage( menu, menuRectPtr );
            break;
        case mChooseMsg:
            DoChooseMessage( menu, menuRectPtr, hitPt, whichItemPtr );
            break;
        case mSizeMsg:
            DoSizeMessage( menu, menuRectPtr );
            break;
    }
}

/**************************** DoSizeMessage */

void DoSizeMessage( MenuHandle menu,
    Rect *menuRectPtr )
{
    short basePICTid, numPICTs, maxPICTWidth, maxPICTHeight;

    GetNumPICTs( menu, &basePICTid, &numPICTs );
    CalcitemHeightAndWidth( basePICTid, numPICTs,
        &maxPICTWidth, &maxPICTHeight );

    (**menu).menuWidth = maxPICTWidth + 2 * kLeftMargin;
    (**menu).menuHeight = (maxPICTHeight + kTopMargin * 2) *
        numPICTs;
/****************************** DoDrawMessage */

void DoDrawMessage( MenuHandle menu, 
     Rect *menuRectPtr )
{
    short basePICTid, numPICTs, maxPICTWidth, 
    maxPICTHeight, itemHeight, i;
    Rect r, tempRect;
    PicHandle pic;

    GetNumPICTs( menu, &basePICTid, &numPICTs );
    CalcitemHeightAndWidth( basePICTid, numPICTs, 
                  &maxPICTWidth, &maxPICTHeight );

    itemHeight = maxPICTHeight + kTopMargin * 2;
    r.top = menuRectPtr->top + kTopMargin;
    r.left = menuRectPtr->left + kLeftMargin;
    r.bottom = r.top + maxPICTHeight;
    r.right = r.left + maxPICTWidth;

    for ( i=0; i<numPICTs; i++ )
    {
        pic = GetPicture( basePICTid + i );
        DrawCenteredPict( pic, &r );
        OffsetRect( &r, 0, itemHeight );
    }
}

/****************************** DoChooseMessage */

void DoChooseMessage( MenuHandle menu, 
     Rect *menuRectPtr, 
     Point hitPt, 
     short *whichItemPtr )
{
    short basePICTid, selectedItem, numPICTs, maxPICTWidth, 
    maxPICTHeight, itemHeight;
    Rect r;
GetNumPICTs( menu, &basePICTid, &numPICTs );
CalcitemHeightAndWidth( basePICTid, numPICTs, &maxPICTWidth, &maxPICTHeight );

itemHeight = (2 * kTopMargin) + maxPICTHeight;

if ( PtInRect( hitPt, menuRectPtr ) )
{
    selectedItem = ( (hitPt.v - menuRectPtr->top) / itemHeight ) + 1;
    if ( ( *whichItemPtr > 0 ) && ( *whichItemPtr != selectedItem ) )
    {
        InvertItem( *whichItemPtr, itemHeight, menuRectPtr );
    }

    if ( *whichItemPtr != selectedItem )
    {
        *whichItemPtr = selectedItem;
        InvertItem( *whichItemPtr, itemHeight, menuRectPtr );
    }
}
else if ( *whichItemPtr > 0 )
{
    InvertItem( *whichItemPtr, itemHeight, menuRectPtr );
    *whichItemPtr = 0;
}

//***************************************************************************
void InvertItem( short itemNumber,
                 short itemHeight,
                 Rect *menuRectPtr )
{
    Rect r;

    r = *menuRectPtr;
r.top += ( (itemNumber-1) * itemHeight );
r.bottom = r.top + itemHeight;

InvertRect( &r );
}

/******************************* DrawCenteredPict */

void DrawCenteredPict( PicHandle pic,
                       Rect *rectPtr )
{
    Rect pictRect;
    pictRect = (**pie).pieFrame;
    OffsetRect( &pictRect, rectPtr->left - pictRect.left,
                rectPtr->top - pictRect.top);
    OffsetRect( &pictRect,(rectPtr->right - pictRect.right)/2,
                (rectPtr->bottom - pictRect.bottom)/2);
    DrawPicture( pic, &pictRect );
}

/******************************* CalcitemHeightAndWidth */

void CalcitemHeightAndWidth( short basePICTid,
                             short numPICTs,
                             short *widthPtr,
                             short *heightPtr )
{
    short i;
    Rect r;
    PicHandle pic;
    *widthPtr = 0;
    *heightPtr = 0;

    for ( i=0; i<numPICTs; i++ )
    {
        pic = GetPicture( basePICTid + i );

How do I install MBarHook and MenuHook procedures?

The MBarHook and MenuHook procedures are installed by setting the low-memory globals, MBarHook and MenuHook, to pointers to your procedures. The universal headers provide two functions to do this: LMSetMBarHook() and LMSetMenuHook(). The MBarHook procedure is called immediately before a menu is drawn by MenuSelect(). The MenuHook function is drawn continuously during the MenuSelect() operation.
The following sample InstallHooks() function installs the two sample functions MyMBarHook() and MyMenuHook() and can be called anytime in the program. Before a menu is drawn, MBarHooks() draws an inverted image of the menu in the window pWindow. Every call to MyMenuHook() flashes the string “MENU HOOK” in the window.

```pascal
pascal short MyMBarHook( Rect * theRect );
pascal void MyMenuHook( void );

WindowPtr pWindow;

void InstallHooks()
{
    LMSetMBarHook(NewMBarHookProc(MyMBarHook));
    LMSetMenuHook( NewMenuHookProc(MyMenuHook));
}

pascal short MyMBarHook( Rect * theRect )
{
    GrafPtr oldPort;
    long theTicks;
    Rect Rect1;

    Rect1 = *theRect;

    GetPort(&oldPort);
    SetPort(pWindow);

    InvertRect(&Rect1);
    Delay(15, &theTicks);
    InvertRect(&Rect1);

    SetPort(oldPort);

    // return 0 for the menu to be drawn
    return(0);
}
```
How can I draw to a window while a menu is pulled down obscuring the window?

Drawing while the menu is pulled down is made difficult because all of the menu processing is done by calling the modal function MenuSelect(). You can do your own processing by installing MBa rHook and the MenuHook procedures. Each time a menu is pulled down and your MBa rHook function is called, take the dimensions of the menu, convert them to your window’s local coordinates and adjust the clipping region of the window so that it does not include the area covered by the pulled down menu. Then you can perform your drawing each time your MenuHook function is called.

The following sample code continuously inverts the window pWindow while the mouse button is pressed in the menu bar or in a window — it does not draw on any menu that is open on top of it. The sample function InstallDrawHooks() installs the sample routines MyDrawMBa rHook() and MyDrawMenuHook() as the MBa rHook and the MenuHook procedures. The routine MyDrawMBa rHook() gets the region of the window minus the area covered by the menu and sets that as the windows clipping region. The coordinates of the menu ‘rect’ that are passed to the MBa rHook function must be converted to local coordinates. The routine MyDrawMenuHook() then inverts the window every time it is called.
/*************************/
pascal short MyDrawMBarHook(Rect * theRect);
pascal void MyDrawMenuHook(void);

WindowPtr pWindow;

/*************************
void InstallDrawHooks()
{
  LMSetMBarHook(NewMBarHookProc(MyDrawMBarHook));
  LMSetMenuHook( NewMenuHookProc( MyDrawMenuHook));
}

/*************************/
pascal short MyDrawMBarHook(Rect *theRect)
{
  RgnHandle uncoveredRgn, menuRgn, windowRgn;
  GrafPtr oldPort;
  Rect menuRect;
  Point originPt = {0,0};

  menuRgn = NewRgn();
  windowRgn = NewRgn();
  uncoveredRgn = NewRgn();

  GetPort(&oldPort);
  SetPort(pWindow);

  menuRect = *theRect;
  GlobalToLocal(&originPt);
  OffsetRect(&menuRect, originPt.h, originPt.v);
  RectRgn(menuRgn, &menuRect);

  // assumes window is the top window and is not covered by
  // any other window
  RectRgn(windowRgn, &pWindow->portRect);
  DiffRgn(windowRgn, menuRgn, uncoveredRgn);
  SetClip(uncoveredRgn);

  SetPort(oldPort);
DisposeRgn(menuRgn);
DisposeRgn(windowRgn);
DisposeRgn(uncoveredRgn);

   // return 0 for the menu to be drawn
   return(0);
}

/***************
pascal void MyDrawMenuHook()
{
    GrafPtr oldPort;
    long theTicks;

    GetPort(&oldPort);
    SetPort(pWindow);

    InvertRgn( pWindow->clipRgn );
    Delay(30., &theTicks);
    InvertRgn( pWindow->clipRgn );

    SetPort(oldPort);
}

Related Topics

☐ See also Chapter 23, “QuickDraw: Drawing.”

☐ See also Chapter 33, “Windows.”

How do I create a menu of pictures?

The standard menu definition procedure only allows you to display text and icons.
You can create a menu of pictures by writing your own MDEF.

Related FAQ

☐ See also FAQ 16-27, “How do I create a customized menu definition procedure (‘MDEF’)?”
Related Topic

See also Chapter 24, "QuickDraw: Pictures."

**Topic-Related FAQs**

See also FAQ 2-10, "What RGB value does the system use for dimmed buttons, menus, and window titles?"

See also FAQ 13-2, "Which icon types can I put in dialog boxes and menus?"
Power Macintosh Programming

When Apple released the first Power Macintosh, life for Macintosh programmers definitely changed. You could no longer just pass a function pointer to a Toolbox function and expect your function to be called properly on a Power Mac. If you expect your programs to run on a Power Mac, you have to make sure you recompile your code using the universal headers released by Apple. This chapter addresses these issues, including questions on creating fat binaries, universal procedure pointers, and other PowerPC-related issues.

There are three subsections in this chapter:

- Definitions
- Programming on the Power Mac
- Universal Procedure Pointers
Definitions

This section of the Power Macintosh Programming chapter answers FAQs about some important terms that are discussed in the following sections. It defines the universal headers now used in the Macintosh programming environment and some new concepts directly related to the Power Macintosh.

What is a fat binary?

A fat binary is an application that contains PowerPC code in its data fork and 68K 'CODE' resources in its resource fork. Both sets of application code share the other resources located in the resource fork. Since the binary contains PowerPC native code, it will run in native mode on PowerPC Macintoshes. The Process Manager will not look for the PowerPC code fragments unless the 'cfrg' resource is present.

Related FAQs

- See also FAQ 17-4, "What is a code fragment?"
- See also FAQ 17-7, "How do I create a fat binary application?"

What are the universal headers files?

Universal headers are the latest system headers from Apple. They are designed to work for code compiled for either a 68K Macintosh or a Power Macintosh. The new header files encompass a variety of changes. For example, QuickDraw globals are now fields of a global variable qd. The main change to the universal headers, however, is the transition from the use of 68K procedure pointers to the new universal procedure pointers (UPP). Each procedure pointer has been replaced with a specific UPP. For each kind of ProcPtr two new macros have been defined:

New***Proc - allocates a new RoutineDescriptor appropriate for the specified callback.
Call***Proc - allows you to call the callback using the UPP.
What is a universal procedure pointer?

A universal procedure pointer (UPP) is defined to be either an address of a routine descriptor, a data structure used by the Mixed Mode Manager to execute a routine, or a normal 680x0 procedure pointer. A UPP describes to the Mixed Mode Manager which processor code the function uses (either PowerPC or 680x0), how many arguments the function uses, and the programming language used to implement the function.

What is a code fragment?

Code fragments are the basic units of PowerPC executable code on the Power Macintosh. They are equivalent to the code resources used in the 68K Macintosh operating systems. A code fragment can exist in several forms: as an application that can be launched from the Finder, as an import library, or as code contained in a stand-alone code resource.

The code and data of a native PowerPC application are usually stored in the application’s data fork. This is different from the ‘CODE’ resources that make up a 68K Macintosh application which are stored in the application’s resource fork. This difference allows for easy creation of “fat” applications, which are applications that contain both PowerPC native code and 68K Macintosh code. Unlike the 68K ‘CODE’ resources which are normally limited to 32K in size, code fragments have no limits on the size of their code or data. Multiple code fragments can be stored in a single data fork.

The Process Manager determines whether or not an application is PowerPC native by checking for the existence of a ‘cfrg’ resource with the ID 0. The ‘cfrg’ resource specifies characteristics of code fragments including the fragment’s instruction set architecture, default stack size, and location.

Related FAQs

- See also FAQ 17-1, “What is a fat binary?”
- See also FAQ 17-7, “How do I create a fat binary application?”
Programming on the Power Mac

This section of the Power Macintosh Programming chapter answers FAQs about general issues involving programming on the Power Macintosh. It discusses the steps that are required to make a 68K application work on the Power Mac. It describes how to make a fat application, and addresses other issues pertinent to Power Macintosh programming.

What do I have to do to port my 68K code to be native?

To port your 68K code to native PowerPC code, you will have to make a few modifications.

The biggest change to your code will be the implementation of universal procedure pointers (UPPs). All functions that are accessed through a procPtr, such as callback or completion routines, need to have a UPP created for them. This UPP is then used in place of the old procedure pointer.

The QuickDraw global variables are now stored as structure fields in the global variable qd.

Some of the interrupt-time functions, such as VBL tasks and Time Manager Tasks, get passed the structures used to create them as parameters instead of only being able to access them via registers using Assembly language.

Native programs do not have an AS world. The AS world setup that was necessary in 68K interrupt-time tasks to access global variables is not necessary. Native programs can access the global variables of the main application by default.

How do I convert procptrs to universal procedure pointers on the Power Macintosh?

Universal procedure pointers (UPPs) are created with the NewRoutineDescriptor() call. The universal headers provide function macros that create all of the standard universal procedure pointers for you. For example, the routine ErrorSound() in Dialogs.h is defined as taking one argument, a SoundUPP procedure.
extern pascal void ErrorSound(SoundUPP soundProc)
    ONEWORDINLINE(0xA98C);

To find the routine that will create a UPP of type SoundUPP to pass to this
function, search the header file for the function that calls
NewRoutineDescriptor() and returns a UPP of type SoundUPP. This is
found in the definition of the NewSoundProc() macro.

#define NewSoundProc(userRoutine)     
    (SoundUPP) NewRoutineDescriptor((ProcPtr)(userRoutine),
        uppSoundProcInfo, GetCurrentArchitecture())

The defined macro calls NewRoutineDescriptor() with the appropriate fields
and returns a UPP of the proper type. All of the macros that create a UPP for you
will be named NewXXXProc().

The call to ErrorSound() changes from the 68K version

ErrorSound(MyAlertSoundProc) );

to a version that uses the universal procedure pointer

UniversalProcPtr newAlertSoundProc;
newAlertSoundProc = NewSoundProc(MyAlertSoundProc);

ErrorSound( newAlertSoundProc );

**How do I create a fat binary application?**

A fat binary application is created by combining in a single application the code
resources of a 68K application and the PowerPC code in the data fork of a native
application along with the other resources in the resource fork.

To create a fat binary using CodeWarrior:

1. Create a 68K project file and a PowerPC project file.
2. Build a 68K version of the application.
3. Create a PowerPC project.
4. Using the Project/Add Files menu command add the 68K application to the
   PowerPC project. The resources that the fat application will require will be
   automatically added if they exist in the 68K version.
5. Select Make from the Project menu to build the fat application.
To create a fat binary using Symantec C++:

1. Using the Think Project Manager create a 68K version of the application.
2. Using the Symantec Project Manager create a PPC version of the application. Do not add the resources in the PPC project. They will be included in the fat application through the 68K application.
3. Select Options from the Project menu. Select the Project Type option from the left column. In the Project Options dialog box check the Merge checkbox. Click on the Select Application button and select the 68K version of the application to merge. Close the dialog.
4. In the Project menu select Build Application.

Related FAQs

- See also FAQ 17-1, “What is a fat binary?”
- See also FAQ 17-4, “What is a code fragment?”

**How can I tell if a program is a native PowerPC, 68K, or fat application?**

68K applications maintain their code in their resource file as ‘CODE’ resources. The ‘CODE’ resource with ID 0 contains the application’s jump table along with other information. A PowerMac application maintains its program code in its data fork in a block of code called a code fragment. It has a ‘cfrg’ resource in its resource file which tells the Process Manager that the application contains PowerPC code.

If the application is a fat application it will have both a ‘cfrg’ resource and a ‘CODE’ resource of ID 0. If the application is native only, it will not have a ‘CODE’ resource of ID 0. If the application is a 68K application, it will not have a ‘cfrg’ resource.

**How can I force a fat binary application to run the 68K code under emulation instead of running the native code?**

To run the 68K code under emulation, delete the ‘cfrg’ resource. Without the ‘cfrg’ resource, the system will not attempt to run the application in native mode.
**Why does my application crash on a PowerMac whenever it tries to access a ProcPtr?**

You need to use universal procedure pointers when referencing functions on the Power Macintosh.

**Related FAQs**

- See also FAQ 17-3, "What is a universal procedure pointer?"
- See also FAQ 17-6, "How do I convert procptrs to universal procedure pointers on the Power Macintosh?"

**Do I have to do all of the A5 work on PowerMacs to access globals as I do on 680x0 machines?**

If you are writing purely native code, that is, code that is meant to run only on Power Macintoshes and not on 68K Macintoshes, you do not have to save and restore A5 to access your application's global variables. This is because a PowerPC application does not have an A5 world (except for a mini-A5 world created by the Process Manager).

680x0 applications running under emulation on a Power Mac still access the A5 world; therefore, you must explicitly set up and restore the application's A5 world.

**Related Topic**

- See also Chapter 15, "Memory."

**Universal Procedure Pointers**

This section of the Power Macintosh Programming chapter answers FAQs about using universal procedure pointers.
Can I leave my universal procedure pointer calls in 68K compiled code?

When compiling for 68K, the macros for creating universal procedure pointers do nothing but return procedure pointers. The macros for calling UPPs do nothing but call the procedure pointers. The macros for disposing UPPs have no function at all.

How do I call a function using a universal procedure pointer?

To call a function through a universal procedure pointer you have to use the CallUniversalProc() routine. UPPs defined by the Toolbox have corresponding customized Call***Proc() procedures defined in the universal headers such as the New***Proc() functions defined for them.

Do I have to dispose of the universal procedure pointer that I get from the New***Proc() macros in the universal header files?

The macros defined in the universal header files that return universal procedure pointers call the routine NewRoutineDescriptor() which creates a new routine descriptor. A UPP to a procedure only needs to be created once in a program. When the application exits, all existing UPPs are automatically disposed of by the system. They can be explicitly disposed of with DisposeRoutineDescriptor(). If a function that takes a UPP as a parameter is called more than once, allocate the UPP ahead of time. For example don’t call ModalDialog() like this:

ModalDialog( NewModalFilterProc(MySimpleDlogFilter), &itemHit );

Instead do this:

ModalFilterUPP myUPP;
myUPP = NewModalFilterProc(MySimpleDlogFilter);
ModalDialog( myUPP , &itemHit );

Since the universal procedure pointers created by NewRoutineDescriptor() are blocks of nonrelocatable memory, they should be allocated early in your program to minimize heap fragmentation.
How do I explicitly call a routine associated with a universal procedure pointer?

Universal procedure pointers are called with the CallUniversalProc() call. The universal headers provide function macros that do this for you through the standard predefined Toolbox procedures. For example, the Dialog Manager's standard filter procedure, which can be retrieved by calling GetStdFilterProc(), is defined as a ModalFilterUPP, the same type of procedure pointer that ModalDialog() accepts. Searching the Dialogs.h universal header file, you find the call CallModalFilterProc which is defined as:

```c
#define CallModalFilterProc(userRoutine, theDialog, theEvent, itemHit) 
    CallUniversalProc((UniversalProcPtr)(userRoutine), 
    uppModalFilterProcInfo, (theDialog), (theEvent), (itemHit))
```

To explicitly call the Dialog Manager’s standard filter procedure you would use CallModalFilter() as shown below:

```c
 ModalFilterUPP standardProc;
 OSErr myErr;

 myErr = GetStdFilterProc(&standardProc);
 if(!myErr)
     returnVal = CallModalFilterProc(standardProc,

     currentDialog, theEventIn, theDialogItem);
```

Can I cast a procptr to be a universal procedure pointer for my Power Macintosh programs?

No. You need to create a universal procedure pointer.

Related FAQ

See also FAQ 17-10, “Why does my application crash on a PowerMac whenever it tries to access a ProcPtr?”
See also FAQ 21-9, "How do I execute a Time Manager task in a native PowerPC application?"

See also FAQ 21-15, "How do I access application globals from a native PowerPC Time Manager task?"

See also FAQ 22-5, "How do I execute a PowerPC native VBL task?"

See also FAQ 22-11, "How do I access application globals from a native PowerPC VBL task?"
Printing

Printing on the Macintosh is almost as easy as drawing in a window. The Printing Manager works with the printer driver of the currently selected printer to allow for QuickDraw-based printing. If you follow a few simple rules, the same code that you use to draw on your application's windows can be called to send the data to the printer.

The steps for basic printing of text or graphics on a single page are:

1. Initialize the printer
2. Call QuickDraw commands to draw text or graphics
3. Close the printer
While this is a slight oversimplification, it is basically the whole story for single-page printing. The process for printing data that extends across multiple pages is a little more complex. This chapter describes the steps necessary to set up your environment for printing, how to print documents that span multiple pages, and how to use the two standard printing dialogs to allow a user to customize the printing operation. It also touches upon the more complex issues of printing that are addressed through picture comments.

There are three subsections in this chapter:

- Definitions
- Printers and Printing
- Using Picture Comments

Definitions

This section of the Printing chapter defines and describes "picture comments," as well as several other concepts central to programming for printing.

What is a picture comment?

A picture comment is composed of data or special commands that are used by an output device, such as a printer, to perform special processing when drawing the picture. After PrOpenPage() is called, picture comments can be inserted into the drawing code that is to be sent to printer with the PicComment() procedure.

Picture comments can also be inserted into a picture by calling PicComment() after OpenPicture() or OpenCPicture() have been called.

Appendix B of Inside Macintosh: Imaging with QuickDraw lists available picture comments and describes how to use them.

Related FAQs

- See also FAQ 18-12, "How can I print rotated text?"
- See also FAQ 18-14, "How can I print good-looking dashed lines?"
Printers and Printing

This section of the Printing chapter answers FAQs that deal with setting up the printer to print text and graphics. It describes how to display the Page Setup and Print dialogs and process the information returned by them. It explains how to print data that spans across multiple pages. It also discusses methods of getting and setting information about the printer and the printing environment.

How do I put up the standard print dialogs?

The standard “Page Setup...” dialog is displayed by calling PrStlDialog(). This dialog gets information such as the size and page orientation to be used when printing. The standard print job dialog is displayed by calling PrJobDialog(). This dialog gets information such as the number of copies and the range of pages to print.

How do I output text or graphics to a printer?

To print text and graphics to a printer you must first initialize the Printing Manager by calling PrOpen(). If you want to display the standard printing dialog you call PrJobDialog() passing it a handle to a TPrint structure which is used to hold options for the print job. You can fill the TPrint record with default settings with PrintDefault(). Next you obtain a handle to the TPrPort structure by calling PrOpenDoc(), passing it the handle to the TPrint structure. As a final step before you start actually drawing, initialize the fields of the printing graphics port and signify that the drawing to a page will begin by calling PrOpenPage(). Now you are ready to draw as you would to a Window’s graphics port. Set the current port to the printing port which is stored in the gPort field of the TPrPort structure. Perform your QuickDraw drawing operations. After you complete drawing to the page, signify that printing to the page is completed with PrClosePage() and end the printing job with PrCloseDoc().
The sample below prints the string “This is a print sample.”

```c
//************************
void PrintStuff(void)
{
    GrafPtr savePort;
    TPrStatus prStatus;
    TPPrPort printPort;
    THPrint hPrint;
    GetPort(&savePort);
    PrOpen();
    hPrint = (THPrint) NewHandle(sizeof(TPrint));
    PrintDefault(hPrint);
    if (PrJobDialog(hPrint))
    {
        printPort = PrOpenDoc(hPrint, nil, nil);
        SetPort(&printPort->gPort);
        PrOpenPage(printPort, nil);
        MoveTo(100, 100);
        DrawString("\pThis is a print sample.");
        PrClosePage(printPort);
        PrCloseDoc(printPort);
        if ( ((*hPrint)->prJob.bJDocLoop == bSpoolLoop) &&
             (!PrError()) )
            PrPicFile(hPrint, nil, nil, nil, &prStatus);
    }
    PrClose();
    SetPort(savePort);
}
```

**How do I print when the user has selected to spool the document to disk?**

First you must determine whether or not the document spooled to disk. You do this by checking the value stored in the bJDocLoop field of the prJob structure stored as a field in the TPrint structure whose handle is passed to PrOpenDoc(). If it is set to the constant bSpoolLoop, then the document was spooled to disk. To have this spooled file output to the printer, call PrPicFile(). The code below prints a job spooled to disk.
How do I print graphics or text that requires more than one page?

It is the program's job to draw each page of data, one at a time, and send the data to the printer. The printable rectangle of the printing graphics port can be retrieved from the rPage field of the TPrInfo structure in the printing record.

You can get the first and last page numbers to print as well as the number of copies from the prJob structure in the printing record.

<table>
<thead>
<tr>
<th>prJob.iFstPage</th>
<th>= page number of first page to print</th>
</tr>
</thead>
<tbody>
<tr>
<td>prJob.iLstPage</td>
<td>= page number of last page to print</td>
</tr>
<tr>
<td>prJob.iCopies</td>
<td>= number of copies to print</td>
</tr>
</tbody>
</table>

The best method is to have your drawing procedure take a destination rectangle and automatically shift the drawing to draw into the printing port coordinates.

The sample below prints the number of copies of the specified page range of a document. DrawMyStuff() draws the appropriate data to fit in the passed in destination rectangle to the current graphics port. Note that the sample below assumes that the page range is valid.

```c
//********************
void PrintMyStuff()
{
    THPrint hPrint;
    short pageWidth, pageHeight, numCopies;
    Rect pageRect;
    ...
    pageWidth = (*hPrint)->prinfo.rPage.right - (*hPrint)->prinfo.rPage.left;
    pageHeight = (*hPrint)->prinfo.rPage.bottom - (*hPrint)->prinfo.rPage.top;
```
for ( currentPage = prJob.iFstPage; currentPage <= prJob.iLstPage; currentPage++)
{
    SetRect( &pageRect, 0, (currentPage - 1) * pageHeight, pageWidth,
            currentPage * pageHeight);
    for (numCopies = 0; numCopies < prJob.iCopies; numCopies++)
    {
        PrOpenPage(printPort, nil);
        DrawMyStuff(&pageRect);
        PrClosePage(printPort);
    }
}...

How do I save options set in the Macintosh Page Setup and Print dialogs?

The information that is entered by the user in Page Setup and Print dialogs is collected in the TPrint structure passed to the two routines that process their interaction with the user. This filled TPrint structure, also called the "print record," is then used to initialize the graphics port used for printing. To save the page setup information previously entered by a user you need only to save the print record used in the last call to PrSt1Dialog(). The options entered into the Print dialog cannot be saved in or restored by the print record. This is because options in the PrJobDialog are considered to be "job dependent," as opposed to "document dependent." For example, suppose the user chooses landscape printing in the Page Setup dialog (see Figure 18-1), and sets the number of copies at 5 in the Print dialog (see Figure 18-2). If the print record is passed to PrSt1Dialog(), the dialog shows that landscape printing was selected in the print record. However, if that print record is passed to PrJobDialog(), the number of copies is shown as 1.

For the most efficient use of applications within the page setup options, save the print record as a resource in your resource file.

Be careful not to call PrintDefault(). This fills in the TPrint structure with default settings.
How do I get the page size?

There are two types of page sizes. The paper rectangle gives the physical size of the paper. The page rectangle represents the area of the paper that the printer can use to format the document, in other words, the printable area.

The paper rectangle can be retrieved from the rPaper field of the TPrint structure, the print record. The page rectangle can be retrieved from the rPage field of the pInfo structure in the print record.

The print record is initialized to the default values by PrintDefault() and is modified by PrStlDialog() which brings up the Page Setup dialog.

How can I set the resolution at which my printer prints?

You can get a list of the resolutions supported by the current printer driver and then set the resolution it uses to print by calling PrGeneral(). To get the supported resolution, call PrGeneral() with the iOpCode field of its parameters set to 4. To set the supported resolution, call PrGeneral() with the iOpCode field of its parameters set to 5. The actual structure passed to PrGeneral() depends upon
the opcode used. When getting the resolutions (opcode 4), PrGeneral() uses the TGetRs1B1k structure. When setting the resolution (opcode 5), PrGeneral() uses the TSetRs1B1k structure.

How do I get the name and type of the currently selected printer?

The name of the selected printer can be found in the printer driver file as a resource of type PAPA, with an ID -8192. To get the name call PrOpen() which will cause the Printing Manager to open the currently selected printer driver for you, and place it at the top of the resource chain. Then use GetResource() to get the 'PAPA' resource with ID -8192.

The type of the currently selected printer is stored in the System file in a resource of type 'STR' with an ID of -8192. This will be the name of the driver selected, such as LaserWriter or ImageWriter.

```c
//*************************
void GetInfoBeforeHand()
{
    Str255 prName, prType;

    GetPrinterName(prName);
    GetPrinterType(prType);
}

OSErr GetPrinterName(Str255 prName)
{
    OSErr err = noErr;
    char **papaDataHandle;
    PrOpen();
    papaDataHandle = Get1Resource('PAPA', -8192);
    if(!papaDataHandle)
    {
        err = ResError();
        prName = "\p";
    }
    else
    {
```
HLock(papaDataHandle);
BlockMove(*papaDataHandle, prName,
    (**papaDataHandle)+1);
HUnlock(papaDataHandle);
ReleaseResource((Handle) papaDataHandle);
}
PrClose();
return err;
}
OSErr GetPrinterType(Str255 prType)
{
    OSErr err = noErr;
    StringHandle strHandle;
    strHandle = GetString(-8192);
    if(!strHandle)
    {
        err = ResError();
        prType = "\p";
    }
    else
    {
        HLock((Handle)strHandle);
        BlockMove(*strHandle, prType, (**strHandle)+1);
        HUnlock((Handle)strHandle);
        ReleaseResource((Handle) strHandle);
    }
    return err;
}

**How can I determine if a printer supports color?**

You can determine if a printer supports color by checking the *GrafPort* returned by *PrOpenDoc()* . If the *rowBytes* field of *GrafPort* is negative, then the Printing Manager returns a color *GrafPort* indicating that the printer supports color and/or grayscale and Color QuickDraw is present. You can then make Color QuickDraw calls.
How can I set the name of the print job created by PrOpen(), as seen in the Print Monitor?

The Printing Manager's default behavior is to use the front window's title as the job name. One way to change this behavior is to create a "status dialog" while you print. If this window is brought up before PrOpen(), the Printing Manager will use that window's title as the job name. Another way is to put up a window outside the screen's visible coordinates.

Using Picture Comments

The section of the Printing chapter answers FAQs about more advanced topics dealing with picture comments. It discusses printing special types of lines and provides an example of using picture comments to print rotated text.

How can I print rotated text?

Picture comments can be used to rotate text on PostScript printers and on QuickDraw-based drivers that support text rotation. For other devices you must still provide a bitmap of the rotated text.

Appendix B of Inside Macintosh: Imaging with QuickDraw shows how to use picture comments.

The sample routine below is called after PrOpenPage() has been called to initialize a page. This routine PostScript to print a string that has been rotated 90 degrees. The values for the picture comments are defined below.

```c
#define PostscriptBegin 190
#define PostscriptEnd 191
#define TextBegin 150
#define TextEnd 151
#define TextCenter 154

typedef struct
{
    Byte tJus;
} Pic
```
Byte tFlip;
short tRot;
Byte tLine;
Byte tCmnt;
}
TTxtPicRec, *TTxtPicPtr, **TTxtPicHdl;

typedef struct
{
    Fixed y;
    Fixed x;
}
TTxtCenter, *TTxtCenPtr, **TTxtCenHdl;

//************************
void PrintRotatedText()
{
    Handle textCHdl, centerCHdl;

textCHdl = NewHandle(sizeof(TTxtPicRec));
centerCHdl = NewHandle(sizeof(TTxtCenter));
if(textCHdl && centerCHdl)
{
    (**(TTxtPicHdl)textCHdl).tJus = 1;
    // no flip
    (**(TTxtPicHdl)textCHdl).tFlip = 0;
    // degrees of rotation
    (**(TTxtPicHdl)textCHdl).tRot = 90;
    (**(TTxtPicHdl)textCHdl).tLine = 0;
    (**(TTxtPicHdl)textCHdl).tCmnt = 0;
    // Tell the postscript driver where the center of
    // rotation is
    // relative to the current pen position.
    (**(TTxtCenHdl)centerCHdl).x = 0;
    (**(TTxtCenHdl)centerCHdl).y = 0;

    // Flush the Postscript state

    PicComment(PostscriptBegin, 0, NULL);
    PicComment(PostscriptEnd, 0, NULL);

    PicComment(TextBegin, sizeof(TTxtPicRec),
              textCHdl);
How can I print thinner lines than one QuickDraw point?

QuickDraw and PostScript define one point to be 1/72 of an inch which is the normal resolution on a Macintosh screen. On some PostScript printers you can draw hairlines, lines less than 1/72 of an inch wide, by using the SetLineWidth picture comment. Since the resolution of a PostScript device is 300 dpi, you can draw hairlines as small as 1/300 of an inch.

How can I print good-looking dashed lines?

There are two picture comments available, DashedLine (value 180) and DashedStop (value 181), that allow you to draw dashed lines on PostScript printers.
Processes: Notification Manager

There are times when you want your application to perform some task in the background and report to the user when the task is completed. But, since the application is in the background, it can't put up a dialog to capture the user's attention. That is exactly why the Notification Manager was created. The Notification Manager lets you queue up a notification record and, when your notification hits the front of the queue, the Notification Manager plays a specified sound or brings up a visual alert telling the user to move your application to the front, or perhaps calls a callback routine that you have provided. Though it is incredibly useful, most Mac programmers know little about the Notification Manager. The questions in this chapter offer a pretty good range of information about this important part of the Mac Toolbox.

There are two subsections in this chapter:

- Using the Notification Manager
- Notification Procedures
Using the Notification Manager

This section of the Processes: Notification Manager chapter answers FAQs about the basics of using the Notification Manager. Here I enumerate and distinguish among the various methods of user notification the Manager makes available. I also discuss the operation of the notification queue, and offer ways to avoid commonly occurring problems.

How can my application get the user's attention when it is in the background?

A process running in the background can send the user an asynchronous notification by using the Notification Manager. There are four different ways that a notification can signal a user. Applications most often use combinations of these.

The Notification Manager can:

1. Display a diamond-shaped mark in the Application menu next to the name of the application that issued the notification. The mark only appears if your application is in the background.
2. Cause a small icon to blink in the menu bar, alternating with the Apple menu icon or the Application menu.
3. Play a 'snd' resource or the user's system alert sound.
4. Bring up an alert box that displays a customized message.

A response notification procedure can also be executed as the result of a notification.

How can my application display a dialog box in front of all other windows on the screen when it is running in the background?

See FAQ 19-1, "How can my application get the user's attention when it is the background?"
Related Topic

See also Chapter 6, “Dialogs and Controls.”

**How do I issue a notification to the user from an application in the background?**

To issue a notification to the user while an application is in the background, you must install a notification request into the Notification Manager's notification queue. This is done by filling in the fields of a NMRec structure and passing its address to the routine NMInstall(). The values assigned to the fields of the NMRec structure determine how the user is notified. The relevant fields are:

```c
struct NMRec {
    QElemPtr qLink;
    short qType;   // queue type
    short nmFlags;
    long nmPrivate;
    short nmReserved;
    short nmMark;  // item to mark in Apple menu
    Handle nmIcon; // handle to small icon
    Handle nmSound; // handle to sound record
    StringPtr nmStr; // string to appear in alert
    NMUPP nmResp; // pointer to response routine
    long nmRefCon; // for application use
};
```

The `qType` field should be set to the enum constant `nmType` (a value of 8).

The `nmMark` field is set to 1 to place a diamond shaped mark next to the application name in the application menu. 0 indicates no mark. If a background-only application (such as a driver or a VBL task) issued the notification, this value should be 0.

The `nmIcon` field is used to place a blinking icon in the menu bar. It can be set to a handle to a small icon or to an icon family containing a small color icon to display a blinking cursor. If it is set to nil then a blinking icon is not displayed.

The `nmSound` field is used to play a sound resource during the notification. If it is set to a handle to a 'snd' resource, the sound is played. If it is set to nil, then no sound is played.
The `nmStr` field is used to display an alert. If it contains a pointer to a Pascal string, then an alert is displayed with this string. If it is set to nil, then an alert is not displayed.

The `nmResp` field is used to execute a response procedure. It can be set to a pointer to the procedure to be executed as the last step of the notification. If it is set to nil, then no procedure is executed. If `nmResp` is set to -1, then the notification request is automatically removed from the notification queue. Otherwise it must be manually removed with `NMRemove()`.

The `nmRefCon` data can be used to contain extra data. In 68K applications it can be set to a pointer to the application's AS world.

Once the fields of the NMRec structure are set to the proper values to define the types of notifications sent to the user, the function `NMInstall()` is called with a pointer to this structure. This places the notification in the notification queue to be executed.

When your application is switched back to the foreground it should remove the notification request from the notification queue with `NMRemove()`.

The routine below installs a notification that puts a diamond next the application's name and plays a sound.

```c
/******************
struct NMRec myNote;
void DoNotification(void)
{
    Handle hSound;
hSound = GetResource('snd ', 128);

    myNote.qType = nmType;
    myNote.nmMark = 1; //get mark in Apple menu
    myNote.nmIcon = nil; //no flashing Icon

    //get the sound you want out of your resources
    hSound = GetResource('snd ', 128);

    myNote.nmSound = hSound; //set the sound to be played
    myNote.nmStr = nil;       //no alert box
    myNote.nmResp = (NMUPP)nil; //no response proc
    myNote.nmRefCon = nil;
    NMInstall(&myNote);
}
```
When I have my notification set to notify the user by an alert or by playing a ‘snd‘ resource, it works fine. However, when I set it to notify the user by marking the application menu or blinking an icon, it doesn’t seem to work. Why?

You are probably removing the notification request from the queue too early. Make sure that the nmResp field in the notification record is not set to -1 and do not call NMRemove() to remove it until your application receives a resume event.

How can I tell if my notification was posted?

You can use a notification response procedure to determine of your notification was posted. The response procedure is executed as the final step of a notification. Have it set a global variable in your application to indicate that it has been run. To do this on a 68K Macintosh, you will have to set up the A5 world of the application.

Related FAQ

See also FAQ 19-12, “How do I get a notification to execute a response procedure?”

How can I get the Notification Manager to blink a color icon?

If you want your notification to blink a color icon, set the nmIcon field of the NMRec structure that is passed to NMInstall() to a handle to an icon family containing a small color icon. To blink a black-and-white icon, set the nmIcon field to a handle to a small icon.

When do I remove a notification record from the notification queue?

When and how you remove the notification record from the notification queue depends on how you are notifying the user. If you are using an audible or alert notification, then you can set the nmResp field of the notification record to -1. This
will cause the notification to be removed from the notification queue automatically as soon as the sound finishes playing or the user closes the alert box.

```c
//****************
void InstallSelfRemovingNotif()
{
    struct NMRec myNote;
    Handle hNotifSnd;

    hNotifSnd = GetResource('snd ', 128);
    myNote.qType = nmType;
    myNote.nmMark = 0;
    myNote.nlmIcon = nil;
    myNote.nmSound = hNotifSnd;
    myNote.nmStr = nil;
    myNote.nmRefCon = nil;

    // -1 causes request to be automatically removed
    myNote.nmResp = (struct RoutineDescriptor *) -1;
    NMInstall(&myNote);
}
```

However, if you are using a blinking icon or putting a mark in the application menu to notify the user, then you will not want to remove the notification from the queue until your application receives a resume event. Otherwise the Notification Manager would remove the diamond mark or the small icon before the user could see it. In this case, set the nmResp field of the notification record to 0 (or to a response procedure) and call NMRemove() upon receiving a resume event.

```c
//****************
void InstallPersistentNotif()
{
    struct NMRec myNote;
    Handle hNotifSnd;

    hNotifSnd = GetResource('snd ', 128);
    myNote.qType = nmType;
    myNote.nmMark = 1;
    myNote.nlmIcon = hIcon;
    myNote.nmSound = hNotifSnd;
    myNote.nmStr = nil;
    myNote.nmRefCon = nil;
```
CHAPTER 19: Processes: Notification Manager

// nil allows notification to remain in queue
myNote.nmResp = (struct RoutineDescriptor *)nil;

NMInstall(&myNote);

// Then in the event loop
void HandleEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
    case osEvt:
        if (eventPtr->message & 0x01000000)
        {
            // a resume event
            if (eventPtr->message & 0x00000001)
                NMRemove(&myNote);
        }
    }
}

If your application installed a notification request while it was in the background and receives a resume event before the notification is issued to the user, it is okay to remove the request from the queue with NMRemove().

Can I remove a notification I have installed before it is issued by the Notification Manager?

Yes, you can remove a notification request from the notification queue at any time with NMRemove(), even before the request is issued. Notification requests are not automatically removed from the queue by the Notification Manager after they are issued.

Related FAQ

See also FAQ 19-7, “When do I remove a notification record from the notification queue?”
Are issued notification requests automatically removed from the queue?

See FAQ 19-7, "When do I remove a notification record from the notification queue?"

Can I send a notification to a user from a interrupt task such as a VBL task or a Time Manager task?

Yes, you can install a notification request at any time, even from functions running at interrupt time such as VBL tasks and Time Manager tasks. A notification request can even be installed by an ‘INIT’ during system startup time.

Related Topics

See also Chapter 21, “Processes: Time Manager.”

See also Chapter 22, “Processes: Vertical Retrace Manager.”

Why does my program crash when it tries to issue a notification to the user?

Make sure that the notification record that you pass to NMInsta11() is not a local variable that is no longer valid. Make it a global variable. Otherwise the Notification Manager will attempt to read from the invalid structure.

If you are running a 68K application and your application is accessing application globals, make sure you are setting up the application’s AS world.

Notification Procedures

This section of the Processes: Notification Manager chapter answers FAQs on executing notification procedures. It discusses how notification procedures are created and how they access global variables. It also presents some common mistakes and how to avoid them.
How do I get a notification to execute a response procedure?

To get a notification to execute a notification response procedure the nmResp field of the notification record must be set to a universal procedure pointer of the procedure. The response procedure will be executed as the last step of the notification. This allows you to use the procedure to remove the notification request from the notification queue.

The notification procedure is defined as:

```pascal
void MyNotification(NMRecPtr nmReqPtr)
```

It gets passed to the notification record that was passed to NMInstall() to install the notification request.

A notification request procedure can access its application’s global variables. If it is running in a PowerPC native application it can automatically reference the globals. If the application is a 68K Macintosh application, the application's A5 world must be set up. The application's A5 value can be passed to the procedure in the nmRefCon field of the notification record.

Below is an example of 68K code that installs a notification that executes a notification procedure. The procedure calls SysBeep() and then increments a global variable.

```c
HANDLE hSound;
struct NMRec myNote;
NMUPP myNMUPP;
short myAppGlobal = 1;
InitNotif()
{
    myNMUPP = NewNMProc(MyNotification);
    hSound = GetResource('snd ', 128);
}

void A5AccessTest()
{
    myNote.qType = nmType;
    myNote.nmMark = 1;
    myNote.nmIcon = nil;
    myNote.nmSound = hSound;
```
Why does my machine hang when my application's notification procedure is activated a second time? The first time it is activated it runs fine.

Your notification request is most likely not being removed from the notification queue before you call NMInstall() again when it goes back into the background.

When your application comes to the foreground you need to remove your notification request from the notification queue by calling NMRemove(). The request will be automatically removed from the notification queue if the nmResp field of the notification record is set to -1. Setting this field to nil will not cause the record to be automatically removed from the queue.

How do I access my application's global variables from a notification procedure?

Notification routines running in PowerPC native applications automatically have access to the global variables of their installing process. They can reference the global variables as if they were regular subroutines.
To access an application's global variables from a notification procedure, you have to set up the application's AS world. When the application is preparing to install the notification request it should store its AS value in the \texttt{nmRefCon} field of the notification record. The notification procedure gets passed to this structure as a parameter. It can then extract the pointer to and set up the application's AS world, allowing it to access the application's globals.

\textbf{Related FAQ}

\hspace{1em} See also FAQ 19-12, "How do I get a notification to execute a response procedure?"
Processes: Process Manager

Another little known part of the Mac Toolbox is the Process Manager. Programmers who come from the Unix world are used to a strictly organized process structure. Until a few years ago, Mac programmers had no easy way to access the information relating to the processes running on their machine. With the advent of the Process Manager, Mac programmers now have a powerful mechanism for controlling and finding out about all running processes. In this chapter, you'll learn about the process list, the difference between the front process and the current process, and how you can walk through the list of all running processes. You'll also learn how to launch an application from your own application.

There are three subsections in this chapter:

- Definitions
- Using the Process Manager
- Launching Applications
Definitions

This section of the Processes: Process Manager chapter answers FAQs about some important terms that are discussed in the following sections. These terms are used in general discussions about processes.

What is an interrupt? What is interrupt time?

An interrupt is an exception that is signaled to the processor by a device, notifying the processor of some special occurrence. An example of this is the Vertical Blanking (VBL) interrupt which gets called each time the screen is refreshed. Interrupts can occur at any time, even when your application is in the middle of executing a call. Interrupt time is the period that a task is executed in response to an interrupt. The primary restriction at interrupt time is that you cannot execute any functions that may move memory.

What is the difference between the current process and the front process?

The current process is the process that is currently executing, for which the A5 world is set and is currently valid. The current process can be either in the background or foreground. The front process is the process that is in the foreground, the active application that is interacting with the user.

The serial number of the current process can be obtained by GetCurrentProcess(). The serial number of the front process can be obtained by GetFrontProcess().

What are background-only applications?

Background-only applications (BOAs), also called faceless background applications, are applications that do not have a user interface. Background-only applications do not call InitWindows() to initialize the Window Manager and have its onlyBackground flag in its ‘SIZE’ resource set. BOAs do not display any windows or a menu bar and are not listed in the Application menu.
BOAs run as normal processes. This gives them significant advantages over System extensions and other routines that are executed at interrupt time. They can launch applications, send and receive Apple events, and execute routines that move memory.

Using the Process Manager

This section of the Processes: Process Manager chapter answers FAQs about how to get various types of information about running processes. It discusses how to get a list of all processes currently being executed. It also lists and describes methods for determining the name and location of a given application.

How can I tell if my application is the front process?

You can tell if the application is the front process by comparing its process serial number, retrieved by calling GetCurrentProcess(), with the process serial number of the front application, retrieved by GetFrontProcess().

```plaintext
//***********************
Boolean IsFrontProcess()
{
    OSErr err;
    ProcessSerialNumber curSerialNum,
    frontSerialNum;
    err = GetCurrentProcess(&curSerialNum);
    if ( err != noErr)
        return false;
    err = GetFrontProcess( &frontSerialNum );
    if ( err != noErr)
        return false;
    if ( curSerialNum.highLongOfPSN ==
        frontSerialNum.highLongOfPSN &&
        curSerialNum.lowLongOfPSN ==
        frontSerialNum.lowLongOfPSN )
        return true;
    return false;
}
```
How do I get a list of all of the processes currently running?

To get a list of all the active processes call GetNextProcess(). To start at the front of the list of processes set the highLongOfPSN field of a ProcessSerialNumber variable to 0 and the lowLongOfPSN field to kNoProcess. Call GetNextProcess() until an error is returned. You can get information about the process by calling GetProcessInformation().

```c
void GetAllProcesses()
{
    OSErr err;
    ProcessSerialNumber serialNum;
    ProcessInfoRec info;
    Str255 processName;

    info.processInfoLength = sizeof(ProcessInfoRec);
    info.processAppSpec = nil;
    info.processName = processName;

    serialNum.highLongOfPSN = 0;
    serialNum.lowLongOfPSN = kNoProcess;

    while ( GetNextProcess(&serialNum) == noErr )
    {
        err = GetProcessInformation(&serialNum,&info);

        // do something with process information
    }
}
```

How can I find out if a specific process is running on my system and if it supports high-level events?

You can determine if a specific process is running by calling GetNextProcess() to iterate through all of the running processes.

You can determine if a process supports high-level events by examining its ‘SIZE’ resource. The ‘SIZE’ resource has a flag, isHighLevelEventAware, that is set
when an application can send and receive high-level events. The Toolbox function, GetProcessInfo(), retrieves information about the specified process in a ProcessInfoRec structure. The 'SIZE' resource flags are returned in the processMode field of this structure. Test this field against the highLevelEventMask mask. The following code shows how to test this:

```c
ProcessInfoRec myPinfoRec;
if (myPinfoRec.processMode & highLevelEventMask)
{
    // process is high level event aware
}
```

Related FAQ

See also FAQ 20-5, "How do I get a list of all of the processes currently running?"

**How can I get the name and location of my application?**

An application can get its own name by calling the Process Manager call GetProcessInformation() on itself. First you must get the process serial number of the application by calling GetCurrentProcess(). Then you fill in the necessary fields of the ProcessInfoRec structure. Set the processName field to point to a string, and set the processInfoLength to be the size of the ProcessInfoRec structure. To get the FSSpec of the application set the processAppSpec field to point to an FSSpec structure.

```c
ушрененей
OSErr GetAppName( Str255 processName,
                   FSSpec procFSSpec)
{
    OSErr err;
    ProcessSerialNumber serialNum;
    ProcessInfoRec info;

    err = GetCurrentProcess(&serialNum);
    if (noErr != err)
        return err;

    info.processInfoLength = sizeof( ProcessInfoRec );
```
Why won’t my application run when it is in the background?

The canBackground flag in the application’s ‘SIZE’ resource must be set for the application to be eligible to get CPU time when it is in the background.

The flags of an application’s ‘SIZE’ resource can be set when building the application through the development environment. As with any other resource, its fields can also be set with a resource editor.
Launching Applications

This section of the Processes: Process Manager chapter answers FAQs about using the routine LaunchApplication(). It describes how to use LaunchApplication as well as avoid some common mistakes.

**How can I get my application to be notified when an application that it launches with LaunchApplication() terminates?**

To have an application be notified when an application it has launched with LaunchApplication() terminates, set the acceptAppDiedEvents flag in the ‘SIZE’ resource. If this flag is set in the launching application and a launched application terminates, the launching application receives an Application Died Apple event ('aevt' 'obit').

**Why does my application terminate after I call LaunchApplication()?**

By using the launchControlFlags field of the LaunchParamBlockRec structure parameter, the programmer can control whether or not the launching application terminates when it launches an application. If this field has the launchContinue flag set, then the calling application does not terminate. Otherwise the calling process terminates after it launches the new process.

Related FAQ

See also FAQ 20-11, “How do I launch another application from my application?”

**How do I launch another application from my application?**

An application can launch another application by using the LaunchApplication() call. LaunchApplication() takes a pointer to a LaunchParamBlockRec structure as a parameter. The application to be launched is identified by an FSSpec
in the launchAppSpec field of the structure. The launchBlockID and launchEPBLength fields are set to extendedBlock and extendedBlockLen, values defined in the Processes.h header file.

LaunchApplication() allows you to specify a number of flags to control the execution of both the calling and called process in the launchControlFlags field. The launchContinue flag allows the calling application to continue to run after it launches the application. The launchDontSwitch flag allows your calling application to remain the front application with the launched application coming up in the background. The launchUseMinimum flag allows the application to be loaded in a size smaller than the preferred size if the preferred amount of memory is not available.

The following example launches the application called “testapp” located in the application’s directory, making it the front process and leaving the calling process running.

```c
//**********************
void LaunchTestApp()
{
    FSSpec testFSSpec;
    ProcessSerialNumber launchSerialNumber;

testFSSpec.vRefNum = 0;
testFSSpec.parID = 0;
pStrcpy( testFSSpec.name, "\ptestapp");

DoLaunchApp( &testFSSpec, &launchSerialNumber );
}

//******************
OSErr DoLaunchApp( FSSpec *appFSSpec, 
                 ProcessSerialNumber *launchSerialNumber )
{
    LaunchParamBlockRec launchParms;
    OSErr err;

    launchParms.launchBlockID = extendedBlock;
    launchParms.launchEPBLength = extendedBlockLen;
    launchParms.launchFileFlags = 0;
```
How do I launch an application with LaunchApplication(), but prevent it from becoming the front application?

Whether or not a launched application moves to the front when it is launched by LaunchApplication() can be controlled by the launchControlFlags field of the LaunchParamBlockRec structure parameter. If this field has the launchDontSwitch flag set, then the launched application is not brought to the front. Otherwise, by default, LaunchApplication() brings the launched application to the front.

Related FAQ

See also FAQ 20-11, “How do I launch another application from my application?”

Topic-Related FAQs

See also FAQ 7-24, “How can I tell when my application goes to the background and returns to the foreground?”

See also FAQ 7-25, “Why won’t my application receive suspend or resume events?”

See also FAQ 11-6, “How can I use Gestalt to communicate between processes?”

See also FAQ 21-4, “How can I determine the amount of time it takes for a routine to run?”
Processes: Time Manager

The Time Manager allows you to execute a function (that has already been loaded into memory) at a specified time. In this chapter, you'll learn the difference between a VBL task and a Time Manager task and when you should use the Time Manager. You'll learn how to install a Time Manager function in memory so that it sticks around even after your application quits, how to get a Time Manager task to run periodically, and how to start and stop a Time Manager task.

There are two subsections in this chapter:

- Using the Time Manager
- Executing Time Manager Tasks
Using the Time Manager

This section of the Processes: Time Manager chapter answers FAQs about how the Time Manager works. It contains information on the different versions of the Time Manager and how the Time Manager can be used to determine the execution time of a task.

When do I use VBL tasks, and when do I use the Time Manager?

Time Manager tasks allow you to execute tasks periodically at varying and extremely small delays. VBL tasks allow you to synchronize tasks with the refresh rate of the screen. Under certain conditions, the system can turn off vertical blanking interrupts.

Use Time Manager tasks if you want to execute tasks with very small delays, with delays other than those provided by the screen refresh rate, or at precise time intervals. Use VBL tasks if you want to synchronize actions with the refresh rate of the screen such as for animation and screen updating.

Related FAQs

- See also FAQ 3-12, “How do I put a spinning cursor in my program?”
- See also FAQ 3-13, “How can I spin the cursor using an interrupt routine?”
- See also FAQ 3-14, “Why does my VBL task crash when I try to spin a cursor?”

Related Topic

- See also Chapter 22, “Processes: Vertical Retrace Manager.”
**What is the finest resolution of time delay in Time Manager?**

The original Time Manager can execute tasks with delays as small as 1 millisecond. The Revised Time Manager and the Extended Time Manager can execute tasks with delays as small as 20 microseconds. (The delay is much larger if you use drift-free, fixed-frequency capabilities.)

If it is important to assure the accuracy of the delay at which your Time Manager task executes, you should use the drift-free, fixed-frequency capabilities available in the Extended Time Manager. The Extended Time Manager allows a task to be installed with an execution time relative to when the task last expired rather than relative to when it was installed, which compensates for overhead time used by the Time Manager and the unpredictable delays of the interrupts.

![Figure 21-1](image1.png)

*Figure 21-1: The original Time Manager does not compensate for any of its processing overhead or delays in the interrupts. Since the execution time is relative to the time when the task is installed, the delays cause the execution time to drift.*

![Figure 21-2](image2.png)

*Figure 21-2: The extended Time Manager compensates for any delay between the time a task expires and the time it is reinstalled. The execution time is relative to the time when the task last expired, providing drift free execution.*

**Related FAQ**

- See also FAQ 21-5, "What is the difference between InsTime() and InsXT ime()?"
How do I specify a period of less than one millisecond?

In the original version of the Time Manager the smallest value that you could specify was 1 millisecond. The revised Time Manager and the extended Time Manager allow the representation of both microseconds and milliseconds. A positive delay value is interpreted as milliseconds. A negative delay value is interpreted as microseconds.

How can I determine the amount of time it takes for a routine to run?

You can determine the time a routine takes to run, its elapsed time, with Time Manager. When RmvTime() is called to remove a task that has not yet been executed from the Time Manager queue, it returns in the tmCount field of the task record the amount of unused time. To measure the time, create a task structure with the tmAddr field set to nil. This means no task will execute if the time period expires. Install the Time Manager task with InsTime() and activate it with PrimeTime(). If you want a duration count in microseconds then pass a large negative number to the msCount parameter of PrimeTime(). This will allow you to measure elapsed times of up to about 35 minutes. If you need a longer time, you can pass a large positive number to PrimeTime() which will be treated as milliseconds. However, this will give you elapsed time in milliseconds. Execute the functions you want to measure. Then call RmvTime(). The elapsed time is the difference between the starting time specified in PrimeTime() and the remaining time specified in tmCount field.

To get an even more accurate measurement of the required time, you can determine the amount of time overhead associated with calling the Time Manager and then subtract this from the elapsed time of your task. To determine the overhead time call InsTime(), PrimeTime(), and then immediately call RmvTime().

The example below determines the Time Manager overhead and then calculates the time required for the routine Delay(120, &finalTicks) to run.

```
//***************
define kStartTime -2000000000

void ComputeElapsedTime()
{

```
TMTask myTask;
long finalTicks;
long elapsedTime;
long TMOverhead;

// set up task structure - note tmAddr is nil
myTask.tmAddr = nil;
myTask.tmWakeUp = 0;
myTask.tmReserved = 0;

// get Time Manager overhead
InsTime((QElemPtr) &myTask);
PrimeTime((QElemPtr)&myTask, kStartTime);
RmvTime((QElemPtr)&myTask);

TMOverhead = myTask.tmCount - kStartTime;

// determine elapsed time of calls
InsTime((QElemPtr)&myTask);
PrimeTime((QElemPtr)&myTask, kStartTime);

//call to measure time for
Delay(120, &finalTicks);
RmvTime((QElemPtr)&myTask);

if (myTask.tmCount < 0 )

    elapsedTime = myTask.tmCount - kStartTime -
    TMOverhead;
else
    elapsedTime = (myTask.tmCount * 1000) - kStartTime -
    TMOverhead;

Related Topic

See also Chapter 20, “Processes: Process Manager.”
Executing Time Manager Tasks

This section of the Processes: Time Manager chapter answers FAQs about executing Time Manager tasks. It discusses the subtle differences between creating a Time Manager task on a 68K and on the Power Macintosh. It also provides examples of how to create Time Manager tasks that execute continuously and tasks that execute after the installing application quits.

What is the difference between InsTime() and InsXTime()?

Both functions install a task into the Time Manager queue. InsXTime() allows you to take advantage of the drift-free fixed-frequency timing that provides for the accurate execution of the task promptly after the specified time delay. InsTime() allows you to execute tasks with a much smaller delay.

Related FAQ

See also FAQ 21-2, "What is the finest resolution of time delay in Time Manager?"

Why does my system crash when I exit my application after installing a periodic Time Manager task?

Make sure that you have loaded your Time Manager task in the System heap. If the routine is located in your application’s heap, your application must still be active when the routine is executed. If your application is going to exit before the specified delay for the routine to execute, then remove the task from the Time Manager queue by calling RmvTime() before it terminates. If you want your application to exit after it has installed and activated a Time Manager task but before the task executes, load the routine into the System heap.

If your Time Manager task accesses the application’s global (by restoring the application’s A5 world), then it must not execute after the application quits.
**How do I get a Time Manager task to execute persistently, execute when the application that installed it is switched out and is no longer in control of the CPU?**

Time Manager tasks run persistently by default. Unlike VBL tasks, Time Manager tasks will run even when the installing application is not in control of the CPU. As long as the installing application is still running, the task does not have to be loaded into the System heap.

**How do I execute a Time Manager task in a 68K application?**

A Time Manager task is executed by creating a TMTask structure pointing to it, installing the structure into the Task Manager queue with InsTime() (or InsXTime()), and activating the timer to run the task with PrimeTime().

If you want the Time Manager task to access the application’s globals or reactivate itself to run periodically, then you must pass it a pointer to the application’s AS world. Since there is not an extra field in TMTask structure, the pointer is put in a structure behind the TMTask structure. This structure is used with InsTime() and PrimeTime(). In a 68K application, the Time Manager task has access to this structure only through the A1 register. The inline assembly routine

```pascal
pascal TMinfoPtr GetTMinfo (void) = 0x2E89;
```

retrieves the data from the A1 register. The Task Manager routine now has access to the application’s AS world to access globals and to the original TMTask structure so it can reactivate itself by calling PrimeTime().

The VBL task must be removed with RmvTime() when the application exits if it is loaded in the applications heap or if it accesses any globals of the application.

The code below installs the Time Manager task, MyTask(), into the Time Manager queue and activates it. MyTask() gets the TMinfo structure off of the A1 register. From the tmRefCon field it retrieves the pointer to and sets up the application’s AS world. MyTask() then accesses the application’s global variable, myAppGlobal, and increments it. As the last step, MyTask() reactivates itself by extracting the TMTask structure from the data pulled off of the A1 register and calling PrimeTime().
```c
#define kDelay 2000

short myAppGlobal = 1;

// structure to piggy back pointer to application's
// A5 world pointer on task structure to pass to
// Time Manager task
typedef struct
{
    TMTask atmTask;
    long tmRefCon;
} TMinfo, *TMinfoPtr;

TMinfo myTMinfo;

pascal TMinfoPtr GetTMinfo (void) = Ox2E89;    // MOVE.L A1, (SP)

void InstallTMTask (void)
{
    myTMinfo.atmTask.tmAddr = MyTask;
    myTMinfo.atmTask.tmWakeUp = 0;
    myTMinfo.atmTask.tmReserved = 0;
    // store pointer to application's A5 world
    myTMinfo.tmRefCon = SetCurrentA5();

    InsTime((QElemPtr) &myTMinfo);
    PrimeTime((QElemPtr) &myTMinfo, kDelay);
}

pascal void MyTask()
{
    long oldA5;
    TMinfoPtr myPtr;
```
myPtr = GetTMInfo();
// setup app's A5 world
oldA5 = SetA5(myPtr->tmRefCon);

// process the application's globals in here
myAppGlobal++;

oldA5 = SetA5(oldA5);

// reactive the task to run again
PrimeTime((QElemPtr) myPtr, kDelay);
}

How do I execute a Time Manager task in a native PowerPC application?

A Time Manager task is executed by creating a TMTask structure pointing to it, installing the structure into the Task Manager queue with InsTime() (or InsXTime()), and activating the timer to run the task with PrimeTime().

Unlike a 68K application, the Time Manager task can automatically access the application's globals, and the structure passed to InsTime() and PrimeTime() in the installing application is passed as a parameter to the Time Manager task. If you want the Time Manager task to reactivate itself to run periodically, it must take this structure and call PrimeTime().

The Time Manager task must be removed with RmvTime() when the application exits if it is loaded in the application's heap or if it accesses any globals of the application.

The code below installs the Time Manager task, MyTask(), into the Time Manager queue and activates it. MyTask() increments the application's global variable, myAppGlobal. As the last step, MyTask() reactivates itself by taking the TMTask structure passed to it as a parameter and calling PrimeTime().

//****************************************************************************
#define kDelay 2000

short myAppGlobal = 1;

TMTask myTmTask:

```c
/****************
void InstallTMTask (void)
{
    myTmTask.tmAddr = NewTimerProc(MyTask);
    myTmTask.tmWakeUp = 0;
    myTmTask.tmReserved = 0;

    InsTime((QElemPtr) &myTmTask);
    PrimeTime((QElemPtr) &myTmTask, kDelay);
}

/****************
pascal void MyTask(TMTask *myPtr)
{
    myAppGlobal++;

    PrimeTime( (QElemPtr) myPtr, kDelay);
}

Related Topic

See also Chapter 17, “Power Macintosh Programming.”
```

**Do I need to reinstall a Task Manager task with InsTime each time I want activate it with PrimeTime?**

No, a Task Manager task only needs to by installed by calling InsTime() or InsXTime() once. It can be activated by PrimeTime() repeatedly until it is removed from the Task Manager queue by RmvTime().
CHAPTER 21: Processes: Time Manager

How do I get my Time Manager task to run periodically?

A Time Manager task only gets executed once when it is activated by PrimeTime(). To run periodically, the Time Manager must get a pointer to the original TMTask structure and reactivate itself by calling PrimeTime().

In a 68K application, the Time Manager task can get the TMTask structure that was set up by its installing application from the A1 register.

Unlike 68K applications, the structure passed to PrimeTime() in a native PowerPC application is accessible to the Time Manager task as a parameter.

Related FAQs

- See also FAQ 21-14, “How do I access application globals from my 68K Time Manager task?”
- See also FAQ 21-15, “How do I access application globals from a native PowerPC Time Manager task?”

How do I leave a Time Manager task running after my application quits?

A Time Manager task remains in the Time Manager queue until it is removed by the RmvTime() command, even after the installing application quits. In order for the task to run after the application quits, both the Time Manager task code AND the timer structure must be loaded into the System heap. Otherwise, when the application terminates, the memory in the application heap (which contains the timer structure and the timer function code) will be released.

How do I stop a periodic Time Manager task from running?

A Time Manager task runs continuously by reinstalling itself into the queue. You can remove a task from the queue, and consequently prevent it from reinstalling itself, by calling RmvTime().
How do I access application globals from my 68K Time Manager task?

For a Time Manager task to access the global variables of its installing application, the A5 world of the application must be restored in the Time Manager task. To do this, the pointer to the application’s A5 world must be passed to the Time Manager task. This is not a trivial matter. First, there is no extra field in the TMTask structure in which the pointer can be placed. A new structure must be created consisting of the TMTask structure as its first field and the A5 pointer in the second field. This must then be passed to InsTime() (or InsXTime()) and PrimeTime().

```
typedef struct
{
    TMTask  atmTask;
    long    tmRefCon; // holds pointer to A5 world
}
TMInfo, *TMInfoPtr;
```

So far, this is not too difficult. The problem comes in when trying to access this structure from within the Time Manager task function itself. It is not passed as a parameter; instead it is passed to the function in the A1 register. The structure can be retrieved by using the following inline assembly routine:

```
pascal MyTMInfoPtr GetTMInfo (void)
    = 0x2E89;
```

The Time Manager task function would call this assembly routine as follows:

```
MyTMInfoPtr myPtr;

myPtr = GetTMInfo();
```

The Time Manager task function now has access to the pointer to its application’s A5 world that it can restore with SetA5(). It also has a pointer to the original TMTask structure so the function can reactivate itself with the PrimeTime() call for continuous execution. Below is sample code from a Time Manager task:
pascal void MyTask()
{
    long oldA5;
    TMInfoPtr myPtr;

    myPtr = GetTMInfo();
    oldA5 = SetA5(myPtr->tmRefCon);
    ...
    // process application's globals
    ...
    oldA5 = SetA5(oldA5);    // restore original A5

    PrimeTime((QElemPtr) myPtr, kDelay);
}

Related FAQs

- See also FAQ 21-11, “How do I get my Time Manager task to run periodically?”
- See also FAQ 21-8, “How do I execute a Time Manager task in a 68K application?”

---

**How do I access application globals from a native PowerPC Time Manager task?**

Accessing the global variables in a native PowerPC Time Manager task is easier than doing so in a 68K application. Unlike a 68K application, the PowerPC application’s A5 world does not have to be restored in the Time Manager task. The Task Manager can access the application’s globals directly.

Related FAQs

- See also FAQ 21-9, “How do I execute a Time Manager task in a native PowerPC application?”
- See also FAQ 21-11, “How do I get my Time Manager task to run periodically?”
Related Topic

☐ See also Chapter 17, "Power Macintosh Programming."

Topic-Related FAQs

☐ See also FAQ 7-29, "How do I delay my program for a time that will be consistent on all machines?"

☐ See also FAQ 19-10, "Can I send a notification to a user from a interrupt task such as a VBL task or a Time Manager task?"
Though the image on your screen looks rock-solid, it is actually being constantly redrawn, one horizontal line of pixels at a time, from top to bottom, till the raster gets to the lower right corner of the screen, where the process starts over again. As the electron beam is being readjusted from the lower-right to upper-left corner, the processor has some free time to take care of some housekeeping. This interval of time is known as the vertical retrace or vertical blanking (VBL) interval. The Vertical Retrace Manager allows you to install a task that gets executed during the vertical blanking interval. The questions in this chapter explore some of the issues surrounding VBL programming.

There are two subtopics in the chapter:

✧ Definitions
✧ Using the Vertical Retrace Manager
Definitions

This section of the Processes: Vertical Retrace Manager chapter answers FAQs about some important terms that are discussed in the following sections. It describes the concept of the VBL and points out the differences between the two types of VBL tasks.

What is a Vertical Blanking (VBL) interrupt?

The Vertical Blanking interrupt is generated by the video circuitry upon the completion of a single screen refresh. To refresh the screen, the monitor's electron beam draws the screen pixels starting at the upper-left corner and moving to the lower-right corner. For built-in monitors this process is repeated approximately 60 times a second. For external monitors the refresh rate is determined by the video hardware.

The Operating System performs several tasks during VBL interrupts including moving the cursor in response to the user moving the mouse, updating the global variable TickCount, and checking if the user inserted a disk.

What is the difference between a system-based VBL and a slot-based VBL?

A system-based VBL task is not linked to an external video device. It is executed approximately every 60.15 times a second and is triggered by a special interrupt that mimics the vertical retrace interrupt on compact Macintosh models. By default a system-based VBL task does not run when its application is not the current process.

A slot-based VBL task is linked to an external video monitor and is executed whenever a VBL interrupt occurs for that device. The frequency at which a slot-based VBL task is executed is dependent upon the refresh rate of the monitor. Different monitors may have different refresh rates resulting in different rates of execution of their corresponding VBL tasks. Consequently the Vertical Retrace Manager maintains a separate task queue for each video device. Slot-based VBL tasks are always persistent (continue to be executed even when the Process Manager switches the application out of context). If you need to synchronize the execution of your VBL task with the retracing of the screen (to avoid flickering) use a slot-based VBL task. Slot-based VBL tasks always run, even when their applications are not the current process.
CHAPTER 22: Processes: Vertical Retrace Manager

Using the Vertical Retrace Manager

This section of the Processes: Vertical Retrace Manager chapter answers FAQs on using VBL tasks. It describes how to set up a VBL task to run when its application goes into the background, how to run continuously, and how to run after its installing application has quit. There are subtle differences between executing VBL tasks on a 68K Macintosh and executing them on a PowerMac. These issues are dealt with here. It also answers a FAQ on how to use VBL tasks to prevent flicker in animation.

How often will a VBL task get executed?

The system VBL interrupt gets generated at approximately 60.15 Hz, resulting in a period of about 16.63 milliseconds. The period of slot interrupts depends upon the associated video hardware (it will be close to that of the system interrupt). You can specify the number of interrupts that are to be generated before your VBL task will be executed. If you set the vb1Count to 1, your task will get executed with each interrupt. If you set the vb1Count to 10 your task will be generated every 10 interrupts, resulting in a period of about 166.3 milliseconds.

How do I execute a 68K VBL task?

A VBL task is executed by creating a VBLTask structure with the vb1Addr field set to a pointer to the VBL task function and the vb1Count field set to the number of VBL interrupts that must be generated before the task will run. The structure is then installed into the VBL queue by Vinstall().

If you want the VBL task to access the application's globals or reactivate itself to run periodically, then you must pass it a pointer to the application's A5 world. Since there is not an extra field in VBLTask structure the pointer is put in a user-defined structure consisting of the TMTask structure and the A5 world pointer. This structure is used with the initial call Vinstall(). In a 68K application, the VBL task has access to this structure only through the A0 register. The inline assembly routine

```pascal
pascal VBLInfoPtr GetVBLInfo (void)
   = 0x2E88;
```

retrieves the data from the A0 register. The VBL routine now has access to the application's A5 world to access globals and to the original VBLTask structure so it can reset the vb1Count field to be executed again.
The VBL task must be removed with VRemove() when the application exits if it is loaded in the application’s heap or if it accesses any globals of the application.

The code below installs the VBL task, MyVBLTask(), into the VBL queue. MyVBLTask() gets the VBLInfo structure off of the A0 register. From the vblA5 field it retrieves the pointer to and sets up the application’s A5 world. MyVBLTask() then accesses the application’s global variable, myAppGlobal, and increments it. As the last step, MyVBLTask() extracts the VBLTask structure from the data pulled off of the A0 register and resets the vblCount field so it will execute again.

```
// number of vbl interrupts before task executes
#define kInterval 6

short myAppGlobal = 1;

// structure to piggy back pointer to application's
// A5 world pointer on task structure to pass to
// VBL task
typedef struct VBLInfo
{
    VBLTask myVBLTask;
    long vblA5;
} VBLInfo, *VBLInfoPtr;

VBLInfo myVBLInfo;

pascal long GetVBLInfo (void)
    = Ox2E88;     // MOVE.L A0,(SP)

void InstallVBL ()
{
    myVBLInfo.myVBLTask.qType = vType;
    myVBLInfo.myVBLTask.vblAddr = NewVBLProc( MyVBLTask);
    myVBLInfo.myVBLTask.vblCount = kInterval;
    // store pointer to application's A5 world
    myVBLInfo.vblA5 = SetCurrentA5();

    VInstall((QElemPtr) &myVBLInfo.myVBLTask);
}
How do I execute a PowerPC native VBL task?

A VBL task is executed by creating a VBLTask structure with the vblAddr field set to a pointer to the VBL task function and the vblCount field set to the number of VBL interrupts that must be generated before the task will run. The structure is then installed into the VBL queue by Vinstall().

Unlike the case in 68K applications, the PowerPC native VBL task can automatically access the application’s globals, and the structure passed to Vinstall() in the installing application is passed as a parameter to the VBL task. If you want the VBL task to run periodically, it must take this structure and reset the vblCount field.

The VBL task must be removed with VRemove() when the application exits if it is loaded in the application’s heap or if it accesses any globals of the application.

The code below installs the VBL task, MyVBLTask(), into the VBL queue. MyVBLTask() increments the applications’ global variable, myAppGlobal. As the last step, MyVBLTask() resets the vblCount field so it will execute again.

```c
//*******************************************************
#define kInterval 6

short myAppGlobal = 1;
```
VBLTask myVBLTask;

// ***********************
void InstallVBL ()
{
    myVBLTask.qType = vType;
    myVBLTask.vblAddr = NewVBLProc( MyVBLTask);
    myVBLTask.vblCount = kInterval;

    VInstall( (QElemPtr)&myVBLTask );
}

// ***********************
void MyVBLTask( VBLTask *recPtr)
{
    myAppGlobal++;

    /* Reset vblCount so that this procedure executes again */
    recPtr->vblCount = kInterval;
}

Related Topic
See also Chapter 17, "Power Macintosh Programming."

**How do I execute a slot-based VBL task?**

A slot-based VBL task is executed the same way that a system-based VBL task is, except that the VInstall() and the VRemove() call are replaced by SlotVInstall() and SlotVRemove(). These two slot-based VBL calls take an additional parameter, the slot number of the slot whose queue the task should be installed in. The slot number of a screen can be determined by the following routine:

//***************
short GetSlotNumber( GDHandle devHandle)
{
    short slotNumber;
    AuxDCEHandle myHandle;

    myHandle = (AuxDCEHandle)GetDCtlEntry( (*devHandle)->gdRefNum );
    slotNumber = (*myHandle)->dCtlSlot;
The code below installs a slot-based VBL function:

```c
//****************
void InstallSlotVBL ( short slotNumber)
{
    gMyVBLInfo.myVBLTask.qType = vType;
    gMyVBLInfo.myVBLTask.vblAddr = NewVBLProc( DoVBL);
    gMyVBLInfo.myVBLTask.vblCount = kInterval;
    gMyVBLInfo.vblA5 = SetCurrentA5();

    SlotVInstall((QElemPtr) &gMyVBLInfo.myVBLTask,
                       slotNumber);
}
```

**How can I get my VBL task to execute after my application quits?**

For a VBL task to run after the installing application quits, both the VBL task and task record must be loaded into the System heap and the VBL task cannot be accessing any of the application's globals.

**How do I get a VBL task to execute persistently, execute when the application is switched out and is no longer in control of the CPU?**

A slot-based VBL task is never disabled. It runs whether or not the calling application is the current application or in the background.

A system-based VBL task is automatically disabled by the Process Manager when the installing application goes to the background if the task is loaded into the application memory. To make the task run persistently, it must be loaded into the System heap. One way to accomplish this is to allocate a block of nonrelocatable memory in the System heap to hold an Assembly language JMP instruction and the address of the original VBL task. The vblAddr of the VBLTask structure is set to the address of this System memory.
The sample code below allocates a block of memory on the System heap and sets it to the JMP instruction and the address of the VBL task MyVBLTask(). The task is then installed with \texttt{Vinstall()}. MyVBLTask() runs even when the installing application is in the background.

```c
//*******************************************************************************
#define kJMPInstruction 0x4EF9

void InstallPersistentVBL ()
{
    short *SysHeapPtr;
    long *TempPtr;

    SysHeapPtr = (short *)NewPtrSysClear(6);
    // JMP instruction
    *SysHeapPtr = (short)kJMPInstruction;

    TempPtr = (long *) (SysHeapPtr + 1);
    *TempPtr = (long)MyVBLTask;

    myVBLInfo.myVBLTask.qType = vType;
    myVBLInfo.myVBLTask.vblAddr = NewVBLProc( SysHeapPtr);
    myVBLInfo.myVBLTask.vblCount = kInterval;
    // store pointer to application's A5 world
    myVBLInfo.vblA5 = SetCurrentA5();

    VInstall((QElemPtr) &myVBLInfo.myVBLTask);
}
```

Another method of creating a persistent VBL task is to create a ‘CODE’ resource, set the System heap and locked bits, and load it into the application that installs the VBL. The code below loads in a ‘CODE’ resource containing the VBL task. Data is shared between the VBL task and the application by piggy backing the data on the VBLTask that is passed to \texttt{Vinstall()}.

```c
//*******************************************************************************

typedef struct VBLInfo
{
    VBLTask    myVBLTask;
    long       vblA5;
    short      myAppGlobal;
}
VBLInfo, *VBLInfoPtr;

VBLInfoPtr myVBLInfoPtr;
Ptr SysHeapPtr;

//***************
void InstallVBL ()
{
    Handle hCode;

    hCode = GetResource('CODE', 1000);

    SetZone(SystemZone());
    HLock(hCode);
    myVBLInfoPtr = (VBLInfoPtr)NewPtr(sizeof(VBLInfo));
    SetZone(ApplicationZone());

    myVBLInfoPtr->myVBLTask.qType = vType;
    myVBLInfoPtr->myVBLTask.vblAddr = NewVBLProc( *hCode );
    myVBLInfoPtr->myVBLTask.vblCount = kInterval;
    // store pointer to application's A5 world
    myVBLInfoPtr->vblA5 = SetCurrentA5();

    VInstall((QElemPtr) &CmyVBLInfoPtr->myVBLTask));
}

**How do I make my VBL task run periodically?**

To run periodically, the VBL task must get a pointer to the original VBLTask structure and reset the vblCount field. The VBL task gets executed when the vblCount field gets decremented 0. Resetting this value will cause the task to get executed again when the count reaches 0.

In a 68K application the VBL task can get the VBLTask structure that was set up by its installing application from the A0 register.

Unlike 68K applications, the structure passed to VInstall() in a native PowerPC application is accessible to the VBL task as a parameter.

**Related FAQs**

See also FAQ 22-10, “How do I access application globals from my 68K VBL task?”
See also FAQ 22-11, “How do I access application globals from a native PowerPC VBL task?”

How do I access application globals from my 68K VBL task?

For a VBL task to access the global variables of its installing application, the A5 world of the application must be restored in the VBL task. To do this, the pointer to the application’s A5 world must be passed to the VBL task. This is not a trivial matter. First, there is no extra field in the VBLTask structure in which the pointer can be placed. A new structure must be created consisting of the VBLTask structure as its first field and the A5 pointer in the second field. This must then be passed to Vinstall().

```c
typedef struct VBLInfo {
    VBLTask myVBLTask;
    long vblA5; // holds pointer to A5 world
} VBLInfo, *VBLInfoPtr;
```

So far, this is not too difficult. Problems arise when trying to access this structure from within the VBL task function itself. It is not passed as a parameter; instead it is passed to the function in the A0 register. The structure can be retrieved by using the following inline assembly routine:

```assembly
pascal long GetVBLInfo (void) = 0x2E88;
```

The VBL task function would call this assembly routine as follows:

```c
VBLInfoPtr recPtr;
recPtr = (VBLInfoPtr) GetVBLInfo();
```

The VBL task function now has access to the pointer to its application’s A5 world that it can restore with SetA5(). It also has a pointer to the original VBLTask structure so the function can reset the vblCount field for continuous execution. Below is sample code from a VBL task:

```c
void MyVBLTask() {
    long curA5;
    VBLInfoPtr recPtr;
```
recPtr = (VBLInfoPtr) GetVBLInfo();
//setup app's A5 world
curA5 = SetA5 (recPtr->vblA5);
...
// process the application's globals in here
...
curA5 = SetA5(curA5);

// Reset vblCount so VBL task runs again
recPtr->myVBLTask.vblCount = kInterval;

Related FAQs
- See also FAQ 22-9, “How do I make my VBL task run periodically?”
- See also FAQ 22-11, “How do I access application globals from a native PowerPC VBL task?”

**How do I access application globals from a native PowerPC VBL task?**

Accessing the application’s global variables in a native PowerPC VBL task is easier than accessing them in a 68K application. Unlike a 68K application, the PowerPC application’s A5 world does not have to be restored in the VBL task. The VBL task can access the application’s globals directly.

Related FAQs
- See also FAQ 22-9, “How do I make my VBL task run periodically?”
- See also FAQ 22-10, “How do I access application globals from my 68K VBL task?”

Related Topic
- See also Chapter 17, “Power Macintosh Programming.”
How do I determine the slot number of a screen?

The slot number of a monitor can be found using the `GetDCtlEntry()` call to get a handle to a device control entry. The slot number is found in the `dCtlSlot` field. Below is a routine that takes a screen's device handle and returns the slot number.

```c
short GetSlotNumber( GDHandle devHandle)
{
    short slotNumber;
    AuxDCEHandle myHandle;

    myHandle = (AuxDCEHandle)GetDCtlEntry((*devHandle)->gdRefNum);
    slotNumber = (*myHandle)->dCtlSlot;

    return(slotNumber);
}
```

Related Topics

See also Chapter 12, "Graphic Devices."

How do I get the slot number for the main monitor?

To get the slot number for the main screen you first must get a handle to its `gDevice` structure with `GetMainScreen()`. Then you get its device control entry by passing the `gdRefNum` field of the main `gDevice` structure to `GetDCtlEntry()`. The slot number is `dCtlSlot` field of the Device Control Entry.

The example below returns the slot number of the main screen:

```c
short GetMainSlotNumber()
{
    short slotNumber;
    GDHandle mainDevice;
    AuxDCEHandle myHandle;

    mainDevice = GetMainDevice();
    myHandle = (AuxDCEHandle)GetDCtlEntry((*mainDevice)->gdRefNum);
```
slotNumber = (*myHandle)->dCtlSlot;

return(slotNumber);

Related FAQ
See FAQ 30-15, "How can I tell if a System routine is available in my system?"

How can I disable a VBL task?

During each VBL interrupt, the Vertical Retrace Manager reduces the vblCount field by 1. When the value is decremented to 0 the VBL task is executed. The task will not execute again unless the vblCount is reset. You can disable a VBL task before it executes by setting the vblCount field of the VBLTask structure to 0. The task record still remains in the VBL queue.

You can remove a VBL task from the VBL queue, and prevent it from executing, by calling VRemove() for system-based VBL tasks or SlotVRemove() for slot-based VBL tasks.

How do I synchronize my drawing to a VBL for smooth animation?

Synchronizing your drawing to a VBL prevents a visual effect called tearing. Tearing occurs when you blit pixels in the normal top-to-bottom order, and the blitter overtakes the scanning beam. The pixels that were blitted above the scanning beam don’t get refreshed until the scanning beam makes another pass. The pixels below the scanning beam get refreshed immediately as the scanning beam passes over them. The result is that for approximately one tick (the period between scanning passes), the old pixels will be visible in the upper half of the blitted image and the new pixels in the lower half of the image.

Flickering is another usually undesirable visual effect of drawing miscalculations. Flickering occurs when your drawing code changes the same pixels twice per frame and those pixels get refreshed by the scanning beam in between the two drawing actions. Flicker can be eliminated by using an offscreen GWorld to blit images to the screen. However, using a GWorld will not prevent tearing.

To eliminate tearing you must synchronize your drawing to the vertical retrace
blanking interrupt (VBL) of the monitor. This involves installing a slot VBL task. Make sure that you use a slot-based VBL task and not a system-based VBL task. A slot-based VBL task is executed in sync with the actual screen refreshing of the monitor associated with the slot. A system-based VBL task is executed approximately every 1/60th of a second, regardless of the screen’s refresh rate. If your monitor’s display gets refreshed every 1/72 of a second, synchronizing your drawing to the system-based VBL task will most likely result in tearing.

To synchronize drawing to the refreshing of your screen, install a slot-based VBL task with SlotInstall(). Have the VBL task increment a global variable of the application that will do the drawing. Have your drawing operation (this should be a bit blitting operation) test the value of the global variable, waiting for it to be changed by the VBL task. When the VBL task increments the variable, your main routine should blit the new image to the screen.

Related Topic

☐ See also Chapter 23, “QuickDraw: Drawing.”

Topic-Related FAQs

☐ See also FAQ 3-13, “How can I spin the cursor using an interrupt routine?”

☐ See also FAQ 3-14, “Why does my VBL task crash when I try to spin a cursor?”

☐ See also FAQ 19-10, “Can I send a notification to a user from a interrupt task such as a VBL task or a Time Manager task?”

☐ See also FAQ 21-1, “When do I use VBL tasks, and when do I use the Time Manager?”
QuickDraw: Drawing

Drawing with the QuickDraw Toolbox functions is one of the basic components of any Macintosh programmer's training. There are, however, many features that are not covered by most of the standard QuickDraw programming texts. The questions in this chapter touch upon the most significant of these overlooked aspects of QuickDraw, for example: drawing dashed lines; drawing outside of a window on the Desktop; the QuickDraw bottleneck routines; highlighting techniques; offscreen bitmaps and pixmaps, and more.

There are six subsections in this chapter:

- Definitions
- Drawing
- Bitmaps and Pixmaps
- GWorlds and Offscreen Drawing
- Marquees and Selection Lines
- Special Drawing
Definitions

This section of the QuickDraw: Drawing chapter defines some important terms related to drawing on the Macintosh. It describes two fundamental objects, the Gworld and the gray region, and introduces the complex topic of bottleneck functions.

What is a GWorld?

A GWorld is an offscreen graphics environment that consists of, among other things, a CGrafPort, a pixmap, a Color Table, and a GDevice. It is used to prepare complex images that can be blitted to the display screen to create the effect of instantaneous drawing.

What is the gray region?

The gray region is the region that represents all of the available desktop area including the rounded rectangular areas on all active screens except for the area under the menu bar on the main screen. A handle to the current gray region is stored in the global variable GrayRgn and can be retrieved by calling the GetGrayRgn() call.

What are the QuickDraw “bottleneck” functions?

The QuickDraw “bottleneck” functions are low level routines each called by a number of high level functions to perform basic graphics operations. These routines include those that paint, erase, frame, invert, and fill a shape; the routine called by CopyBits() to perform bit and pixel transfer; the routine that measures the width of text and is called by CharWidth, StringWidth, and TextWidth; the routine that processes picture comments; and the routines that save drawing commands to define a picture and retrieve them.

These low-level functions can be replaced by a user-defined function that will be called in its place. The user-defined function can then either call the standard low level routine or return.

SetStdProcs() and SetStdCProcs() can be used to replace QuickDraw’s standard low level functions. GrafPort and CGrafPort both contain a grafProcs field which is a pointer to a structure of pointers to customized drawing routines.
The standard QuickDraw low-level functions are:

- StdText: Draws text
- StdLine: Draws a line
- StdRect: Draws a rectangle
- StdRRect: Draws a rounded rectangle
- StdOval: Draws an oval
- StdArc: Draws an arc or a wedge
- StdPoly: Draws a polygon
- StdRgn: Draws a region
- StdBits: Does bit and pixel transfer
- StdComment: Processes a picture comment
- StdTxMeas: Measures text width
- StdGetPic: Retrieves information from the definition of a picture
- StdPutPic: Saves information as the definition of a picture

Related FAQs

See also FAQ 6-28, “How do you deactivate (gray-out) static text items and edit text items in a dialog?”

See also FAQ 14-13, “How can I change the size and style of text in my lists?”

See also FAQ 23-39, “How can I replace the standard QuickDraw low-level or ‘bottleneck’ functions?”

See also FAQ 24-16, “How can my application process picture comments when drawing a picture?”
Drawing

This section of the QuickDraw: Drawing chapter answers FAQs about many different aspects of drawing graphics. It discusses inverting and highlighting, the graphics pen, drawing with patterns and pixel patterns, and more.

What system initialization do I have to perform before my program can run?

Here is a standard block of code that you can call at the start of your application to initialize your program:

```c
//***************
void ToolboxInit()
{
    InitGraf( &qd.thePort );
    InitFonts();
    FlushEvents( everyEvent, 0 );
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( NIL_POINTER );
    InitCursor();
}
```

`InitGraf()` initializes QuickDraw. This should be the first Toolbox Manager initialized. Among other things, the QuickDraw global variables are initialized.

`InitFonts()` initializes the Font Manager. It loads the System font into memory if it isn't already. `InitFonts()` must be called after `InitGraf()` and before any Toolbox routine that uses the Font Manager.

`FlushEvents()` is called at the beginning of the program to empty the event queue of any keystrokes or mouse events that were generated by the user while your application was being loaded.

`InitWindows()` initializes the Window Manager.

`InitMenus()` creates the application’s current menu list. It also draws an empty menu bar.
TEInit() initializes TextEdit. Even if you do not explicitly call any TextEdit functions, you must call TEInit() before you initialize the Dialog Manager.

InitDialogs() initializes the Dialog Manager.

InitCursor() sets the current cursor to the standard arrow cursor and makes it visible.

Why can't I access QuickDraw globals like 'thePort' anymore?

The QuickDraw global variables are now elements of a global structure variable called qd. The definition of the structure and the declaration of qd as found in the universal headers are listed below:

```c
struct QDGlobals {
    char privates[76];
    long randSeed;
    BitMap screenBits;
    Cursor arrow;
    Pattern dkGray;
    Pattern ltGray;
    Pattern gray;
    Pattern black;
    Pattern white;
    GrafPtr thePort;
};

typedef struct QDGlobals QDGlobals, *QDGlobalsPtr,
                        **QDGlobalsHdl;

extern QDGlobals qd;

thePort, screenBits, randSeed, and the five standard patterns, white, ltGray, gray, dkGray, and black, are now fields of this structure. Previously you could make the call:

    InitGraf( &thePort );

Now you must call it like this:

    InitGraf( &qd.thePort );
What is the difference between highlighting and inverting?

Inverting reverses the color of all pixels within the shape or region. White pixels are set to black and black to white. The values of color pixels are reversed. On indexed devices this can lead to unpredictable results. Inverting is done with the Invert functions: InvertAre(), InvertOval(), InvertPoly(), InvertRect(), InvertRgn(), and invertRoundRect().

Highlighting only affects the background pixels in the specified shape or region. It sets them to the current Hilite Color. If a rectangle with the coordinates {100, 100, 200, 200} is filled with the black (solid) pattern and the rectangle defined by {50, 50, 250, 250} is highlighted, only a 50 pixel border outside the solid rectangle will be affected, as shown in Figure 23-1.

Highlighting is done by setting a highlighting mode flag and then calling the Invert functions. You can also get the current RGB value of the highlight color and draw with it using RGBForeColor() and RGBBackColor().

Figure 23-1: Inverted and highlighted text

Related FAQs

- See also FAQ 2-9, “Why does inverting a color often produce unexpected results on a 256-color system?”
- See also FAQ 2-11, “How do I get and set the current highlight color?”
- See also FAQ 2-12, “How can I get drawing commands to draw in the highlight color?”
- See also FAQ 23-7, “How do I highlight an object?”
How do I highlight an object?

You can highlight an area in a GrafPort or CGrafPort by clearing the highlight bit and then calling the appropriate invert function. The highlight mode is stored in the global variable HiliteMode and can be retrieved by the call LMGetHiliteMode() defined in the universal headers. The highlight bit is defined by pHiliteBit (equal to 0) and is used to clear the bit flag in HiliteMode. The new mode is then set as the highlight mode by the call LMSetHiliteMode().

```c
void HiliteRects()
{

Rect iRect1 = {100, 100, 200, 200};
Rect iRect2 = {100, 300, 200, 400};
UInt8 hMode;

hMode = LMGetHiliteMode();
BitClr(hMode, pHiliteBit);
LMSetHiliteMode(hMode);

// draw rectangles
```
InvertRect(&iRect1);
InvertRect(&iRect2);

// draw the highlighted border around each
InsetRect(&iRect1, -20, -20);
LMSetHiliteMode(hMode);
InvertRect(&iRect1);

InsetRect(&iRect2, -20, -20);
LMSetHiliteMode(hMode);
InvertRect(&iRect2);

while (!Button());

// remove the highlighted border around each
LMSetHiliteMode(hMode);
InvertRect(&iRect1);

LMSetHiliteMode(hMode);
InvertRect(&iRect2);

Related FAQs

See also FAQ 2-9, “Why does inverting a color often produce unexpected results on a 256-color system?”

See also FAQ 2-11, “How do I get and set the current highlight color?”

See also FAQ 2-12, “How can I get drawing commands to draw in the highlight color?”

See also FAQ 23-6, “What is the difference between highlighting and inverting?”

How do I draw lines and shape outlines of different widths?

The call PenSize(width, height) adjusts width and height of lines drawn in the current graphics port (see Figure 23-2). The width and the height are adjusted separately. This affects the line drawing and shape framing routines.
How do I draw dashed lines?

Dashed lines are created by drawing with a pattern. The white space in a pattern will give the illusion of a dashed line. Be careful when selecting a pattern. Make sure that no path through the pattern will result in all white space. The best means to assure this is by using a checkered pattern, since a traversal through the pattern from any point in any direction will hit part of the pattern (see Figure 23-3).

The example below uses the standard pattern number 26, diagonal lines, to draw a rectangle of dashed lines. Note this pattern will not work with diagonal lines running parallel to the lines in the pattern.

```c
//**************
void DrawDashedRect()
{
    Pattern thePat;
    Rect dottedRect = {100, 100, 200, 250};

    GetIndPattern( &thePat, sysPatListID, 26);
    PenPat( &thePat );

    FrameRect( &dottedRect);
}
```
Related FAQs

- See also FAQ 18-14, “How can I print good looking dashed lines?”
- See also FAQ 23-34, “How can I draw ‘marching ants’ marque?”

**How do I draw a shape filled with a solid color?**

There are three things that you need to manage here: the shape that will be drawn, the color in which it will be drawn, and the current pen pattern that will be used when drawing.

The are three types of routines to draw shapes. Frame functions draw an outline of your object; this is not what we want here. Paint functions draw the interior of an object with the pen’s current pattern and pattern mode. Fill functions fill an object with the specified pattern in the patCopy pattern mode. Note that to draw in a pattern mode other than patCopy you must call the fill routines. You can specify arcs, ovals, rectangles, round rectangles, and regions as the shapes by calling the appropriate functions (for example FrameRect(), FillOval(), and PaintRoundRect()). To draw in a color you must set the foreground color of the graphics port with RGBColor() or ForeColor() to that color. To draw in a solid color the current pattern must set to qd.black.

Alternatively, you can call the pixel pattern filling functions (such as FillCRect()) which fill a shape with a specified pixel pattern. Just pass in a solid color pixel pattern.

**Related Topic**

- See also Chapter 2, “Color.”

**How do I set the pen pattern with universal headers?**

The global patterns dkGray, ltGray, gray, black, and white are now structures in the QuickDraw global variable, qd. Their values in the structures have changed from PatPtr to Patterns. The declaration of PenPat has changed from PenPat(PatPtr) to PenPat(const Pattern *pat).
The old call:

```c
PenPat( gray );
```

The new call:

```c
PenPat((const Pattern *)&qd.gray);
```

**How can I draw in a pattern?**

A bit pattern is a 64-bit image that is organized as an 8 bit by 8 bit square. It is used to define a repeating design or tone by being drawn in a continuous and repeating fashion. To draw in a defined pattern, you must first call `PenPat()` to set the graphics pen of the current graphics port to use the pattern. You can also call the Fill routines, such as `FillRect()`, which allow you to specify a pattern to fill the specified shape.

QuickDraw provides five predefined patterns that are stored as fields in the global variable `qd`. They are:

- white
- black
- gray
- ltGray
- dkGray

These are not colors. Do not try to call `RGBForeColor(ltGray)` or `ForeColor(black)`. Note that black is the pattern to use when you want to draw a solid color. The command below fills a rectangle with a ltGray pattern.

```c
FillRect( &myRect, qd.ltGray );
```

The lines below do the same thing:

```c
PenPat( qd.ltGray );
PaintRect( &myRect );
```

Patterns can be stored in 'PAT' resources. To retrieve a handle to a pattern call `GetPattern()` passing it the resource ID of the desired pattern. This must be dereferenced to the bit pattern before it can be passed to `PenPat()`.
The system maintains a set of 38 standard patterns stored in the System file in a pattern list, `PAT#`, resource with ID 0. The call `GetIndPattern()` is used to extract a particular pattern in pattern list. Unlike `GetPattern()`, `GetIndPattern()` passes back a pattern, not a handle to a pattern.

It is easy to create a pattern in your code. Since a pattern is a series of 64 bits, it can be specified as a series of eight unsigned chars. The pattern below alternates set pixels throughout the pattern.

```c
unsigned char myPattern[] =
{ 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55};
```

The pattern appears like this, with the 1s setting a pixel, and the 0s clearing a pixel:

```
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
```

Related FAQs

- See also FAQ 23-13, “How do I get the standard patterns?”
- See also FAQ 23-14, “How can I draw in a color pattern?”

**How do I get the standard patterns?**

There are five predefined patterns that are stored as fields in the QuickDraw global variable `qd`. They are `qd.dkGray`, `qd.ltGray`, `qd.gray`, `qd.black`, and `qd.white`.

There are 38 more patterns, referred to as the standard patterns, that are stored in the System file in a single `PAT#` resource with an ID of 0 (defined by the constant `sysPatListID`). To access one of these standard patterns, you use the `GetIndPattern()`, passing to it `sysPatListID` as the pattern list.

The lines below get the standard pattern of diagonal lines (see Figure 23-4), which is pattern number 28.
CHAPTER 23: QuickDraw: Drawing

Pattern thePattern;
GetIndPattern(&thePattern, sysPatListID, 28);

Figure 23-4: The 38 standard patterns

Related FAQs
- See also FAQ 23-12, "How can I draw in a pattern?"
- See also FAQ 23-14, "How can I draw in a color pattern?"

**How can I draw in a color pattern?**

There are two ways to draw in color using a pattern. The first is to set the foreground color with `RGBForeColor()`, set the pen to draw using a black-and-white pattern (a 'PAT' resource) with `PenPat()`, and then draw. This will only allow you to draw in a pattern in a single color. Note that `PenPat()` takes a pattern as a parameter so you must doubly dereference it.

```c
PatHandle hPattern;
RGBColor redColor = {OxFFFF, 0x0000, 0x0000};
Rect r = {100, 100, 200, 300};
```
hPattern = GetPattern(128);
PenPat( **hPattern);
RGBForeColor(redColor);
PaintRect( &r );

If you want multi-colored patterns, you must use pixel patterns (a ‘ppat’ resource). Note
that PenPixPat() takes a handle to a pixel pattern as a parameter.

PixPatHandle hPixPattern;

hPixPattern = GetPixPat(128);
PenPixPat( hPixPattern );
PaintRect( &r );

There are also routines that allow you to fill shapes with specified patterns. These
are the fill functions (FillRect(), Fill0val(), and so forth). If you want to
fill a shape with a pixel pattern, you must use the functions FillCRect(),
FillC0val(), etc.

Related FAQs

☐ See also FAQ 23-12, “How can I draw in a pattern?”

☐ See also FAQ 23-13, “How do I get the standard patterns?”

How can I get and set the RGB value of a single pixel?

GetCPixel(short x, short y, RGBColor *rgb) gets the RGB value of
the pixel at coordinate (h,v) in the current CGrafPort. SetCPixel(short x,
short y, RGBColor *rgb) sets the RGB value of a pixel.

You can also set the RGB value of a pixel by setting the pen size to (1,0) and calling
Line(0,1). This is actually faster than calling SetCPixel().
Why does my drawing trash the screen whenever I draw in response to an update event for a window that is not the frontmost window?

You need to set the port of your window to be the active port with the `SetPort()` command. When you receive an update event, get the current port with the call `GetPort()`. Then set the active port to the window to be updated. After you do the appropriate drawing to update the window, restore the active port to the original. The code below illustrates this.

```c
void DoUpdate( EventRecord *eventPtr )
{
    WindowPtr   pWindow;
    GrafPtr     oldPort;

    GetPort( &oldPort );  // get current port

    pWindow = (WindowPtr)eventPtr->message;
    SetPort( pWindow );

    BeginUpdate( pWindow );

    // Draw window contents
    EndUpdate( pWindow );

    SetPort( oldPort );
}
```

Related Topics

- See also Chapter 7, “Events.”
- See also Chapter 33, “Windows.”
Bitmaps and Pixmaps

This section of the QuickDraw: Drawing chapter answers FAQs about manipulating bitmaps and pixmaps. It shows how to flip and rotate the contents of bitmaps and pixmaps, and discusses how to obtain various pieces of information about them. The chapter also explains one of the most confusing values of a structure, the rowBytes field of a pixmap.

How do I flip a pixmap (put up its mirror image)?

There is no Toolbox routine that flips a pixmap; you have to manipulate the bits manually. To flip a pixmap horizontally you simply switch the pixels on the left side of the pixmap with the corresponding pixels on the right side. To flip a pixmap vertically you switch the pixels on the top of the pixmap with the corresponding pixels on the bottom. For the best effect, you should perform the bit manipulation offscreen (see Figure 23-5).

The sample function below, DoPictFlip(), uses a GWorld to flip a picture resource horizontally or vertically and display it on a specified window. The actual bit manipulation is performed by the two routines FlipHorizontal() and FlipVertical(). These two routines transfer data a character at a time. This limits the programmer to working with 8 bit color. To work with other color depths, the sample function can be modified to manipulate bits rather than bytes.
/*******************************/
void DoPictFlip( WindowPtr pictWindow,
                 PicHandle flipPict,
                 short dir)
{
    PixMapHandle map = nil;
    unsigned long rowBytes;
    Ptr mapPtr;
    GrafPtr oldPort = nil;
    Rect windRect;
    short wide, high;
    char mode;
    Boolean swapped = false;

    // create a GWorld
    GetGWorld(&gSavedWorld, &gSavedDevice);
    windRect = (**flipPict).picFrame;
    gGWorld = CreateGWorld(windRect);
    SetGWorld(gGWorld, nil);
    map = GetGWorldPixMap(gGWorld);

    // get current addressing mode
    mode = GetMMUMode();

    if(mode == false32b)
    {
        swapped = true;
        SwapMMUMode(&mode);
    }

    NoPurgePixels(map);
    if(LockPixels(map))
    {
        HLock((Handle)flipPict);
        DrawPicture(flipPict, &gGWorld->portRect);

        wide = (*map)->bounds.right - (*map)->bounds.left;
        high = (*map)->bounds.bottom - (*map)->bounds.top;
        rowBytes = (*map)->rowBytes & Ox3FFF;

        mapPtr = GetPixBaseAddr(map);
        if(mapPtr)
        {

```
switch(dir)
{
    case 0:
        FlipHorizontal(mapPtr, wide, high, rowBytes);
        break;
    case 1:
        FlipVertical(mapPtr, wide, high, rowBytes);
        break;
}

ForeColor(blackColor);
BackColor(whiteColor);
CopyBits((&(GrafPtr)gGWorld)->portBits,
    &pictWindow->portBits, 
    &gGWorld->portRect,
    &pictWindow->portRect, 
    srcCopy, nil);

HUnlock((Handle)flipPict);
UnlockPixels(map);
}
AllowPurgePixels(map);

if(swapped == true)
    SwapMMUMode(&mode);

SetGWorld(gSavedWorld, gSavedDevice);
DisposeGWorld(gGWorld);

 completionHandler
***************
GWorldPtr CreateGWorld(Rect theRect)
{
    OSErr iErr = noErr;
    GWorldPtr theGWorld = nil;
    GWorldPtr currWorld = nil;
    GDHandle currDevice = nil;
    PixMapHandle map = nil;

    // Initialize the GWorld
    theGWorld = CreateGWorld(theRect);
    currWorld = CreateGWorld(currRect);
    currDevice = CreateGDevice(currRect);
    map = CreatePixmap(mapRect);

    // Configure the GWorld
    ConfigureGWorld(currWorld, currDevice, map);
    ConfigureGDevice(currDevice, map);
    ConfigurePixmap(map, theRect);

    // Draw to the GWorld
    DrawGWorld(currWorld, theGWorld, map);
    DisposeGDevice(currDevice);
    DisposePixmap(map);
    DisposeGWorld(currWorld);

    return theGWorld;
}
GetGWorld(&currWorld, &currDevice);
iErr = NewGWorld(&theGWorld, 8, &theRect, nil, nil, 0);
if (iErr == noErr)
{
    map = GetGWorldPixMap(theGWorld);
    if (LockPixels(map))
    {
        SetGWorld(theGWorld, nil);
        EraseRect(&theGWorld->portRect);
        UnlockPixels(map);
    }
}
SetGWorld(currWorld, currDevice);
return(theGWorld);

//********************
void FlipHorizontal (  
    register Ptr pBits,  
    register short width,  
    register short height,  
    register unsigned long rowBytes )
{
    register short i, j, k, temp;
    for (i = 0; i < height; i++)
    {
        for (j = 0, k = width - 1; j < k; j++, k--)
        {
            temp = *(pBits + j);
            *(pBits + j) = *(pBits + k);
            *(pBits + k) = temp;
        }
        pBits += rowBytes;
    }
}

//********************
void FlipVertical (  

register Ptr pBits,
register short width,
register short height,
register unsigned long rowBytes)
{
  register short i, j, k, temp;

  for (i = 0; i < width; i++)
  {
    for (j = 0, k = height - 1; j < k; j++, k--)
    {
      temp = *(pBits + j * rowBytes);
      *(pBits + j * rowBytes) = *(pBits + k * rowBytes);
      *(pBits + k * rowBytes) = temp;
    }
    pBits += 1;
  }
}

Related FAQs

- See also FAQ 23-19, “How do I rotate a pixmap?”
- See also FAQ 24-17, “How do I rotate a picture?”
- See also FAQ 31-18, “How can I draw rotated text?”

Related Topic

- See also Chapter 24, “QuickDraw: Pictures.”

**What can I do to speed up CopyBits()?**

`CopyBits()` is used to transfer an image from one pixmap to another, most often from an offscreen pixmap to a window’s graphics port. It can be executed repeatedly with different images to produce the affect of animation. One of the primary benefits of performing these operations with `CopyBits()` is the speed with which the image can be transferred. To maximize the speed, producing the smoothest animation and cleanest effects, there are a few conditions that you must make sure are met:
Make sure that the source rectangle is exactly the same as the destination rectangle. If they are even one pixel off, CopyBits() will have to scale the image.

Make sure that the intersection of the current graphics port's clipRgn and visRgn and the maskRgn parameter passed into CopyBits() is rectangular. If it is not, then CopyBits() has to check each pixel to make sure that it falls within the intersection. This significantly slows down the copying operation. You may want to set the visRgn and clipRgn regions of the destination window to its entire rectangle, just to make sure.

Source and destination pixel maps times the bit depth are the same. The necessary alignment is 32 bits (4 bytes). Both pixmaps should have the same bit offset from 32 bit alignment. If a pixmap is offscreen, its alignment can be calculated with a global coordinate offset of (0,0).

Make sure that the source pixmap and the destination pixmap have the same pixel depth. If they do not, then CopyBits() will automatically do color mapping to match the colors. However, setting the pixel depths of the source and destination pixmaps the same does not necessarily avoid the color mapping. The pixmaps must have the same color tables. If they do not, CopyBits() will inspect each pixel to find the closest match in the target pixel map instead of simply doing a quick data copy. Fine, we will just make sure that the color tables contain the same colors. Not so fast. The first action CopyBits() performs to compare color tables is to check their ctSeed field. If they are the same, CopyBits() assumes that the color tables are identical and does not perform any color translation.

If you assign a color table to a pixmap or a GWorld, make sure that it is complete for the pixel depth.

**How do I rotate a pixmap?**

There is no function that will rotate a pixmap. You must manually manipulate the bits. This is best accomplished by creating an offscreen GWorld, manipulating the pixmap's bits onto the GWorld, and then using CopyBits() to transfer the image to the destination graphics port.
How do I get the most frequently used colors in a pixmap?

You can get the most frequently used colors in a pixmap or get a weighted distribution of colors used in a pixmap with the `GetPixmapInfo()` command.

Related FAQ

- See FAQ 24-21, “How do I get a palette or color table of a picture?”

Related Topic

- See also Chapter 2, “Color.”

Why is the rowBytes field of my pixmap a negative number?

The first bit is used to distinguish between a pixmap and a bitmap. The second bit is used in conjunction with the first bit to distinguish between parameter types in calls to `CopyBits()`, `CopyMask()`, `CopyDeepMask()`, `SeedCFill()`, and `CalcCMask()`.

Related FAQs

- See also FAQ 23-23, “How can I tell if my structure is a bitmap or a pixmap?”
- See also FAQ 23-24, “Can I pass a pixmap of a color window to `CopyBits()`, which takes a bitmap as a parameter?”
Why do the colors in my image copied with CopyBits() get distorted?

CopyBits() applies the current graphics port's foreground and background colors to the image of the destination pixel map. This causes the foreground color to replace all black pixels and the background color to replace all white pixels in the destination. To prevent this, use RGBForeColor() to set the foreground color to black and RGBBackColor() to set the background color to white.

Related Topic

☑ See also Chapter 2, “Color.”

How can I tell if my structure is a bitmap or a pixmap?

You can determine if a structure is a bitmap or a pixmap by examining the rowBytes field. The most significant bit of the rowBytes field is set in a pixmap. It is not set in a bitmap.

Related FAQs

☑ See also FAQ 23-21, “Why is the rowBytes field of my pixmap a negative number?”

☑ See also FAQ 23-24, “Can I pass a pixmap of a color window to CopyBits(), which takes a bitmap as a parameter?”

☑ See also FAQ 23-27, “Why is the rowBytes field of my GWorld’s pixmap larger than what I defined? How much memory should I allocate for my GWorld buffer?”
Can I pass a pixmap of a color window to `CopyBits()`, which takes a bitmap as a parameter?

You can pass a pixmap of a color window in the first two parameters which are designated as type `Bitmap`. To do this, you type cast the `portPixMap` field to a bitmap and pass its address. The example below casts the `CGrafPtr` of a color window to a `GrafPtr` and then passes the address of `portBits`.

```c
CGrafPtr      myCPort;
GrafPtr       savedPort;

CopyBits( &((GrafPtr)myCPort)->portBits, &savedPort->portBits, 
          &myCPort->portRect, &savedPort->portRect, srcCopy, nil );
```

This seems confusing since `portPixMap` is a handle to a pixmap and `CopyBits()` expects a pointer to a bitmap. `CopyBits()` resolves this problem by examining the first two bits that are offset from the location pointed to by the address passed to it. If a pointer to a bitmap or pixmap was passed in, then the first two bits of the `rowBytes` field of the structure are examined. For a bitmap they will be set to 00. For a pixmap they will be set to 10. If a pointer to a color graphics port's `portPixMap` field is passed in, then the first two bits of the color graphics port's `portVersion` field are examined. These will be set to 11. Once `CopyBits()` has determined what type of data it has been passed, it can act accordingly.

Related FAQs

- See also FAQ 23-21, “Why is the `rowBytes` field of my pixmap a negative number?”
- See also FAQ 23-23, “How can I tell if my structure is a bitmap or a pixmap?”
- See also FAQ 23-27, “Why is the `rowBytes` field of my GWorld's pixmap larger than what I defined? How much memory should I allocate for my GWorld buffer?”

Can `CopyBits()` copy overlapping rectangles in the same window?

Yes, you can copy an image from one rectangle to an overlapping source rectangle in the same window. `CopyBits()` handles this properly.
GWOrlds and Offscreen Drawing

This section of the QuickDraw: Drawing chapter answers FAQs about using GWOrlds and offscreen drawing. It provides examples of using GWOrlds, shows how to create a picture from the contents of a GWOrld, and more.

How do I draw in my window using a GWOrld?

Using an offscreen graphics world to update a window requires three basic steps: creating the GWOrld, drawing the graphics in the GWOrld, and copying the graphics from the GWOrld to the window.

To create a GWOrld you use NewGWOrld() and pass the portRect of your destination window in global coordinates and a pointer to the window’s graphics device. The window’s graphics device can be retrieved with GetGWOrld().

To draw in the offscreen graphics world you must make it the current port by calling SetGWOrld(). Before you do any drawing you must first prevent the GWOrld from moving in memory during the drawing operations by calling LockPixels(). LockPixels() takes a pointer the pixel map of an offscreen graphics world. Do not get this value directly from the pixel map’s structure; use GetGWOrldPixMap().

To transfer the image from the offscreen graphics port to your window, use CopyBits(). Set the foreground color to black and the background color to white before calling CopyBits() to avoid unwanted colorization. Again call LockPixels() before transferring the image.

The code below uses an offscreen GWOrld to draw on and then transfer the contents to the current window.

//***********************
// macros to get upper left and lower right points of Rects
#define mTopleft(r) (((Point *)&(r))[0])
#define mBotRight(r) (((Point *)&(r))[1])

void UseAGWorld( )
{
    CGrafPtr curPort;
    GDHandle curDevice;
}
PixMapHandle offPix;
Rect myWindowRect;
Rect r = {100, 100, 200, 200};
GWorldPtr myGWorld;
RGBColor blackRGB = {0, 0, 0};
RGBColor whiteRGB = {65535, 65535, 65535};
OsErr gErr
GetGWorld(&curPort, &curDevice);

/********************************************
// create GWorld
// get rectangle defining current port's interior,
// translate it into global coordinates.
********************************************
myWindowRect = curPort->portRect;
LocalToGlobal(&mTopLeft(myWindowRect));
LocalToGlobal(&mBotRight(myWindowRect));

myGWorld = nil;
gErr = NewGWorld(&myGWorld, 0, &myWindowRect, nil,
curDevice, nil);
if ( !myGWorld )
    return;

/********************************************
// draw to offscreen
********************************************
offPix = GetGWorldPixMap(myGWorld);
if(LockPixels(offPix))
{
    SetGWorld( myGWorld, nil );
    EraseRect( &(myGWorld->portRect) );

    // do your drawing here
    InvertRect(&r);

    UnlockPixels(offPix);
}
SetGWorld(curPort, curDevice);

/********************************************
// blit offscreen image to window
/********************************************
Why is the rowBytes field of my GWorld's pixmap larger than what I defined? How much memory should I allocate for my GWorld buffer?

The bitmap and pixmap structures both contain a rowBytes field, which represents the row width, and a bounds field, which represents the bounding rectangle. If there is a bounding rectangle defined, why does there also have to be a rowBytes field? If you examine the rowBytes field and compare it with the width of the bounding rectangle, you will notice that these numbers are not always in sync. This is because there are unused bits at the end of each row (we'll address the reasons for this later). The bounding rectangle is used to indicate the right edge of the actual image. The rowBytes field actually defines the offset in memory from one row to the next.

The value of the rowBytes field depends upon the pixel size and the number of color bits per pixel. QuickDraw will pad the rowBytes so the address can be long-word aligned for performance reasons. The formula for calculating the rowBytes value for a bitmap is:

\[
\text{bytesPerRow} = (((\text{bounds}->\text{right} - \text{bounds}->\text{left}) - 1) / 32) + 1 ) * 4;
\]

where bounds is the bounding rectangle of the pixmap. The formula for calculating the rowBytes value for a bitmap using bit shifting is:
bytesPerRow = ((depth * (bounds->right - bounds->left) + 31) >> 5) << 2;

where depth is the number of bits per pixel and bounds is the bounding rectangle of the pixmap.

It is also important to remember that the first two bits of the rowBytes field are reserved for use by QuickDraw. The first bit is used to distinguish between a pixmap and a bitmap. The second bit is used in conjunction with the first bit to distinguish between parameter types in calls to CopyBits(), CopyMask(), CopyDeepMask(), SeedCFill(), and CalcCMask(). To get the bytes per row value from the rowBytes field you must perform a bitwise AND on the rowBytes value and 0x3FFF. With these two high bits reserved, the maximum value of rowBytes is 0x3FFE.

Related FAQs

- See also FAQ 23-21, “Why is the rowBytes field of my pixmap a negative number?”
- See also FAQ 23-23, “How can I tell if my structure is a bitmap or a pixmap?”
- See also FAQ 23-24, “Can I pass a pixmap of a color window to CopyBits(), which takes a bitmap as a parameter?”

How do I get the address of the pixels of a GWorld’s pixmap?

The address of the GWorld’s offscreen buffer is not directly accessible from its pixmap. Use GetPixBaseAddr(). If the offscreen buffer has been purged, GetPixBaseAddr() will return nil.

How do I put contents of a GWorld into a picture?

To put the contents of a GWorld into a picture, first set the GWorld as the current graphics world. Next, open the picture, and use CopyBits() to copy the contents of the GWorld to itself. Finally, close the picture.
The sample below creates a picture from the contents of a GWorld.

/***************/

PicHandle MakeGWorldPict(GWorldPtr theGWorld)
{
    PicHandle    hPict;
    PixMapHandle hPixMap;
    CGrafPtr    port;
    GDHandle    hGDev;

    GetGWorld(&port, &hGDev);
    SetGWorld(theGWorld, NULL);

    hPixMap = GetGWorldPixMap(theGWorld);

    LockPixels(hPixMap);

    hPict = OpenCPicture(&header);

    CopyBits((BitMapPtr) *hPixMap, (BitMapPtr) *hPixMap,
             &theGWorld->portRect, &theGWorld->portRect,
             srcCopy, NULL);

    ClosePicture();

    UnlockPixels(hPixMap);
    SetGWorld(port, hGDev);
    return(hPict);
}

Related Topic

See also Chapter 24, "QuickDraw: Pictures."
Why do my colors get distorted when I use CopyBits() from my GWorld?

Make sure that your foreground color is set to black and your background color is set to white. Depending on the transfer mode used, CopyBits() will integrate the foreground and background colors into the operation to produce different coloration effects. For example, if the srcCopy mode is used, Color QuickDraw determines how close a pixel's color is to black and then assigns the same proportion of the foreground color to the destination pixel. It also determines how close a pixel's color is to white and then assigns the same proportion of the background color to the destination pixel.

If you have changed or replaced the color table of the GWorld then you need to make sure that the inverse table gets rebuilt. Color QuickDraw will automatically rebuild the inverse table for you if it detects that the color table has changed. You can signal that the color table has changed by modifying the ctSeed field of the color table. This is done by calling CTabChanged().

Why do I get garbage when I use CopyBits() from an offscreen bitmap or a GWorld?

Assuming that you are copying from a valid bitmap or pixmap, a ‘dirty image’ may result if you did not clear the memory allocated to create the bitmap or pixmap. If you are allocating memory for one of these structures use NewPtrClear(). If you are using a GWorld, call EraseRect() to clear the area that you will be drawing to before you do any drawing.

How can I use my own colors in a GWorld?

To assign your custom colors to a GWorld you can call NewWorld() with a color table that contains your customized colors. If you want to assign a new color table to an existing GWorld then you can call UpdateGWorld(). If you want to assign the colors of a palette to a GWorld you can call Palette2CTab() to copy the colors of a palette into a color table and then assign the resulting color table to the GWorld. Then set the assign the palette to your destination window.

To get maximum speed from the CopyBits() function, you will want to avoid requiring your application to use the GWorld's color matching algorithm to synchronize the colors from the offscreen world with colors of the palette. You can
set up a color table so that it refers to palette indices instead of RGB values, thus automatically synchronizing the color table with the window's palette. To do this, set bit 14 in the ctFlags field of the GWorld's color table. Normally the color table maps a pixel value to a color. When bit 14 is set, synchronizing the color table and a palette, the color table maps an index value to a color. Bit 14 of a color table is set by the following command:

(*colorTableHandle)->ctFlags |= 0x4000

Synchronizing the color table and with a palette enables you to use the calls PMForeColor() and PMBackColor() when drawing in offscreen ports. This also makes it possible to use animated indices in an offscreen world.

In sum, the proper use of a GWorld with customized colors requires you to:

- Get a color table that contains your custom colors.
- Set bit 14 of the ctFlags field of the color table.
- Create a palette from the color table with NewPalette().
- Assign the palette to a window with SetPalette().
- Create a GWorld with the color table.

Now you can draw to GWorld and use CopyBits() to transfer the image to your window.

Related Topic

See also Chapter 2, "Color."

Marquees and Selection Lines

This section of the QuickDraw: Drawing chapter answers FAQs about displaying rubber banding lines and the selection marquee. Rubber banding lines are used in an application to present to the user instant feedback on a line or the outline of a shape being adjusted by dragging the mouse. The selection marquee displays a rectangular outline made of dashed lines that appear to be moving. Code samples are presented to display these items and to demonstrate how to display a color selection marquee.
How do I draw 'rubber banding' lines (anchored lines and selection rectangles)?

The key to drawing anchored lines and selection rectangles is the pen pattern transfer mode, patXor. The patXor mode inverts the screen where the drawing pattern and the screen are both black. The second consecutive identical drawing call with the pen mode set to patXor undoes whatever drawing the first call performed. Setting the pen pattern to gray produces a light, slightly broken line for a better visual effect.

The example below draws a selection rectangle. This routine is called when a mouse-down occurs; the cursor position of the mouse-down event would be passed to it. The function sets the pen mode to patXor and the pattern to gray. It then sets the point of the mouse-down event as the anchor point and draws a rectangle using the anchor point and the new point to define the rectangle. While the user has the mouse button pressed, tested by StillDown(), each time the mouse is moved the old rectangle is redrawn to erase it and a new rectangle is drawn.

```c
//******************
void DoSelectRect( Point mLoc )
{
    Point BeginPt, Last, Next;
    Rect PreviousRect;

    GlobalToLocal(&mLoc);

    SetPt(&BeginPt, mLoc.h, mLoc.v);
    PenNormal();
    PenPat((const Pattern *)&qd.gray);
    PenMode(patXor);

    GetMouse(&Last);

    Pt2Rect(BeginPt, Last, &PreviousRect);
    FrameRect(&PreviousRect);

    while( StillDown() )
    {
        GetMouse(&Next);
        if ( ( Next.h != Last.h ) || ( Next.v != Last.v ) )
        {
            Pt2Rect(BeginPt, Last, &PreviousRect);
            FrameRect(&PreviousRect);
        }
    }
}
```
The example below draws an anchored line. It works the same as `DoSelectRect()` above, except that it draws a line instead of a rectangle.

```c
//******************
static DoAnchoredLine( Point mLoc )
{
    Point BeginPt,
        Last,
        Next;

    GlobalToLocal(&mLoc);

    SetPt(&BeginPt,mLoc.h, mLoc.v);
    PenNormal();
    PenPat((const Pattern *)&qd.gray);
    PenMode(patXor);

    GetMouse(&Last);

    MoveTo( BeginPt.h, BeginPt.v);
    LineTo( Last.h, Last.v );

    while( StillDown() )
    {
        GetMouse(&Next);
        if ( ( Next.h != Last.h ) || ( Next.v != Last.v ) )
        {
            MoveTo( BeginPt.h, BeginPt.v);
            LineTo( Last.h, Last.v );
            MoveTo( BeginPt.h, BeginPt.v);
            LineTo( Next.h, Next.v );
            Last = Next;
        } else
```
MoveTo( BeginPt.h, BeginPt.v );
LineTo( Last.h, Last.v );
PenNormal();

Related FAQs

See also FAQ 23-34, "How can I draw 'marching ants' marque?"
See also FAQ 23-35, "How can I make a selection rectangle in color?"

How can I draw 'marching ants' marque?

The "marching ants" marque is a dashed line or rectangle where the dashes appear to be moving. Drawing this is similar to drawing the rubber banding lines and selection rectangles. To create the effect of animated dashes, the lines are drawn in the patXor mode with the current pattern continuously cycling through a series of patterns each with the diagonal line at a different offset. The lines are drawn each time a new pattern is set as the current pattern.

The pattern that displays a diagonal line can be defined by the eight bytes:

```
{ 0xF8, 0xF1, 0xE3, 0xC7, 0x8F, 0x1F, 0x3E, 0x7C }
```

By repeating this string of bytes so that we have an array of 15 bytes, we can use it to define eight different bitmaps. The first bitmap consists of the eight bytes starting at patterns[0]; the next bitmap consists of the eight bytes starting at patterns[1]; etc.

```
unsigned char patterns[] =
{
  0xF8, 0xF1, 0xE3, 0xC7, 0x8F,
  0x1F, 0x3E, 0x7C, 0xF8, 0x1F,
  0xE3, 0xC7, 0x8F, 0x1F, 0x3E
};
```

The sample below draws a rectangle with the marching ants animated lines. Every two clock ticks the line is redrawn with a new pattern. The rectangle stays displayed until the mouse button is pressed.
/************************************************
#define MARCHINGANTDELAY 2

void ShowMarchingAntsRect()
{
    unsigned char    patterns[] =
    {
        0xF8, 0xF1, 0xE3, 0xC7, 0x8F,
        0x1F, 0x3E, 0x7C, 0xF8, 0xF1,
        0xE3, 0xC7, 0x8F, 0x1F, 0x3E
    };
    PenState penState;
    Rect    nowRect    = { 10, 10, 200, 200};
    long    tickValue,
    oldTickValue;
    short   patternIndex;

    oldTickValue = 0;
    patternIndex = 0;

    GetPenState(&penState);
    PenMode(patXor);

    PenPat((const Pattern *)&patterns[patternIndex]);
    FrameRect(&nowRect);

    while( !Button() )
    {
        tickValue = TickCount();
        if( (oldTickValue + MARCHINGANTDELAY) < tickValue )
        {
            oldTickValue = tickValue;
            FrameRect(&nowRect);
            patternIndex = patternIndex < 7 ?
                patternIndex + 1 : 0;
            PenPat((const Pattern *)&patterns[patternIndex]);
            FrameRect(&nowRect);
        }
        FrameRect(&nowRect);
        SetPenState(&penState);
    }
Related FAQs

- See also FAQ 18-14, “How can I print good looking dashed lines?”
- See also FAQ 23-9, “How do I draw dashed lines?”
- See also FAQ 23-33, “How do I draw ‘rubber banding’ lines (anchored lines and selection rectangles)?”
- See also FAQ 23-35, “How can I make a selection rectangle in color?”

How can I make a selection rectangle in color?

Displaying a color selection rectangle is similar to displaying a black-and-white one. The only difference is instead of calling PenPat() to draw in patXor mode with a pattern, you call PenPixPat() to draw with a pen pixel pattern. By loading in a color pixel pattern, drawing in patXor mode will draw in color. Note that the color will change when the lines are drawn over a non-white background. If you want to maintain the color of the selection rectangle over a non-white background, you have to draw the lines normally (in scrOr mode) and use CopyBits() to erase them.

The sample below draws a rectangle and loads eight ‘ppat’ resources into an array. It then sets the pen mode to patXor and repeatedly calls PenPixPat() to load in different pixel patterns. Each time a new pixel pattern is loaded, the rectangle is drawn. The pixel patterns are designed so that rotating through them produces a marching ants effect.

```c
//************************************************
#define MARCHINGANTDELAY 2

void ShowColorMarchingAntsRect()
{
    PenState penState;
    Rect nowRect = { 10,10, 200, 200};
    long tickValue, oldTickValue;
    PixPatHandle ppat[8];
    short patternIndex, i;

    oldTickValue = 0;
    patternIndex = 0;
```
GetPenState(&penState);
PenMode(patXor);

for ( i = 0; i < 8; i++)
    ppat[i] = GetPixPat( 128 + i);

PenPixPat(ppat[patternIndex]);
FrameRect(&nowRect);

while( !Button() )
{
    tickValue = TickCount();
    if ( (oldTickValue + MARCHINGANTDELAY) < tickValue )
    {
        oldTickValue = tickValue;
        FrameRect(&nowRect);
        patternIndex = patternIndex < 7 ?
            patternIndex + 1 : 0;
        PenPixPat(ppat[patternIndex]);
        FrameRect(&nowRect);
    }
}
FrameRect(&nowRect);
SetPenState(&penState);

Related FAQs

- See also FAQ 23-33, “How do I draw ‘rubber banding’ lines (anchored lines and selection rectangles)?”
- See also FAQ 23-34, “How can I draw ‘marching ants’ marque?”

Special Drawing

This section of the QuickDraw: Drawing chapter answers FAQs about doing some wild and crazy things with drawing. It shows how you can intercept the QuickDraw “bottleneck” functions to enhance or completely change a QuickDraw command. It also discusses how to dim out text and other objects, how to draw outside your window, and more.
How do I implement a ZoomRect (a thin gray rectangle that “zooms out” when a window opens or closes)?

You can implement zoom rectangles from one rectangle to another by drawing in the patXor mode. Drawing in this mode allows you to draw a rectangle frame with FrameRect() and then erase the rectangle, returning the graphics port back to its original state with another call to FrameRect(). Simply draw and erase a series of rectangles, increasing or decreasing the rectangle size with each new rectangle. For the zoom window effect employed by the Finder, a small number of rectangles will be visible at the same time.

There are also two routines in the Drag Manager, ZoomRects() and ZoomRegion(), that give you built-in zooming behavior.

The sample below zooms out from a small rectangle to a larger rectangle and then zooms back to the small rectangle.

```c
#define kZoomSteps 15
#define kZoomRects 3
#define kZoomRatio 0.7
#define kZoomDelays 2

//*******************************
void DoZoomRect()
{
    Rect myZoomRect1 = {10, 10, 30, 30};
    Rect myZoomRect2 = {110, 110, 320, 320};

    ZoomRect(true,&myZoomRect1,&myZoomRect2);
    ZoomRect(false,&myZoomRect1,&myZoomRect2);
}

//*******************************
void ZoomRect(Boolean doGrow,
              Rect *smallRect,
              Rect *bigRect)
{
    double firstRatio, leaderRatio, followerRatio;
    long finalTicks;
```
short i;
Rect curRect;

PenPat((const Pattern *)&qd.gray);
PenMode(patXor);

// determine size of ratio that will set src rect
// to dest rect after kZoomSteps
firstRatio = kZoomRatio;
for (i=0; i < kZoomSteps; i++)
    firstRatio *= kZoomRatio;
if (!doGrow)
    firstRatio = 1.0 - firstRatio;

leaderRatio = firstRatio;
followerRatio = firstRatio;

for (i = 0; i < (kZoomSteps + kZoomRects); i++)
{
    // draw new frame
    if (i < kZoomSteps)
    {
        if (doGrow)
            leaderRatio /= kZoomRatio;
        else
            leaderRatio *= kZoomRatio;
        GetNewRect(&curRect, smallRect, bigRect, leaderRatio);
        FrameRect( &curRect );
    }

    // erase old frame
    if (i >= kZoomRects)
    {
        if (doGrow)
            followerRatio /= kZoomRatio;
        else
            followerRatio *= kZoomRatio;
        GetNewRect( &curRect, smallRect, bigRect, followerRatio);
        FrameRect( &curRect );
    }
Related Topic

See also Chapter 33, "Windows."

**How can I make text or an object appear dimmed or grayed-out without redrawing it?**

You can make text and objects appear to be grayed-out in a bitmapped gray by using the `notPatBic` transfer mode. Set the pen mode to `patBic` and Paint over the area that you want to appear dimmed.
CHAPTER 23: QuickDraw: Drawing

```c
PenPat(&qd.gray));
PenMode( patBic );
PaintRect(&destRect);
```

**Related FAQs**

- See also FAQ 2-10, “What RGB value does the system use for dimmed buttons, menus, and window titles?”
- See also FAQ 6-16, “How do I get my controls to dim with light gray instead of bitmapped gray?”
- See also FAQ 6-28, “How do you deactivate (gray-out) static text items and edit text items in a dialog?”
- See also FAQ 31-17, “How do I draw dimmed (grayed-out) text?”

*Is there a way to draw outside a window on the Desktop?*

To draw outside the window on the Desktop, you draw on the Window Manager port. This is obtained by calling `GetWMgrPort()` on systems that only support the old monochrome system and `GetCWMgrPort()` on all systems that support Color QuickDraw. The Window Manager port can be handled in the same way one handles a port to a window. To draw on the window with QuickDraw routines, set the port to the Window Manager port with `SetPort()`. Then set the clipping rectangle to be the port's `portRect` so you can draw anywhere on the screen. Make sure that you get the old clipping region before you set it. When you are done drawing, reset the clipping region to the original value.

When you draw outside the boundaries of your window onto the Desktop (see Figure 23-6), make sure you clean up and reset the Desktop the way you found it.
The sample code below gets a pointer to the Window Manager port and then displays an inverted rectangle traveling diagonally downward on the desktop.

```c
//********************************************************************
// Passes back Window Manager port
//********************************************************************
void GetDeskTopGrafPort( GrafPtr *wMgrPort )
{
  OSErr err;
  long  response;

  err = Gestalt( gestaltQuickdrawVersion, &response);
  if ( (err == noErr) && (response >= gestalt8BitQD) )
     GetCWMgrPort((CGrafPtr *)wMgrPort);
  else
     GetWMgrPort(wMgrPort);
}

//********************************************************************
// Moves an inverted square from the upper left corner of the screen diagonally downwards
//********************************************************************
void RunningInvert()
{
CHAPTER 23: QuickDraw: Drawing

GrafPtr oldPort, wMgrPort;
RgnHandle oldClip;
Rect r = {0,0,100,100};
long finalTick;
int i;

GetPort( &oldPort );

GetDeskTopGrafPort( &wMgrPort);
SetPort(wMgrPort);
oldClip = NewRgn();
GetClip(oldClip);
ClipRectC&wMgrPort->portRect);

InvertRect(&r);
Delay(30, &finalTick);
for ( i = 0; i < 20; i++)
{
    InvertRect(&r);
    OffsetRect(&r, 30, 30 );
    InvertRect(&r);
    Delay(30, &finalTick);
}

SetClip(oldClip);
DisposeRgn(oldClip);
SetPort(oldPort);

Related FAQ

See also FAQ 33-17, “How do I get the GrafPort of the Desktop?”

How can I replace the standard QuickDraw low-level or “bottleneck” functions?

To replace one of the standard QuickDraw low level routines you first get a window’s QDProcs or CQDProcs structure containing pointers to the current routines. You then insert a pointer to your customized function into the proper field of the structure. Finally, you store the modified structure into the grafProcs field.
of the window's graphics port. Whenever a function that uses the low level routine that you replaced is executed, your function will be called.

If a graphics port only uses the standard low level routines, then its grafProcs field is set to nil. If any of the routines are replaced, then the grafProcs field contains a pointer to a CQDProcs or QDProcs structure whose fields contain pointers to the different low level routines, including pointers to both the standard and customized routines. To get the pointers to the standard functions currently used by the graphics port, call SetStdCProcs() or SetStdProcs(). If you are working with a color graphics port, call SetStdCProcs() to get a CQDProcs structure defined below. Otherwise call SetStdProcs() to get a QDProcs structure which is the first thirteen fields of the CQDProcs structure.

```c
struct CQDProcs
{
    QDTextUPP      textProc;
    QDLineUPP      lineProc;
    QDRectUPP      rectProc;
    QDRRectUPP     rRectProc;
    QDOvalUPP      ovalProc;
    QDArcUPP       arcProc;
    QDPolyUPP      polyProc;
    QDRgnUPP       rgnProc;std
    QDBitsUPP      bitsProc;
    QDCommentUPP   commentProc;
    QDTxMeasUPP    txMeasProc;
    QDGetPicUPP    getPicProc;
    QDPutPicUPP    putPicProc;
    // fields added to QDProcs
    QDOpcodeUPP    opcodeProc;
    UniversalProcPtr newProc1;
    UniversalProcPtr newProc2;
    UniversalProcPtr newProc3;
    UniversalProcPtr newProc4;
    UniversalProcPtr newProc5;
    UniversalProcPtr newProc6;
};
```

After you call SetStdCProcs(), copy that CQDProcs structure to maintain pointers to the standard routines. This will enable you to have your customized functions call them to complete the standard actions if necessary. Next you can insert a pointer to your customized function into the appropriate field in the structure passed back by SetStdCProcs(). Use the NewQD***Proc() routines to turn your function pointer into a universal procedure pointer when assigning it...
to the CQDPprocs structure. It is important that your customized function use the same parameters as the standard functions. The function prototypes are listed in Table 23-1.

Table 23-1: QuickDraw “Bottleneck” Function Prototypes

<table>
<thead>
<tr>
<th>Function Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>void StdText(short byteCount, Ptr textBuf, Point numer, Point denom);</td>
</tr>
<tr>
<td>void StdLine(Point newPt);</td>
</tr>
<tr>
<td>void StdRect(GrafVerb verb, Rect *r);</td>
</tr>
<tr>
<td>void StdRRect(GrafVerb verb, Rect *r, short ovalWidth, short ovalHeight);</td>
</tr>
<tr>
<td>void StdOval(GrafVerb verb, Rect *r);</td>
</tr>
<tr>
<td>void StdArc(GrafVerb verb, Rect *r, short startAngle, short arcAngle);</td>
</tr>
<tr>
<td>void StdPoly(GrafVerb verb, PolyHandle poly);</td>
</tr>
<tr>
<td>void StdRgn(GrafVerb verb, RgnHandle rgn);</td>
</tr>
<tr>
<td>void StdBits(BitMap *srcBits, Rect *srcRect, Rect *dstRect, short mode, RgnHandle maskRgn);</td>
</tr>
<tr>
<td>void StdComment(short kind, short dataSize, Handle dataHandle);</td>
</tr>
<tr>
<td>short StdTxtMeas(short byteCount, Ptr textAddr, Point *numer, Point *denom, FontInfo *info);</td>
</tr>
<tr>
<td>void StdGetPic(Ptr dataPtr, short byteCount); void StdPutPic(Ptr dataPtr, short byteCount);</td>
</tr>
</tbody>
</table>

Next store a pointer to the modified structure into the grafProcs field of the graphics port. When you no longer want the QuickDraw to call your customized routines, simply set the grafProcs field to nil.

Along with performing its own customized functionality, your customized function can call the standard routine to have the standard functionality performed. The pointers to the standard routines were passed to you in the CQDPprocs structure from SetStdCProcs(). To use one of these function pointers to execute a standard function, use the CallQD***Proc() routines.
The sample below installs a customized line drawing procedure, then beeps, and then calls the original procedure. After drawing ten lines, it removes the pointer to the customized line drawing routine and draws another ten lines.

```c
//******************************************
CQDProcs qd_procs;
CQDProcs old_qd_procs;

void SetupCustomLineProc( CWindowPtr pWindow );
void BeepingLine( Point newPt);
void ResetStdProcs( CWindowPtr pWindow );

/****************************************************************************
void StandardDrawing()
{
    int i;
    long tickCount;

    SetupCustomLineProc(pWindow);

    for (i = 1; i < 10; i++)
    {
        MoveTo( 0, i * 10);
        LineTo( 100, 100);
        Delay(15, &tickCount);
    }

    ResetStdProcs(cGrafPort * pWindow);

    for (i = 11; i < 20; i++)
    {
        MoveTo( 0, i * 10);
        LineTo( 100, 100);
        Delay(15, &tickCount);
    }
}

/****************************************************************************
void SetupCustomLineProc( CWindowPtr pWindow )
{
    // get the current set of QuickDraw's standard routines
    SetStdCProcs( &qd_procs );
```
old_qd_procs = qd_procs;

// replace our line drawing proc for the standard one
qd_procs.lineProc = NewQDLineProc(BeepingLine);

pWindow->grafProcs = &qd_procs;
}

void BeepingLine( Point newPt)
{
    SysBeep(20);

    // call QuickDraw's original line drawing proc
    CallQDLineProc( old_qd_procs.lineProc, newPt );
}

void ResetStdProcs( CWindowPtr pWindow )
{
    pWindow->grafProcs = nil;
}

Related FAQs

- See also FAQ 6-28, "How do you deactivate (gray-out) static text items and edit text items in a dialog?"
- See also FAQ 14-13, "How can I change the size and style of text in my lists?"
- See also FAQ 23-3, "What are the QuickDraw 'bottleneck' functions?"
- See also FAQ 24-16, "How can my application process picture comments when drawing a picture?"
Are the QuickDraw “bottleneck” functions called at interrupt time?

No, the QuickDraw “bottleneck” functions are not called at interrupt time. They are called by QuickDraw while it is performing its normal drawing or calculating functions.

Topic-Related FAQs

- See also FAQ 2-13, “What are the RGB values of the eight colors used in Basic QuickDraw?”
- See also FAQ 6-9, “Why can I only draw in the eight basic colors in my dialog?”
- See also FAQ 12-11, “How can I get the image of a screen (as in a screen capture)?”
- See also FAQ 12-15, “If my window overlaps multiple screens with different depths, how can I draw the contents of the window properly on all screens?”
- See also FAQ 16-29, “How can I draw to a window while a menu is pulled down obscuring the window?”
- See also FAQ 22-15, “How do I synchronize my drawing to a VBL for smooth animation?”
- See also FAQ 24-5, “How can I create a picture from a pixmap?”
- See also FAQ 31-12, “How can I determine the current text characteristics (font, size, mode, and style) that a graphics port will use to draw text?”
- See also FAQ 33-14, “Is the origin of a window the upper-left corner of the window, or the upper-left corner of the drawing area, below the title bar?”
- See also FAQ 33-17, “How do I get the GrafPort of the Desktop?”
- See also FAQ 33-21, “How can I tell what part of my window needs to be redrawn in response to an update event?”
Most Macintosh programs make use, in one way or other, of pictures and 'PICT' resources. In this chapter, you'll learn how to create a picture in memory, both from a pixmap and from QuickDraw commands that you control. You'll learn how to create a picture offscreen, how to rotate a picture, even how to turn a window's contents into a picture. You'll learn how to read and write a PICT file, how to get a picture's color table, and more.

There are five subsections in this chapter:

- Definitions
- Creating Pictures
- Drawing Pictures
- Picture Files
- Getting Picture Information
Definitions

This section of the QuickDraw: Pictures explains how QuickDraw defines what a picture is. Manipulations of pictures form the basis of this chapter.

What is a picture?

A picture is a series of QuickDraw commands that can be played back by the DrawPicture() function to draw an image on a graphics port.

Creating and Releasing Pictures

This section of the QuickDraw: Pictures chapter answers FAQs about creating and releasing pictures. It discusses how to combine two pictures into one and how to create a picture of the contents of a window. It also answers questions about some problems that may arise when recording a picture.

How do I create a picture?

Pictures are created by recording drawing operations. OpenPicture() and OpenCPicture() start the recording procedure and ClosePicture() stops the recording. Any drawing actions that occur during the recording process become part of the picture.

OpenPicture() is used to create the original version 1 picture format on basic graphics ports and the version 2 picture format on color graphics ports. The version 1 picture format supports only black-and-white drawing operations at 72 dpi. The version 2 picture format supports color drawing at 72 dpi.

OpenCPicture() is used to create the extended version 2 picture format on all computers running System 7, including those that only support basic QuickDraw. This format allows you to specify a resolution for the picture in both color and black-and-white.

To create a picture with OpenPicture():

```c
void DoCreatePicture( void ) {
```
RgnHandle oldClip;
PicHandle hPict;
Rect pictRect = {10,10,100, 100};

OldClip = NewRgn();
GetClip(oldClip);
ClipRect( &pictRect );
hPict = OpenPicture( &pictRect ); // start recording
// Do your drawing
MoveTo(20, 20); Drawstring("\pMyText");

ClosePicture(); // stop recording
SetClip(oldClip);
DisposeRgn(oldClip);

To create a picture with OpenCPicture():

/***********************
void DoCreateCPicture( void )
{
    RgnHandle oldClip
    PicHandle hPict;
    OpenCPicParams ocpParams;
    Rect pictRect = {10,10,100, 100};

    ocpParams.srcRect = pictRect;
    ocpParams.hRes = 0x00480000; // 72 dpi
    ocpParams.vRes = 0x00480000; // 72 dpi
    ocpParams.version = -2;

    oldClip = NewRgn();
    GetClip(oldClip);
    ClipRect( &ocpParams.srcRect );
hPict = OpenCPicture( &ocpParams ); // start recording
// Do your drawing
MoveTo(20, 20); Drawstring("\pMyString");

ClosePicture(); // stop recording
SetClip(oldClip);
DisposeRgn(oldClip);
}
If I want to record a picture, do I have to do it offscreen so it won't disrupt my windows?

No. OpenCPicture() and OpenPicture() call HidePen() so the pixmap is not affected and no drawing occurs on the screen while the picture is open. This will not be true if you call the ShowPen() procedure just after you open the picture or if you called ShowPen() previously without balancing it with a call to HidePen().

How can I combine two pictures into one?

To combine two pictures into one, begin recording a picture by calling OpenCPicture(), draw both pictures with DrawPicture() and then call ClosePicture().

The code below loads two 'PICT' resources and creates a picture by drawing the loaded pictures while recording.

```c
PicHandle CombineTwoPicts(void) {
    PicHandle hPict, hSrcPict1, hSrcPict2;
    OpenCPicParams ocpParams;
    Rect pictRect = {0, 0, 200, 200},
    rSrcl = {0,0,100,200},
    rSrc2 = {100,0, 200, 200};
    ocpParams.srcRect = pictRect;
    ocpParams.hRes = 0x00480000;
    ocpParams.vRes = 0x00480000;
    ocpParams.version = -2;
    hSrcPict1 = GetPicture(128);
    hSrcPict2 = GetPicture(129);
    ClipRect( &ocpParams.srcRect );
    hPict = OpenCPicture( &ocpParams ); // start recording
    // Do your drawing for picture
    DrawPicture(hSrcPict1, &rSrcl);
    DrawPicture(hSrcPict2, &rSrc2);
```
ClosePicture();  // stop recording
ReleaseResource( (Handle)hSrcPict1 );
ReleaseResource( (Handle)hSrcPict2 );
return( hPict );

**How can I create a picture from a pixmap?**

You can create a picture from a pixmap by calling CopyBits() to copy the pixmap while recording a picture. Use OpenPicture(), CopyBits(), and ClosePicture().

**Related FAQ**

See also FAQ 24-10, "How do I create a picture of the contents of a window?"

**Related Topic**

See also Chapter 23, "QuickDraw: Drawing."

**Why do pictures I record sometimes draw as empty space?**

You may be running into problems with an invalid clipping region. Always call ClipRect() to set an appropriate clipping region before calling OpenPicture(). If you do not use ClipRect() to specify a clipping region, OpenPicture() uses the clipping region specified in the current graphics port. If the clipping region is very large, scaling the picture, and therefore scaling the clipping region, may result in an invalid region which would prevent DrawPicture() from drawing the picture. When a graphics port is initialized, its clipping area is set to the rect of dimensions [-32767, -32767, 32727, 32727]. If the clipping region is too small, some or all of the drawing will be clipped. If you set the clipping region equal to the destination size of the picture you will avoid these problems.

```c
PicHandle RecordMyPicture( Rect *pictRect )
{
    PicHandle       hPict;
```
OpenCPicParams  ocpParams;

ocpParams.srcRect = *pictRect;
ocpParams.hRes = 0x00480000;
ocpParams.vRes = 0x00480000;
ocpParams.version = -2;

ClipRect( &ocpParams.srcRect );
hPict = OpenCPicture( &ocpParams ); // start recording

// Do your drawing for picture
ClosePicture();  // stop recording

return( hPict );

How can I determine if the large picture I am recording with OpenCPicture() recorded successfully or if I ran out of memory?

You can test whether or not a picture was created successfully by checking the picFrame field. If unsuccessful, the picFrame is an empty rectangle. The routine below draws a picture and tests to see if it was successful.

//***********************
Boolean RecordPicture( PicHandle hPict )
{
  WindowPtr myWindow

  ...

  // start recording
  hPict = OpenPicture( &(myWindow->portRect) );

  // Do picture drawing

  // stop recording
  ClosePicture();

  HLock( hPict );
if (EmptyRect(&(**hPict ).picFrame))
{
    // ran out of memory creating picture
    return( false );
}
HUnlock( hPict );
return( true );

Can I interrupt picture recording, execute some other calls, and then continue recording?

You can interrupt the recording of picture information by switching to another graphics port or you can save the value of the picSave field and set it to nil. To resume picture recording set the GrafPort to the one on which recording was initiated and restore the value of the picSave field.

//***************************************************************************
void InterruptPict(void)
{
    PicHandle hPict;
    Handle hPicSave;
    OpenCPicParams ocpParams;
    Rect pictRect = {0, 0, 200, 200},
        rSrc1 = {0, 0, 100, 100},
        rSrc2 = {100, 0, 200, 100},
        rSrc3 = {0, 100, 100, 200},
        rSrc4 = {100, 100, 200, 200};

    ocpParams.srcRect = pictRect;
    ocpParams.hRes = 0x00480000;
    ocpParams.vRes = 0x00480000;
    ocpParams.version = -2;

    ClipRect( &ocpParams.srcRect );
    hPict = OpenCPicture( &ocpParams );

    InvertRect( &rSrc1);

    // INTERRUPT recording
Why can’t I put CopyMask() in my picture? What other routines don’t work in picture definitions?

CopyMask() is one of a handful of functions that are not recorded in a picture. These routines are:

CopyMask(), CopyDeepMask(), SeedFill(), SeedCFill(),
CalcMask(), CalcCMask() and PlotIcon().

How do I create a picture of the contents of a window?

To create a picture of the contents of a window you use CopyBits() on the window’s pixmap while recording a picture.

1. First set the clipping rectangle to the dimensions of the window and start recording the picture by calling OpenPicture().

2. Next copy the contents of the window’s pixmap to itself with CopyBits() and specify the window’s own portBits and portRect fields (OpenPicture() hides the pen so drawing to a window will not disturb its pixmap).

3. When the operation is complete, close the picture.
related Topics

See also Chapter 12, "Graphics Devices."

See also Chapter 33, "Windows."

How can I control the resolution of a picture that I create?

When you create a picture, you can specify the horizontal and vertical resolution at which the picture is to be created by setting the hRes and vRes fields of the OpenCPicParams structure. To create a picture of 72 dpi, set the fields to 0x00480000. To create a picture of 300 dpi, set the fields to 0x012C0000. The sample below creates a 300 dpi picture.

```c
#define kPict72dpi 0x00480000
#define kPict300dpi 0x012C0000

PicHandle RecordMyPicture()
{
    PicHandle hPict;
    OpenCPicParams ocpParams;
    Rect r.pictRect = {50, 50, 200, 200};
```


```c
// set pict params, picture at 300 dpi
ocpParams.srcRect = pictRect;
ocpParams.hRes = kPict300dpi;
ocpParams.vRes = kPict300dpi;
ocpParams.version = -2;

ClipRect( &ocpParams.srcRect );
// start recording
hPict = OpenCPicture( &ocpParams );

// Do your drawing for picture
r = pictRect
InsetRect(&r, 5, 5);
FrameRect(&r);
InsetRect(&r, 5, 5);
FrameRect(&r);

ClosePicture(); // stop recording
return( hPict );
```

**To destroy a PicHandle, when do I call KillPicture(), ReleaseResource(), and DisposeHandle()?**

Use `KillPicture()` to release memory that is occupied by a picture created by `OpenPicture()` or `OpenCPicture()`. This is equivalent to calling `DisposeHandle()`.

`KillPicture(myPicture)` is equivalent to `DisposeHandle(Handle(myPicture))`.

To release a picture that is a resource obtained by calling `GetPicture()`, use `ReleaseResource()`.

A picture created by `OpenCPicture()` and assigned to a window for drawing on the background using the Window Manager call `SetWindowPic()` is automatically released when `DisposeWindow()` is called. `DisposeWindow()` automatically calls `KillPicture()`. If the picture assigned to the window is stored in a 'PICT' resource, then you must call `ReleaseResource()` before `DisposeWindow()` or `CloseWindow()` is called.
Drawing Pictures

This section of the QuickDraw: Pictures chapter answers FAQs on producing special effects when drawing pictures. It shows how to draw a picture with a transparent background so images under a picture will still be visible in the white spaces of the picture. It also discusses rotating a picture and centering a picture in the middle of a window.

**How can I draw a picture with a transparent background instead of a white background?**

`DrawPicture()` always draws in `srcCopy` mode. To draw the picture with a transparent background (as shown in Figure 24-1), draw the picture into an offscreen `GWorld` and then call `CopyBits()` to copy it into your window in transparent mode.

![Figure 24-1: The picture of the light bulb was drawn over the picture of the map using `DrawPicture()`.

![Figure 24-2: The picture of the light bulb was drawn over the picture of the map using `CopyBits()` and the transparent mode.](image)
The example below draws the picture of resource ID `pictId` to the current color GrafPort.

```c
void DrawTransparentPict( short pictId)
{
    PicHandle hPict;
    GWorldPtr curport, myworld;
    GDHandle curdevice;
    QDErr err;
    Rect rPict;

    hPict = GetPicture(pictId);
    rPict = (*hPict)->picFrame;
    OffsetRect(&rPict, -rPict.left, -rPict.top);
    // origin at (0, 0)

    // save current context
    curport = nil;
    curdevice = nil;
    GetGWorld(&curport, &curdevice);

    // create an offscreen graphics world
    myworld = nil;
    err = NewGWorld(&myworld, 0, &rPict, nil, nil, 0);

    // if GWorld was created successfully,
    // draw the picture into it
    if ( myworld )
    {
        LockPixels(myworld->portPixMap);
        SetGWorld(myworld, nil);
        EraseRect(&rPict);
        DrawPicture(hPict, &rPict);

        // copy the picture into the current window
        SetGWorld(curport, curdevice);
        ForeColor(blackColor);
        BackColor(whiteColor);
        CopyBits((BitMap*)(*(myworld->portPixMap)),
                 (BitMap*)(*(curport->portPixMap)),
                 &rPict, &rPict, transparent, nil);
    }
}
```
CHAPTER 24: QuickDraw: Pictures 437

// get rid of offscreen world
UnlockPixels(myworld->portPixMap);
DisposeGWorld(myworld);

// release the memory of the picture resource
ReleaseResource((Handle)hPict);

How can I draw a picture in a mode other than srcCopy mode?

You can display a picture in any source mode by drawing it offscreen and then using CopyBits() to copy it to your window. CopyBits() allows you to specify a source mode to use when transferring an image.

Related FAQ

See also FAQ 24-13, “How can I draw a picture with a transparent background instead of a white background?”

How can I center a picture in a window?

To center a picture in a window, get the size of the window from the window’s portRect field and the size of the picture from the picture’s picFrame field. Then center the picture’s frame rectangle in the window’s rectangle.

The example below draws a picture centered inside a window.

/**********************
void CenterPictInWindow( PicHandle hPict,
                         WindowPtr pWindow )
{
    Rect pictRect;

    HLock((Handle)hPict);
    pictRect = (**(hPict)).picFrame;
    HUnlock((Handle)hPict);
}
CenterRect( pWindow->portRect, &pictRect );

DrawPicture( hPict, &pictRect);
}

void CenterRect( Rect boundingRect, 
    Rect *centeredRect )
{
    OffsetRect( centeredRect, boundingRect.left - 
    centeredRect->left, 
    boundingRect.top - 
    centeredRect->top);

    OffsetRect( centeredRect, (boundingRect.right - 
    centeredRect->right)/2, 
    (boundingRect.bottom - 
    centeredRect->bottom)/2);
}

How can my application process picture comments when drawing a picture?

To process picture comments, your application must replace the standard QuickDraw low level function used to process picture comments, StdComment(), with a customized routine that can handle them. The default StdComment() function of a graphics port does not process picture comments. To override this function you must store the customized function in the commentProc field in the CQDProcs structure of a graphics port.

Related FAQs

☐ See also FAQ 6-28, “How do you deactivate (gray-out) static text items and edit text items in a dialog?”

☐ See also FAQ 14-13, “How can I change the size and style of text in my lists?”

☐ See also FAQ 18-1, “What is a picture comment?”

☐ See also FAQ 18-14, “How can I print good looking dashed lines?”
How do I rotate a picture?

There is no function to rotate a picture. You must manually manipulate the bits in the picture's pixmap. This is best accomplished by creating an offscreen GWorld, manipulating the picture's bits onto the GWorld, and then using CopyBits() to transfer the image to the destination graphics port.

Related FAQs

- See also FAQ 23-17, "How do I flip a pixmap (put up its mirror image)?"
- See also FAQ 23-19, "How do I rotate a pixmap?"
- See also FAQ 31-18, "How can I draw rotated text?"

Picture Files

This section of the QuickDraw: Pictures chapter answers FAQs about PICT files. It discusses the format of a picture file, how to read from a PICT file, and how to write a picture to a PICT file.

What is the format of a PICT file?

A PICT file contains a 512 byte header that is reserved for your application's own purposes followed by the data that make up a picture structure.

Related Topic

- See also Chapter 8, “Files: File Manager.”
How do I write a picture to a PICT file?

A PICT file contains a 512 byte header that is reserved for your application's own purposes followed by the data that make up a picture structure.

First create a file and open up the data fork (the picture information is stored in the data fork, not the resource fork). Write 512 bytes to the file. Then write the picture data referenced by your PicHandle. The amount of data to write is determined by calling GetHandleSize() on the PicHandle.

```c
void WritePictToFile(
    FSSpec *mySpec,
    PicHandle hPict)
{
    short fRefNum;
    long zeroData = 0L;
    long zeroDataLen, pictLength;
    int i;
    OSErr err;

    FSpCreate( mySpec, 'TEST', 'PICT', smSystemScript );
    err = FSpOpenDF(mySpec, fsRdWrPerm, &fRefNum );

    zeroDataLen = sizeof(long);
    for ( i = 0; i < 512 / zeroDataLen; i ++ )
    {
        err = FSWrite( fRefNum, &zeroDataLen, &zeroData);
    }

    pictLength = GetHandleSize((Handle)hPict);
    HLock((Handle)hPict);
    err = FSWrite(fRefNum, &pictLength, *hPict);
    HUnlock((Handle)hPict);

    FSClose( fRefNum);
}
```

Related Topic

See also Chapter 8, "Files: File Manager."
How do I read a picture from a PICT file?

A PICT file contains a 512 byte header that is reserved for your application’s own purposes followed by the data that makes up a Picture structure. To read the picture data from a PICT file, open up the data fork of the file (the picture information is stored in the data fork, not the resource fork). Then move 512 bytes into the file to get to the start of the picture data. Get the size of the picture by calling GetEOF() to get the logical EOF and subtract the 512 bytes of the header from it. Read this number of bytes from the file into the pointer referenced by a locked, newly created handle of the same size. Then close the file — you have your picture.

//********************
PicHandle DoReadPictFromFile(FSSpec *mySpec)
{
    PicHandle hPict;
    short fRefNum;
    long fileSize;
    long pictLength;
    OSErr err;

    err = FSpOpenDF(mySpec, fsRdWrPerm, &fRefNum);
    GetEOF(fRefNum, &fileSize);
    SetFPos(fRefNum, fsFromStart, 512);

    pictLength = fileSize - 512;
    hPict = (PicHandle)NewHandle(pictLength);
    HLock((Handle)hPict);
    err = FSRead(fRefNum, &pictLength,*hPict);
    HUnlock((Handle)hPict);
    FSClose(fRefNum);

    return(hPict);
}

Related Topic

See also Chapter 8, “Files: File Manager.”
Getting Picture Information

This section of the QuickDraw: Pictures chapter answers FAQs about retrieving information about pictures. It discusses how to get the palette of a picture and how to get the picture’s bounding box.

How do I get a palette or color table of a picture?

You can retrieve the most frequently used colors in a picture or the widest range of colors in a picture by calling GetPictInfo(). If you want the colors returned in a palette, then pass 'returnPalette' in the third parameter of GetPictInfo(). If you want the colors returned in a color table, then pass 'returnColorTable'. To get the most used colors, pass 'popularMethod' in the fifth parameter. To get the widest range of colors, pass 'medianMethod'.

The example below returns the handle to a palette containing the 256 most used colors in the picture hPict.

```c
//********************
PaletteHandle GetPictPalette( PicHandle hPict )
{
    PictInfo    thePictInfo;
    OSErr       err;
    err = GetPictInfo( hPict, &thePictInfo, returnPalette, 256, popularMethod, 0);
    return( thePictInfo.thePalette );
}
```

The example below returns the handle to a color table containing a range of 256 colors in the picture hPict. Some of the colors may not actually appear in the picture.

```c
//********************
CTabHandle GetPictColorTable( PicHandle hPict )
{
    PictInfo    thePictInfo;
    OSErr       err;
    err = GetPictInfo( hPict, &thePictInfo, returnColorTable, 256, medianMethod, 0);
```
Related Topic

See also Chapter 2, "Color."

**How do I get the size (bounding rectangle) of a picture?**

The bounding rectangle of a picture is stored in the `picFrame` field of the `Picture` structure. It is accessed by:

```c
PicHandle myPictHandle;
(*myPictHandle)->picFrame;
```

**Topic-Related FAQs**

- See also FAQ 2-29, "How can I get a picture that does not use the default palette to display properly in a window?"
- See also FAQ 6-8, "How can I put a PICT item in a background of a dialog without obscuring the other items?"
- See also FAQ 13-7, "Why doesn't PlotCIcon( ) get recorded as part of a picture by OpenPicture( )/ClosePicture( )?"
- See also FAQ 16-30, "How do I create a menu of pictures?"
- See also FAQ 18-1, "What is a picture comment?"
- See also FAQ 23-17, "How do I flip a pixmap (put up its mirror image)?"
- See also FAQ 23-29, "How do I put contents of a GWorld into a picture?"
If there is one concept that forms the basis of Macintosh programming and distinguishes Macintosh programming from any other platform, that concept is resources. The Resource Manager is another widely used yet frequently misunderstood Toolbox manager. The questions in this chapter clear up a lot of frequently misunderstood resource programming issues. Among other things, you'll learn the proper way to load and update a resource. You'll learn about the resource map, the resource search path, how to detach a resource, and much more.

There are three subsections in this chapter:

- Definitions
- Using Resources
- Resource Files
Definitions

This section of the Resources chapter defines the resource map, which will provide the context for a fuller understanding of some of the answers to the FAQs below.

What is the resource map?

The resource map is a table that stores information for each resource in a resource fork. It stores the resource type, ID, name, attributes, and location in the file for each resource. When the resource fork of a file is opened, the resource map is loaded into memory where it stays until the file is closed. When a resource is loaded into memory, its entry in the resource map in memory is set to the resource's handle. If a resource in the resource map is not loaded into memory, its entry in the resource map is set to nil.

Using Resources

This section of the Resources chapter answers FAQs about using resources in an application. It covers how to get a resource from a resource file, when and why you should detach a resource, and how you load a resource into the System heap.

Are resource types case sensitive?

Yes, resource types are case sensitive and can consist of any characters. Apple reserves all resource types that consist of all lowercase letters, all spaces, or all international characters. If you define your own resource type, it must be unique and consist of all uppercase letters.

How do I put a code resource in my application?

To put a code resource in your application, you must first create a separate project to create and build the resource. Set the project type to a code resource and the resource type to the appropriate value, 'MDEF', 'LDEF', 'CODE', etc. Then build the project. This will create a resource file that contains the newly created code resource. Copy this code resource into the resource file of your application.
How you use this resource depends upon the type of the resource. For example, if you are adding an ‘MDEF’ to your application, you add the resource ID of the ‘MDEF’ resource to the mdef field in the menu resource. If you are adding a ‘CODE’ resource you load it with GetResource('CODE', resId) and then cast the resource handle's master pointer into a procedure pointer. The handle should be locked so that it will not move out from under you when you do something that causes memory to move. If you want the code resource to stay around after your application closes, then make sure that you call DetachResource() to detach it from the resource map. If DetachResource() is not closed then the code resource will be released from memory when the file storing the resource is closed.

The sample below loads a code resource, locks it, and then marks it as a VBL task:

Handle hCode;

hCode = GetResource('CODE', 1000);
HLock(hCode);
myTask.vblAddr = NewVBLProc(*hCode);

What does detaching a resource do, and when should I do it?

Detaching a resource sets the value of the resource’s handle in the resource map stored in memory to nil. After this is done, the Resource Manager no longer recognizes the handle as pointing to a resource. If your application calls a Resource Manager routine to get a resource that has been released, the Resource Manager assigns a new handle to it. When a resource file is closed, it releases all resources to which its resource map points. When a resource is detached, its handle is not released when the resource file is closed. This procedure is necessary when a resource must stay in memory after its file is closed such as a VBL task’s ‘CODE’ resource that gets loaded into System memory and executes after the installing application is closed.

How do I load a resource into the System heap?

The easiest way to load a resource into the System heap is to set the System heap bit and the locked bit of the resource. When a resource is loaded with GetResource(), it is loaded into the System heap. You can also manipulate the System heap yourself. You can set the current heap to be the System heap and then load the resource. Do not forget to detach the resource with DetachResource()
when you allocate a code resource directly into System memory so it won't be discarded when the resource file is closed. You could load the resource into your application heap, get the size of the resource, allocate a block of memory in the System heap, and copy the resource into it. In this case you won't have to call DetachResource(). This is necessary for 'CODE' resources that are loaded from an extension and continue to execute after the extension has stopped executing.

The code below loads a code resource into the System heap by setting the System heap as the current zone:

```c
    //***********
    void LoadInitResource()
    {
        THz   oldZone;
        Handle   hFunction;

        oldZone = GetZone();
        SetZone( SystemZone() );

        hFunction = GetResource( 'INIT', -4048);
        DetachResource( hFunction );

        SetZone(oldZone);
    }
```

The code below loads a code resource into the current heap and copies it into the System heap. In this case it is not necessary to call DetachResource().

```c
    //************
    void LoadInitResource()
    {
        Handle   hFunction;
        Size    funcSize;
        Ptr     ptrSys;

        hFunction = GetResource( 'INIT', -4048 );
        funcSize    = SizeResource( hFunction);
        ptrSys      = NewPtrSys( funcSize );
        BlockMove( *hFunction, ptrSys, funcSize );
    }
```
How can I get the machine name and user name?

The machine name and user name are stored in 'STR' resources in the system. The machine name is stored in the 'STR' resource with ID -16413. The user name is stored in the 'STR' resource with ID -16096. You can retrieve the string by calling `GetString()`. The example below gets a handle to a string holding the user name.

```c
hName = GetString(-16096);
```

What are the valid resource IDs that I can use for my resources in my application?

Any resource ID less than 128 is reserved by Apple. As a general rule you can use resource IDs 128 to 32767. There are some exceptions. The ID range for definition procedures is 128 to 4095 since the ID is stored in only 12 bits.

The Resource Manager functions do not return a value; how do I tell if they were successful or if an error occurred?

After you call a Resource Manager function you can call the function `ResError()` to see if an error occurred. There are some cases where `ResError()` will return `noErr` when a Resource Manager routine was unable to perform its operation. You can check the routine description for such cases.

How can I determine the order that resources will be returned by `GetIndResource()`?

Resources are returned by `GetIndResource()` in the order that they were added to the resource file, not in an order determined by their ID number. However, Apple
states this is subject to change. The best way to retrieve resources in a specific order by ID number is to use \texttt{GetIndResource()} to create a list of handles to the resources and then use \texttt{GetResInfo()} to determine the resource ID numbers. You can then sort the list.

**How can I tell if a purgeable resource has been purged?**

When a purgeable resource has been purged its master pointer is set to nil. If this happens, the resource must be loaded again. The code below tests if a resource has been purged and, if so, loads the resource again.

```c
If (*myResHandle == nil)
    LoadResource( myResHandle );
```

**How do I update a Macintosh resource?**

To modify a resource, follow these steps:

1. Load the resource into memory with \texttt{GetResource()}. Once you have a handle to the resource, you can make the changes to it.
2. Call \texttt{ChangedResource()} to mark the resource as modified in the resource map.
3. Call \texttt{WriteResource()} to force the modified resource to be written to the resource file on disk.

**Resource Files**

This section of the Resources chapter answers FAQs about working with resource files. It describes the search path the Resource Manager follows to search for a resource and how to limit the search to a single file. It also shows how to save new resources in a resource file. Finally, it describes how to get various types of information about resource files.
What does the "1" stand for in the Resource Manager routines?

A Resource Manager routine that includes a "1" in its name will look exclusively in the current resource file for a particular resource. Functions that do not include a "1" in their names search all of the open resource files, including the System's resource files.

Related Topic

See also Chapter 8, "Files: File Manager."

Why can't I open the resource fork of a file I just created with FSpCreate()?

FSpCreate() creates both forks of a new file but does not create a resource map. Before you can open the resource fork of the file, you will have to call one of the Resource Manager procedures that creates the resource map, for example, CreateResFile(), HCreateResFile(), or FSpCreateResFile(). To access the resource fork, you must call one of the resource fork-specific open routines such as OpenResFile(), HOpenResFile(), or FSpOpenResFile().

How do I save a resource to a file?

To save a resource, you need a handle to the resource you want saved and the file reference number of an open resource file. Next you need to determine the resource type and an ID for the resource. A unique ID for a specified type within the current resource file can be determined by UniqueId(). Once you have all of the relevant information, you call AddResource() to convert the handle into a resource handle recognized by the Resource Manager and saved in the resource map. AddResource() automatically sets the resChanged bit of the resource attribute which signifies to the Resource Manager that the resource data and resource map need to be written to disk. To ensure that the modifications are written to disk, call WriteResource().

Related FAQs

See also FAQ 8-28, "How do I read data from and write to a file?"
How do I copy resources from one resource file to another?

To copy a resource from one file to another you must first get a handle to its resource data. You must then call DetachResource() to free the resource from the original resource file. If DetachResource() is not called, the resource will not be added to the destination resource files resource map. Now you are ready to add the resource to the destination resource file. The current resource file must be made the destination resource file with UseResFile(). Then get a unique resource ID that can be assigned to the added resource. AddResource() creates a new entry for the resource in the memory resident resource map and sets the entry's location to refer to the resource's data and sets the resChanged attribute to 1. It does not modify the actual resource file. WriteResource() writes the added resource to the disk.

The example below copies an 'ICON' resource with ID 128 from the resource file srcResFile to the resource file destResFile.

```c
//**********************************************************
v
```
Related FAQs

- See also FAQ 8-28, “How do I read data from and write to a file?”
- See also FAQ 25-14, “How do I save a resource to a file?”

Related Topic

- See also Chapter 8, “Files: File Manager.”

**How can I set the current resource file?**

To make an open resource file the current resource file, call the routine UseResFile(). If you pass 0 to UseResFile() the System file is made the current resource file. The current resource file is searched first when locating a resource.

Related Topic

- See also Chapter 8, “Files: File Manager.”

**How do I access resources from another file?**

- See FAQ 25-24, “How does an application access the resources in its own resource fork? In the System file?”

**How can an application get the file reference number of its own resource file?**

An application can get the file reference number to its own resource fork by calling CurResFile() when it starts up. This could be used if an application wanted to make its own resource file current after opening other resource files.

Related Topic

- See also Chapter 8, “Files: File Manager.”
**Given a handle to a resource, how can I determine which resource file it came from?**

Given a handle to a resource you can determine the resource file that it came from by calling the `HomeResFile()` function. `HomeResFile()` returns 0 if the resource is from the System resource file, 1 if the resource file is resident in ROM, or the file reference number of the source resource file.

The example below calls `HomeResFile()` to get the resource's file reference number and checks its value. If the resource is from the System file it gets the actual file reference number of the System resource file from the low memory global SysMap. It then calls `PBGetFCBInfo()` to get information about the file.

```c
II****************************
OSErr GetResourceSourceFile ( Handle hResource,
    short *vRefNum,
    long *parID,
    Str255 name)
{
    OSErr err;
    FCBPBBRec fcbPB;

    // Get resource file reference number associated with the resource
    fcbPB.ioRefNum = HomeResFile(hResource);
    err = ResError();
    if (err == noErr)
    {
        // check if resource is in ROM
        if (fcbPB.ioRefNum != 1)
        {
            // check if resource if from the system file - if it was
            // then get the System file's ref num
            if (fcbPB.ioRefNum == 0)
            {
                fcbPB.ioRefNum = LMGetSysMap();
            }
            // get the file information from the file
            // control block
            fcbPB.ioNamePtr = name;
        }
    }
}
fcbbPB.ioVRefNum = 0;
fcbPB.ioFCBIindx = 0;
err = PBGetFCBInfo(&fcbPB, false);
if (err == noErr)
{
    *vRefNum = fcbPB.ioFCBVRefNum;
    *parID = fcbPB.ioFCBParID;
}
else
{
    // resource is from ROM, return paramErr to
    // indicate invalid file
    err = paramErr;
}
return (err);

Related Topic

See also Chapter 8, “Files: File Manager.”

If I have multiple resource files open, what is the order in which they are searched for a resource?

The first resource file that is searched is the one designated as the current resource file. When your application starts, this is the application’s resource file. Whenever a new resource file is opened, it becomes the current resource file. The file that had been the current resource file moves to the second position in the search list. By default, the System’s resource file is searched last. An open resource file can be made the current file by calling UseResFile().

Related FAQ

See also FAQ 25-24, “How does an application access the resources in its own resource fork? In the System file?”
How can I get the number of resources of a particular type in a resource file?

There are two routines, `CountResources()` and `Count1Resources()`, that return the number of available resources of a particular type. `CountResources()` counts all resources in all open resource files available to your application, including the System file. Use `Count1Resource()` to count the resources in the current resource file only.

How can I limit the searching for a resource to check only a single file?

Many Resource Manager routines have related routines that perform the same function but limit the scope of their actions to one file, the current resource file. These functions have the character “1” in their names. `Get1Resource()` retrieves resource data from the current resource file. `Unique1ID()` returns an id that is unique with respect to the current file.

Is there a maximum number of items for Macintosh resources in a resource file?

The maximum number of resources that a resource file can contain is 2727. However, it is recommended that you have no more than 500.
How does an application access the resources in its own resource fork? In the System file?

When an application starts up it automatically has access to the resources from its own resource fork and the resources from the System file. To access resources from any other files, it must open those files with FSpOpenResFile() or HOpenResFile(). The application searches all of the resource files it has opened.

Related FAQ

- See also FAQ 25-20, “If I have multiple resource files open, what is the order in which they are searched for a resource?”

Related Topic

- See also Chapter 8, “Files: File Manager.”

Topic-Related FAQs

- See also FAQ 1-10, “Can I send an Apple event from a code resource?”
- See also FAQ 8-5, “What are resource and data forks of a file?”
- See also FAQ 8-28, “How do I read data from and write to a file?”
- See also FAQ 27-3, “How do I play a ‘snd‘ resource?”
- See also FAQ 27-32, “Why can’t I get a ‘snd‘ resource? I know it is in my resource file.”
Scrap Manager

The Scrap Manager is the set of routines that controls and implements the Macintosh clipboard, the device that allows you to share data between applications. In this chapter, you'll learn about the concept of scrap types, how to initialize the scrap, how to read and write data to the scrap, how the scrap gets copied out to disk, and more.

There are three subsections in this chapter:

- Definitions
- Getting Information About the Scrap
- Using the Scrap Manager

Definitions

This section of this chapter defines the scrap. The chapter as a whole is concerned with the scrap's functions and use.
**What is the "scrap"?**

The scrap is the storage area available to an application, maintained by the Scrap Manager, to hold the last data cut or copied by the user. This storage area can reside either in memory or on disk.

**Getting Information About the Scrap**

This section of the Scrap Manager chapter answers FAQs on getting different types of information about the scrap. It describes how you can detect if there is data in the scrap, how you can determine all of the types of data in the scrap, and other types of information.

**How can I tell if there is data in the scrap?**

To see if there is any data stored in the clipboard, call `InfoScrap()` and test the `scrapSize` field of the returned `ScrapStuff` structure.

**How can I tell what types of data are in the scrap?**

You can determine what types of data are in the scrap by traversing through all of the data stored in the scrap. To do this, you first get the handle to the scrap stored in the `scrapHandle` field of the `ScrapStuff` structure returned by the `InfoScrap()` function. Then, after locking the handle, begin reading the data stored in the scrap. The data is stored in the following order:

- Four bytes representing the `OSType` of the data
- Four bytes storing a long representing the size of the data
- Number of bytes determined above containing the actual data stored in the scrap

Read the first four bytes to get the type of the data. Then read the next four bytes to get the length of the data stored in scrap. Then read the specified number of bytes to get the actual data stored. To do this, create a handle of the size of the data, lock it, and then do a `BlockMove()` to copy the data to the handle. You can determine when you have read all of the data stored in the scrap by comparing the number of bytes that have been read with the total size of the scrap. The total scrap size is stored
in the scrapSize field of the ScrapStuff structure returned by the InfoScrap() function. The scrapSize is the sum of sizes of all data plus 8 bytes for each individual scrap data (to store the type and the size).

The sample below traverses the scrap, reading the data type, data size, and then the data itself for each scrap entry.

```c
/******************
void TraverseScrap()
{
    PScrapStuff myStuff;
    Ptr currentPtr;
    long lTotal = 0, itemSize;
    OSType scrapType;
    Handle dataHandle;

    dataHandle = NewHandle(0);

    myStuff = InfoScrap();
    HLock(myStuff->scrapHandle);

    currentPtr = *(myStuff->scrapHandle);

    while ( lTotal < myStuff->scrapSize )
    {
        // get the scrap type
        scrapType = *((OSType *)currentPtr);

        // move to get the size of item
        currentPtr += 4;
        itemSize = *((long *)currentPtr);

        // move to and get the data
        currentPtr += 4;
        SetHandleSize(dataHandle, itemSize);
        HLock(dataHandle);
        BlockMove( currentPtr, *dataHandle, itemSize);
        HUnlock(dataHandle);

        // move pointer to next data in scrap
        currentPtr += itemSize;

        // increment total size read from scrap
        lTotal += (itemSize + 8);
    }
```
HUnlock(myStuff->scrapHandle);  

**How can I determine if a certain type of data is in the desk scrap?**

You can call GetScrap() with the second parameter, the type parameter, set to the type you want to check. If a scrap item exists, GetScrap() returns the size of the data. If it does not exist, the value noTypeErr is returned.

**How can I tell if the data in the scrap has been changed by another application?**

Each time the desk scrap is modified the scrapCount field of the ScrapStuff structure returned by the InfoScrap() function is also modified. In response to a resume event your application can compare the value of scrapCount to the value of the field at the corresponding suspend event to see if another application modified the scrap.

A better way of doing this is to test bit 1 in the message field of a resume event. There is a defined bit mask, convertClipboardFlag, that can be used to do a bitwise AND with the message field. If the field is set, then the scrap was changed.

```c
// we have a resume event, test if clipboard has been modified
if ( eventPtr->message & convertClipboardFlag )
{
    // scrap is modified
}
```

**How do I determine the size of the handle to pass to GetScrap()?**

You don’t have to. GetScrap() will resize the handle for you. You can pass in a valid handle of any size created with NewHandle().
Using the Scrap Manager

This section of the Scrap Manager chapter answers FAQs about how the scrap works and is integrated into an application. It shows how data is written to and retrieved from the scrap, how to incorporate the scrap into TextEdit, and how you can read all of the data on the scrap (including multiple items of the same type). It also covers when and why the scrap gets transferred from memory to a file.

What types of data can the desk scrap hold?

The clipboard can store any type of data. `PutScrap()` simply writes the data type and the size to the scrap and copies the specified number of bytes of the data to the scrap. The data type is up to the discretion of the programmer.

How do I read data from and write data to the scrap?

To put data in the desk scrap, call the `PutScrap()` function. The example below puts a picture in the desk scrap.

```c
HLock(hPicture);
PutScrap( GetHandleSize(hPicture), 'PICT', *hPicture );
HUnlock( hPicture );
```

To read data from the scrap, use the `GetScrap()` function. The example below retrieves a picture from the desk scrap.

```c
hPicture = NewHandle(0);
GetScrap( hPicture, 'PICT', &scrapOffset);
```

TextEdit fields use the private TextEdit scrap, not the desk scrap. Use `TECopy()`, `TECut()`, and `TEPaste()` with TextEdit.

When data is put on the scrap, does it replace the data that is already there?

No. New data is put at the end of the scrap. It does not remove existing data of the same type.
Why doesn't the data I copy onto the scrap get retrieved when I paste? Why does only the old data get pasted?

Before you call PutScrap() to place data in the scrap you must call ZeroScrap() to purge any data in the desk scrap. If the old data is not removed from the scrap, the new data is added to the scrap after it, even if data of the same type is already stored. Consequently it is possible that the scrap will contain two items of the same type. GetScrap() will always return the first scrap item.

How do I clear data from the scrap?

To clear all types of data from the scrap, call ZeroScrap().

How can I read all of the data from the desk scrap, including more than one item of the same type?

You can read all of the data in the scrap, including multiple items of the same type, by getting the handle to the scrap in memory or the name of the scrap file on disk and then traversing through all of the data yourself. First you must call InfoScrap() to determine if the scrap is in memory or on disk. If the scrap is in memory, the scrapHandle field of the PScrapStuff structure will contain a handle to the scrap. If the scrap is stored on disk then scrapHandle will be nil and the scrapName field of the PScrapStuff structure will contain the name of the scrap file. The following code shows how to test for the scrap location:

```
PScrapStuff myStuff;

myStuff = InfoScrap();
if ( myStuff->scrapHandle )
{
```
The data is stored in the scrap in the following format:

- 4 bytes representing the OSType of the data
- 4 bytes storing a long representing the size of the data
- number of bytes determined above containing the actual data stored in the scrap

Related FAQ

See also FAQ 26-3, “How can I tell what types of data are in the scrap?”

When does the scrap get copied to a file?

The System software allocates space in every application's heap to hold a copy of the desk scrap and a handle to this desk scrap. Each time an application is made active, the contents of the scrap of the previously active application is stored in its scrap. If there is not enough room in the application’s heap, the system copies the scrap to disk.

The contents of the scrap can be manually written to disk by calling UnloadScrap(). This causes the contents of the scrap to be written to the file whose name is pointed to by the scrapName field of the ScrapInfo global variable. The scrap can be loaded back into memory with LoadScrap().

Related Topic

See also Chapter 8, “Files: File Manager.”
**How can I tell if the scrap is in memory or has been written to a file?**

You can determine if the scrap has been written to a file by the `scrapState` field of the `ScrapStuff` structure returned by `InfoScrap()`. If `scrapState` is greater than 0, then the scrap is in memory. If it equals 0, then the scrap is being maintained on disk and the file name is stored in the `scrapName` field of `ScrapStuff`.

**Related Topic**

- See also Chapter 8, “Files: File Manager.”

---

**Why doesn’t GetScrap() and PutScrap() from a TextEdit field or DialogCopy() from a dialog affect the desk scrap?**

TextEdit uses its own private scrap, not the desk scrap accessed by `GetScrap()` and `PutScrap()`. To manipulate the TextEdit scrap, use the functions `TECut()`, `TECopy()`, and `TEPaste()`. `TEToScrap()` and `TEFromScrap()` can be used to transfer data between the TextEdit private scrap and the desk scrap.

Dialog `Cut()`, `DialogCopy()`, and `DialogPaste()` call the TextEdit scrap functions to pass data between the active TextEdit field of a dialog and the TextEdit scrap.

**Related Topic**

- See also Chapter 32, “Text: TextEdit.”
The Sound Manager makes the Macintosh one of the most sophisticated sound handling computers that has ever been created. The Sound Manager can create sounds ranging from a simple beep to a multiphonic, multichannel symphony. In this chapter, you'll learn about sound channels and the sound commands that queue up in those channels. You'll learn what causes the annoying static in your speakers when you play certain sounds, how to play sounds from a 'snd' resource and from a file, and how to play sounds asynchronously. You will also discover how to manipulate the different characteristics of a sound, such as the volume and speed at which a sound plays.

This chapter is not limited to the playing of sound, but also answers FAQs on how to record sound with your Macintosh. There are examples that illustrate how to record sounds synchronously and asynchronously, to a file, and to a 'snd' resource. You will even learn how to record a sound without displaying the sound recording dialog box.
There are five subsections in this chapter:

- Definitions
- Playing Sound
- Adjusting the Volume
- Recording Sound
- Using the Sound Manager

Definitions

This section of the Sound chapter answers FAQs about some important terms that are discussed in the following sections. It defines sound commands, the basic method of controlling the sound manipulation on the Macintosh, and methods used to playback and record sound.

What is double buffering?

Double buffering is a technique used by the Sound Manager to play large sounds from disk by utilizing two buffers. Initially only one of the two buffers is filled with sound data. While the sound in that buffer is being played, the second buffer is filled with the next sound data. While the sound in the second buffer is being played, the first buffer is filled with subsequent sound data, and so on. This method allows continuous sound playing while reading sound data from large files into memory.

What are sound commands?

A sound command is an instruction passed to a sound channel to affect the playing of sound from that channel. Each sound channel has its own queue in which it stores sound commands to be processed.

```c
nullCmd = 0;  // do nothing
```
- quietCmd = 3; {stop a sound that is playing}
- flushCmd = 4; {flush a sound channel}
- reInitCmd = 5; {reinitialize a sound channel}
- waitCmd = 10; {suspend processing in a channel}
- pauseCmd = 11; {pause processing in a channel}
- resumeCmd = 12; {resume processing in a channel}
- callbackCmd = 13; {execute a callback procedure}
- syncCmd = 14; {synchronize channels}
- availableCmd = 24; {see if initialization options are supported}
- versionCmd = 25; {determine version}
- totalLoadCmd = 26; {report total CPU load}
- loadCmd = 27; {report CPU load for a new channel}
- freqDurationCmd = 40; {play a note for a duration}
- restCmd = 41; {rest a channel for a duration}
- freqCmd = 42; {change the pitch of a sound}
- ampCmd = 43; {change the amplitude of a sound}
- timbreCmd = 44; {change the timbre of a sound}
- getAmpCmd = 45; {get the amplitude of a sound}
- volumeCmd = 46; {set volume}
- getVolumeCmd = 47; {get volume}
- waveTableCmd = 60; {install a wave table as a voice}
Macworld Mac Programming FAQs

- soundCmd = 80; {install a sampled sound as a voice}
- bufferCmd = 81; {play a sampled sound}
- rateCmd = 82; {set the pitch of a sampled sound}
- getRateCmd = 85; {get the pitch of a sampled sound}

Playing Sound

This section of the Sound chapter answers FAQs on playing sounds. It covers playing sound from 'snd' resources and from files. It also covers how to manipulate the sound characteristics such as the frequency and playback rate (yes, you too can now sound like Alvin). This section also contains FAQs about how to play a sound asynchronously.

How do I play a 'snd' resource?

The easiest way is to use SndPlay(). Get a handle to a sound resource with GetResource() and pass it to SndPlay(). If you pass nil as the first argument, SndPlay() creates a sound channel for you, however the async parameter is ignored. If you want SndPlay() to play a sound asynchronously then you will have to create a sound channel with SndNewChannel() and pass it to SndPlay().

```c
void PlaySndResource( int sndId )
{
    SndChannelPtr mySndChan;
    Handle mySndHandle;
    OSErr err;

    // Read in 'snd' resource from resource file
    mySndHandle = GetResource ( 'snd ', sndId);
    if ( mySndHandle != nil )
    {
        err = SndPlay ( nil, (SndListResource **)mySndHandle,
                       TRUE);
    }
```
// creating sound channel yourself
mySndChan = nil;
err = SndNewChannel(&mySndChan, sampledSynth, initMono, NULL);
if (!err && mySndHandle != nil )
{
    HLock(mySndHandle);
    err = SndPlay (mySndChan, (SndListResource **)mySndHandle, TRUE);
}

If the 'snd' resource is too large to load into memory with GetResource(), you
can play it directly from disk with SndStartFilePlay().

For more control of your sound playing you can issue the sound commands yourself
with SndDoCommand() and SndDoImmediate(). SndDoCommand() sends a
sound command to a sound channel at the end of the channel's message queue.
SndDoImmediate() executes the command immediately before any commands
already on the queue. To play a 'snd' resource, you issue the sound command
bufferCmd. It is important to note that what is passed to the command is not the
sound handle but a pointer to the sound header contained within the resource. You
can obtain this pointer this by calling GetSoundHeaderOffset() if you have
version 3.0 or later of the Sound Manager. Otherwise you will have to parse the
sound resource manually to get to the sound data. This is detailed in Inside
Macintosh: Sound.

/***********************
void PlayLowlvlSndResource( int sndId )
{
    SndChannelPtr mySndChan;
    SndCommand theCmd;
    Handle mySndHandle;
    OSErr err;
    long myOffset;

    // Read in 'snd' resource from resource file
    mySndHandle = GetResource ('snd ', sndId);
    mySndChan = nil;
    err = SndNewChannel(&mySndChan, sampledSynth, initMono, NULL);
HLock(mySndHandle);
err = GetSoundHeaderOffset((SndListHandle)mySndHandle,
    &myOffset);
if ( err )
{
    HUnlock(mySndHandle);
    return;
}
theCmd.cmd = bufferCmd;
theCmd.param1 = 0;
theCmd.param2 = (long)(*mySndHandle) + myOffset;
SndDoCommand( mySndChan, &theCmd, TRUE);
HUnlock(mySndHandle);

Related FAQ

See also FAQ 27-5, "How do I play a sound continuously from disk, either a large 'snd' resource or an AIFF file?"

Related Topic

See also Chapter 25, "Resources."

How do I play large sounds that won't fit in memory?

See FAQ 27-5, "How do I play a sound continuously from disk, either a large 'snd' resource or an AIFF file?"

How do I play a sound continuously from disk, either a large 'snd' resource or an AIFF file?

There are three functions that handle playing a sound from disk, SndStartPlay(), SndPausePlay() and SndStopPlay(). SndStartPlay() can play either a 'snd' resource or an AIFF sound file from disk. To play a 'snd' resource, pass 0 as the second parameter and the 'snd' resource ID in the third parameter. To play
sound from an AIFF or AIFF-C file, pass the file reference number in the second parameter and 0 in the third parameter. `SndStartPlay()` uses double buffering to play the sound smoothly.

`SndPauseFilePlay()` can be used to pause and restart play from a file. `SndStopFilePlay()` can be used to stop it.

If you want more control over the double buffering used to read sound data from disk you can use `SndPlayDoubleBuffer()`.

The example below reads and plays 'snd' resource with resource number `resId`. The first mouse down pauses the sound play, the next resumes it, and the third stops play.

```c
//*******************
void PlaySndRsrcFromFile( short resId )
{
    SndChannelPtr mySndChan = nil;
    OSErr err;

    err = SndNewChannel( &mySndChan, sampledSynth, 0, nil );
    err = SndStartFilePlay( mySndChan, 0, resId,
                           32768, NULL, NULL, NULL, NULL, TRUE );

    while( !Button() );
    while( Button() );
    SndPauseFilePlay( mySndChan );

    while( !Button() );
    while( Button() );
    SndPauseFilePlay( mySndChan );

    while( !Button() );
    while( Button() );
    SndStopFilePlay( mySndChan, TRUE );

    err = SndDisposeChannel(mySndChan, TRUE );
}
```

**Related FAQ**

See also FAQ 27-3, "How do I play a 'snd' resource?"
How can I begin playing a sound from a point other than the beginning?

You can pause an asynchronously playing sound from a file with SndPauseFilePlay() and then restart it from the paused location with a subsequent call to SndPauseFilePlay(). You can initially play a sound from a file from a specific location by specifying an Audio Selection in the sixth parameter of SndStartFilePlay(). The Audio Selection enables you to specify both the starting point and ending point of the selection to play. The selection points are measured in seconds represented by fixed point data type. You can use the fixed point math Toolbox routines to convert your seconds to fixed point representation. The first field of the Audio Selection structure should be set to 0x0000 to signify that the times are in seconds. There is no other unit of time available as a location option.

```c
//*******************
void PlaySndSelection()
{
    SCStatus chanStatus;
    SndChannelPtr mySndChan = nil;
    OSErr err;
    AudioSelection ASel;

    err = SndNewChannel( &mySndChan, sampledSynth, 0, nil );
    ASel.unitType = 0x0000;
    ASel.selStart = X2Fix(1.5);
    ASel.selEnd = X2Fix(7.25);

    err = SndStartFilePlay( mySndChan, 0, 200, 65536, NULL,
                            &ASel, NULL, TRUE );
}
```

How do I play more than one sound at a time?

The Sound Manager allows concurrent play of multiple sounds. To do this you must play a sound asynchronously. When control has returned to your program you can start another sound. This is true for both sounds in memory, such as a loaded 'snd' resource, and sounds played from disk.

The example below plays five sounds concurrently from disk.
void MultChannelPlay( void )
{
    SndChannelPtr channelArray[NUMCHANNELS];
    OSErr err;
    int i;
    long finalTick;

    for ( i = 0; i < NUMCHANNELS; i++)
    {
        channelArray[i] = nil;
        err = SndNewChannel( &channelArray[i], sampledSynth, 0, nil );

        err = SndStartFilePlay( channelArray[i], 0, 200, 32768, NULL, NULL, NULL, TRUE );

        Delay( 10, &finalTick );
    }
}

How many sound channels can play sounds concurrently?

The number of sound channels that can play sounds concurrently is limited only by memory and the processing power of the machine. The Sound Manager will allocate and play as many sound channels as memory allows. A large number of sound channels playing a sound at once will affect the quality of sound playback and will have an impact on other processing. More powerful machines and machines with more memory will be able to produce more simultaneously playing sounds.

How do I stop a sound that is playing?

If you want the ability to stop a sound, you have to play the sound asynchronously. If the sound is being played from a file with SndStartFilePlay(), you can pause or stop the sound with SndPauseFilePlay() and SndStopFilePlay(). Sounds not initiated by SndStartFilePlay() can be stopped by sending a
quietCmd command to the channel. To clear the channel of all sound commands that are still in the command queue, send the flushCmd command to the channel. These commands must be issued by SndDoImmediate().

The example below plays a sound for 2 seconds and then the sound commands quietCmd and flushCmd to stop playing the sound.

```c
//*****************
void StopPlaying( void )
{
    SndChannelPtr mySndChan;
    SndCommand theCmd;
    Handle mySndHandle;
    OSerr err;
    long myOffset;
    long finalTicks;

    // Read in 'snd' resource from resource file
    mySndHandle = GetResource ('snd ', 200);
    mySndChan = nil;
    err = SndNewChannel(&mySndChan, sampledSynth, initMono, NULL);
    err = GetSoundHeaderOffset((SndlistHandle)mySndHandle, &myOffset);
    if ( err )
        return;

    HLock(mySndHandle);
    theCmd.cmd = bufferCmd;
    theCmd.param1 = 0;
    theCmd.param2 = (long)(*mySndHandle) + myOffset;
    SndDoCommand( mySndChan, &theCmd, TRUE);
    Delay(120, &finalTicks);

    theCmd.cmd = quietCmd;
    theCmd.param1 = 0;
    theCmd.param2 = 0;
    SndDoImmediate( mySndChan, &theCmd);

    theCmd.cmd = flushCmd;
```
How can I change the playback rate, to slow down or speed up the sound playing?

You can get and set the current playback rate of a sound by issuing the sound commands `getRateCmd` and `rateCmd` respectively. The playback rate accessed by these commands is relative to 22 Khz and represented in fixed point type. To play back a sound at 11 KHz you would send a value of .5, 0x00008000, to specify half the playback rate of 22 Khz. To play back a sound at 44 Khz you would send a value of 2, 0x00020000.

The Toolbox routine `UnsignedFixMulDiv()` can be used to calculate the sample rates. To determine the actual sample rate given a playback rate relative to 22 Khz:

\[ \text{actualRate} = \text{UnsignedFixMulDiv}(\text{rate22kHz}, \text{currentRelRate}, 0x00010000); \]

To determine the value of the relative multiplier to produce the new rate:

\[ \text{newRelRate} = \text{UnsignedFixMulDiv}(\text{newActualRate}, 0x00010000, \text{rate22kHz}); \]

The snippet below gets the current rate of a playing sound and then sets the sound to play at the rate of 11khz.

```c
void ChangeRate()
{
    SndChannelPtr mySndChan;
    SndCommand theCmd;
    UnsignedFixed newRate, currentRelRate, actualRate;

    // start an asynchronous sound playing :
    // get the current actual rate
    theCmd.cmd = getRateCmd;
    theCmd.param1 = 0;
```
How do I install a sound and play different notes with it?

You can install a sampled sound as a voice by calling the Sound command soundCmd. You can then play the sampled sound back at different notes with the Sound commands freqCmd and freqDurationCmd. The notes played are specified by MIDI note values. You will find a chart on page 2-43 of Inside Macintosh: Sound that maps frequencies to the MIDI note values.

Why doesn’t my sound play asynchronously?

There are a few rules that you must follow to play sound asynchronously:

- When you call SndPlay() or SndStartFilePlay() make sure the async parameter is true.

- You must pass a sound channel created by SndNewChannel() to SndPlay() and SndStartFilePlay(). If you pass nil, thereby allowing the Sound Manager to create a sound channel for you, sound will only play synchronously.

- If you call SndStopFilePlay() with the async parameter set to false, the routine doesn’t execute until the sound completes playing.

- If you call SndDisposeChannel() with the quietNow parameter set false, the routine doesn’t execute until the sound completes playing.
How do I set up callback and completion routines for playing asynchronous sounds?

A sound completion routine is a routine that is called by the Sound Manager when a sound is finished being played by the SndStartFilePlay() command. The completion routine is installed by passing a pointer to it into the seventh parameter of SndStartFilePlay().

A callback routine is a routine that is executed by the Sound Manager whenever it receives a callbackCmd sound command. The callback routine is installed by passing a pointer to it into the fourth parameter of SndNewChannel().

Related FAQ

See also FAQ 27-14, "How can I tell when a sound playing asynchronously has completed playing?"

How can I tell when a sound playing asynchronously has completed playing?

There are several ways to determine whether an asynchronous sound has completed playing:

- by continuously polling the status of the sound channel with SndChannelStatus()
- by installing a callback function into the sound channel and sending a callbackCmd causing it to execute after a bufferCmd
- by passing a completion routine to SndStartFilePlay() to execute upon completion of the sound

To use SndChannelStatus(), start playing a sound asynchronously and then periodically call the function. If the channel is still playing a sound, the scChannelBusy field of the third parameter, an SCStatus structure, will be true. If it is not, it will be false. You can make this call in your event loop. The example below simply loops until the sound stops playing.

```c
/****************************
void CheckByStatus()
{
```
SndChannelPtr     mySndChan;
handle            mySndHandle;
OSErr             err;
SCStatus          chanStatus;

// Read in 'snd' resource from resource file
mySndHandle = GetResource ('snd ', 200);
mySndChan = nil;
err = SndNewChannel(&mySndChan, sampledSynth, initMono, NULL);
HLock(mySndHandle);
if ( !err && mySndHandle != nil )
    err = SndPlay (mySndChan, 
        (SndListResource **)mySndHandle, TRUE);

    do
    {
        err = SndChannelStatus( mySndChan, 
            sizeof(chanStatus), &chanStatus);
    }
while ( (err == noErr) && (chanStatus.scChannelBusy) );

HUnlock(mySndHandle);
err = SndDisposeChannel( mySndChan, TRUE);

To avoid continuously polling the status of the channel with SndChannelStatus(),
you can call SndStartFilePlay() and have it use a completion routine. The
completion routine will be executed after the sound has completed playing. It can
be used to set a global variable that indicates when the sound is done. The seventh
parameter of SndStartFilePlay() is a pointer to the completion routine. The
example below uses universal headers and defines the procedure as a
NewFilePlayCompletionProc.

The completion procedure is called at interrupt time. This means that it cannot
access its application’s global variables unless its A5 world is set up correctly. To
do this, set the userInfo field of the sound channel to the pointer to the current
A5 world. When the completion routine is executed a pointer to the sound channel
is passed to it as its one parameter. It can then retrieve the application’s A5 and
restore it to access global variables. Only programs written for Power Macintosh
computers are exempt from this complication.

It is important to note that the completion routine is different from the callback
procedure assigned to a channel by SndNewChannel(). The callback procedure
is used in the next example.
Boolean gSoundComplete;

void CheckByCompletion()
{
    SndChannelPtr mySndChan = nil;
    OSErr err;
    FilePlayCompletionUPP myProc;
    myProc = NewFilePlayCompletionProc(MyCompletionRoutine);
    err = SndNewChannel(&mySndChan, sampledSynth, 0, nil);
    gSoundComplete = FALSE;
    mySndChan->userinfo = SetCurrentA5();
    err = SndStartFilePlay(mySndChan, 0, 200, 65536, NULL, NULL, myProc, TRUE);

    while (!gSoundComplete)
    {
        // do some action
    }
}

Pascal Void MyCompletionRoutine(SndChannelPtr mySndChan)
{
    long myA5;
    myA5 = SetA5(mySndChan->userinfo);
    gSoundComplete = TRUE;
    myA5 = SetA5(myA5);
}

The third way of determining when a sound is completed is to install a callback procedure in the sound channel and send the channel a callbackCmd sound command. The sound channel will put the callbackCmd command in its command queue. When the command that is playing the sound completes, the next command in the queue, the callbackCmd, will be processed, causing the installed callback procedure of the sound channel to be executed. As with completion routines, callback procedures are executed during interrupt time and must restore the application's A5 world to access globals. This is done the same way as described...
above in the previous example of completion routines.

To install a callback procedure, pass a pointer to the third parameter of the SndNewChannel() call. Note that the callback procedure takes two parameters while the completion routine takes only one.

```c
//************************
void CheckByCallBack( )
{
    SndChannelPtr mySndChan = nil;
    SndCommand theCmd;
    OSErr err;

    err = SndNewChannel( &mySndChan, sampledSynth, 0,
                          NewSndCallBackProc(MyCallBackProc) );

    gSoundComplete = FALSE;
    mySndChan->userInfo = SetCurrentA5();
    err = SndStartFilePlay( mySndChan, 0, 200, 65536, NULL,
                            NULL, NULL, TRUE );

    theCmd.cmd = callBackCmd;
    theCmd.param1 = 0;
    theCmd.param2 = 0;
    SndDoCommand( mySndChan, &theCmd, FALSE );

    while ( !gSoundComplete )
    {
        // do some action
    }
}

//************************
pascal void MyCallBackProc(SndChannelPtr mySndChan,
                            SndCommand *pCmd )
{
    long myA5;

    myA5 = SetA5(mySndChan->userInfo);
    gSoundComplete = TRUE;
    myA5 = SetA5(myA5);
}
```
Related FAQ

See also FAQ 27-29, "How do I unlock and release a sound resource and dispose of a sound channel when I play a sound asynchronously?"

**Why do I get static when I play a sound?**

A commonly made mistake is the releasing of a sound handle that is being played asynchronously immediately after the function that initiated the asynchronous play is executed. This means that the Sound Manager is reading data from memory that could be overwritten. Do not release the sound handle until after the sound has completed playing. You can detect when a sound has completed by calling `SndChannelStatus()` or by being notified by a callback routine installed in your sound channel or by a completion routine passed to `SndStartFilePlay()`.

It is also important to lock your sound handle before passing it to a Sound Manager function.

**How can I synchronize the playing of sound by sound channels?**

You can synchronize the sound playing of multiple sound channels with the `syncCmd` sound command. `syncCmd` uses both parameters of the sound command structure; parameter 1 contains a count and parameter 2 contains a group identifier. All of the sound channels that are to be synchronized are assigned the same group identifier. When `syncCmd` is sent to a sound channel, that channel is assigned the group number in parameter 2 and its count gets set to the value in parameter 1. When the sound channel is assigned a count > 0 it stops processing commands. Each time `syncCmd` is sent, all sound channels assigned to the group number in parameter 2 have their count value decremented. When a channel’s count value gets decremented or set to 0 it resumes processing commands. The `syncCmd` must be issued with `SndDoImmediate()`.

The sample below synchronizes three sound channels so they start playing sound at the same time. It first creates three sound channels, gets three ‘snd’ resources and gets the offset to the sound data. Next it issues the `syncCmd` to the three channels. Notice that the first channel is sent a count value of 4. The next channel is sent a count value of 3. The third channel is sent a value of 2. Since previously existing channels get their count decremented with every `syncCmd` all three channels now have the same count value. Then the sound channels get sent a `bufferCmd`
command which is stored in their queues. Sending another syncCmd with a count value of 1 causes all three channels to have their count decremented to 0, and they all start playing sound.

Sound channel synchronization can also be used to stop channels simultaneously or to perform any type of sound command in a synchronized manner.

```c
//********************
#define kldentifier 1
void SynchronizeSound()
{
    SndChannelPtr channelArray[3] = {nil, nil, nil};
    Handle mysndHandleArray[3];
    long myOffsetArray[3];
    int soundIdArray[3] = {128, 129, 200};
    SndCommand theCmd;
    OSErr err;
    int i;

    // allocate three sound channels
    for ( i = 0; i < 3; i++)
    {
        err = SndNewChannel(&channelArray[i], sampledSynth,
                             initMono, NULL);
    }

    // Get three sound handles from the three sound ids and
    // get the offsets to each
    for ( i = 0; i < 3; i++)
    {
        mysndHandleArray[i] = GetResource ('snd ',
                                             soundIdArray[i]);
        HLock(mysndHandleArray[i]);
        err = GetSoundHeaderOffset(
            (SndListHandle)mysndHandleArray[i],
            &myOffsetArray[i]);
        if ( err )
            return;
    }

    // send a syncCmd to the three channels. The first channel
    // gets as the sync count
    // the second get 3 and the third gets 2
```
for (i = 0; i < 3; i++)
{
    theCmd.cmd = syncCmd;
    theCmd.param1 = 4 - i;
    theCmd.param2 = kIdentifier;
    SndDoImmediate( channelArray[i], &theCmd);
}

// send the three channels commands that you want executed
for (i = 0; i < 3; i++)
{
    theCmd.cmd = bufferCmd;
    theCmd.param1 = 0;
    theCmd.param2 = (long)(*(mySndHandleArray[i])) +
        myOffsetArray[i];
    SndDoCommand( channelArray[i], &theCmd, TRUE);
}
// send a sync command to decrement the count on all
// channels and start them playing
theCmd.cmd = syncCmd;
theCmd.param1 = l;
theCmd.param2 = kIdentifier;
SndDoImmediate( channelArray[l], &theCmd);
}

Adjusting the Volume

This section of the Sound chapter answers FAQs about controlling the volume at
which a sound is played back. It also explains how to control the volume setting of
your computer, and discusses autonomous control of left and right volumes.

How can I get and set the volume of my computer?

You can adjust the default volume of your computer so that it affects all sound from
all sources. Two functions are available to manipulate the default volume,
GetDefaultOutputVolume() and SetDefaultOutputVolume().

The sound volume is represented by a single long value. The 16 high bits make up
the right channel volume; the 16 low bits make up the left channel volume. 0 for
each channel (0x00000000) is silence and 256 for each channel (0x08000800) is
full or natural volume. It is possible to overdrive the volume. A value of 512 for each channel would double the volume. This will often produce a distorted sound.

The macros below are useful for getting and setting the individual volumes of the two channels. `COMPUTER_VOLUME(LVOL, RVOL)` takes the volume settings for both channels and creates the single long value to pass to `SetDefaultOutputVolume()`. `GET_LEFT_VOLUME(lVolume)` and `GET_RIGHT_VOLUME(lVolume)` extract the individual channel volumes for the left and right channels respectively.

```c
#define COMPUTER_VOLUME(LVOL, RVOL)  ((RVOL << 16L | LVOL )
#define GET_LEFT_VOLUME(lVolume)      (lVolume & 0x0000FFFF)
#define GET_RIGHT_VOLUME(lVolume)     ((lVolume & 0xFFFF0000) >> 16L)
```

The example below first sets the volume on both channels to full volume, then half volume, and then silence. The first for loop increases the left channel volume while leaving the right channel silent. The second for loop brings the right channel volume up to the left channel volume level. The individual controlling of the left and right volumes enables a panning effect.

```c
//***************
void AdjustComputerVolume()
{
    long lVolume, leftVolume, rightVolume;
    int i;
    long finalTicks;

    leftVolume = 256;
    rightVolume = 256;
    lVolume = COMPUTER_VOLUME(leftVolume, rightVolume);
    SetDefaultOutputVolume(lVolume);
    GetDefaultOutputVolume(&lVolume);
    Delay(120, &finalTicks);

    leftVolume = GET_LEFT_VOLUME(lVolume);
    rightVolume = GET_RIGHT_VOLUME(lVolume);
    leftVolume = leftVolume / 2;
    rightVolume = rightVolume / 2;
    lVolume = COMPUTER_VOLUME(leftVolume, rightVolume);
    SetDefaultOutputVolume(lVolume);
    GetDefaultOutputVolume(&lVolume);
}```
Delay(120, &finalTicks);

leftVolume = 0;
rightVolume = 0;
lVolume = COMPUTER_VOLUME(leftVolume, rightVolume);
SetDefaultOutputVolume(lVolume);
GetDefaultOutputVolume(&lVolume);

Delay(120, &finalTicks);

leftVolume = GET_LEFT_VOLUME(lVolume);
rightVolume = GET_RIGHT_VOLUME(lVolume);
for ( i = 0; i < 10; i++ )
{
    rightVolume = rightVolume + 20;
lVolume = COMPUTER_VOLUME(leftVolume, rightVolume);
    SetDefaultOutputVolume(lVolume);
    GetDefaultOutputVolume(&lVolume);
    Delay(30, &finalTicks);
}

leftVolume = GET_LEFT_VOLUME(lVolume);
rightVolume = GET_RIGHT_VOLUME(lVolume);
for ( i = 0; i < 10; i++ )
{
    leftVolume = leftVolume + 20;
lVolume = COMPUTER_VOLUME(leftVolume, rightVolume);
    SetDefaultOutputVolume(lVolume);
    GetDefaultOutputVolume(&lVolume);
    Delay(30, &finalTicks);
}

GetDefaultVolume() and SetDefaultVolume() are available only with
versions 3.0 and above of the Sound Manager. SetDefaultVolume() needs a
corresponding GetDefaultVolume() call after it in order for its changes to take
affect.

Related FAQs

☐ See also FAQ 27-18, “How can I adjust the left and right volumes, like a balance
control (panning)?”

☐ See also FAQ 27-19, “How do I adjust the volume at which a channel plays a sound?”
**How can I adjust the left and right volumes, like a balance control (panning)?**

Both the default volume of the computer, which affects all output sound, and the volume of each channel, which affects only the sound from that channel, can be individually adjusted for the left and right channels. The volumes are represented as a single long with the high 16 bits used for the right channel and the low 16 bits used for the left channel. A value of 0 in each of the 16 bits is silence; 256 is full volume. The macro below can be used to get a single volume value representing both channels.

```c
#define COMPUTER_VOLUME(LVOL, RVOL) (RVOL << 16L | LVOL )
```

A single value can be used to represent balance. You can think of the range of the balance control as -128 to 128. -128 is full left volume and 0 right volume. 128 is 0 left volume and full right volume. The volume setting for a corresponding balance becomes

```c
balanceVolume = COMPUTER_VOLUME(128 - balance, 128 + balance)
```

**Related FAQs**

- See also FAQ 27-17, “How can I get and set the volume of my computer?”
- See also FAQ 27-19, “How do I adjust the volume at which a channel plays a sound?”

**How do I adjust the volume at which a channel plays a sound?**

Besides changing the default volume of the computer, which affects the volume of all sound output, you can adjust the output volume of an individual channel so that only the sound produced by that channel is affected. There are two sound commands that affect a channel's volume, `ampCmd`, which controls the amplitude or loudness of the sound, and `volumeCmd` which can be used to control the volume of the left and right channels individually. There also exist sound commands to get the current amplitude and the current volume of the channel, `getAmpCmd` and `getVolumeCmd`.

The sound volume is represented by a single long. The 16 high bits make up the right channel volume, the low 16 bits make up the left channel volume. 0 for each channel
(0x00000000) is silence and 256 for each channel (0x08000800) is full or natural volume. It is possible to overdrive the volume. A value of 512 for each channel would double the volume.

The macros below are useful for getting and setting the individual volumes of the two channels. `COMPUTER_VOLUME(LVOL, RVOL)` takes the volume settings for both channels and creates the single long value to pass to `SetDefaultOutputVolume()`. `GET_LEFT_VOLUME(lVolume)` and `GET_RIGHT_VOLUME(lVolume)` extract the individual channel volumes for the left and right channels respectively.

```c
#define COMPUTER_VOLUME(LVOL, RVOL) (RVOL << 16L | LVOL )
#define GET_LEFT_VOLUME(lVolume)   (lVolume & 0x0000FFFF)
#define GET_RIGHT_VOLUME(lVolume)  
    ((lVolume & 0xFFFF0000) >> 16L)
```

The example below is a snippet that sets the loudness of both channels to 0 and gradually increases it. Next it does the same with the left channel volume, and then the right channel volume.

```c
/**
 * PlayWithVolume
 */
void PlayWithVolume()
{
    SndChannelPtr mySndChan;
    SndCommand theCmd;
    long lVolume, finalTicks;
    int i, loudness, volume, leftVolume;

    // start asynchronous sound playing :

    loudness = 0;
    for ( i = 0; i < 10; i++ )
    {
        loudness += 20;
        theCmd.cmd = ampCmd;
        theCmd.param1 = loudness;
        theCmd.param2 = 0;
        SndDoImmediate( mySndChan, &theCmd);
        Delay(30, &finalTicks);
    }

    // silence the channel
```
theCmd.cmd = volumeCmd;
theCmd.param1 = 0;
theCmd.param2 = 0L;
SndDoImmediate( mySndChan, &theCmd);

// increase left channel volume
volume = 0;
for ( i = 0; i < 10; i++ )
{
    volume = volume + 20;
    lVolume = COMPUTER_VOLUME(volume, 0);

    theCmd.cmd = volumeCmd;
    theCmd.param1 = 0;
    theCmd.param2 = lVolume;
    SndDoImmediate( mySndChan, &theCmd);

    Delay(30, &finalTicks);
}

// get current volume
theCmd.cmd = getVolumeCmd;
theCmd.param1 = 0;
theCmd.param2 = (long)&lVolume;
SndDoImmediate( mySndChan, &theCmd);
leftVolume = GET_LEFT_VOLUME(lVolume);

// increase right channel volume
volume = 0;
for ( i = 0; i < 10; i++ )
{
    volume = volume + 20;
    lVolume = COMPUTER_VOLUME(leftVolume, volume);
    theCmd.cmd = volumeCmd;
    theCmd.param1 = 0;
    theCmd.param2 = lVolume;
    SndDoImmediate( mySndChan, &theCmd);

    Delay(30, &finalTicks);
}
CHAPTER 27: Sound

Related FAQs

- See also FAQ 27-17, "How can I get and set the volume of my computer?"
- See also FAQ 27-18, "How can I adjust the left and right volumes, like a balance control (panning)?"

Why does my volume go down every time another channel plays a sound?

If a single sound is playing, the Sound Manager will play it at full volume. If there are two sounds playing simultaneously, each will be played at one half of the volume at which they would be played if they were being played alone. If three sounds are being played simultaneously each will be played at one third the volume. And so on.

Recording Sound

This section of the Sound chapter answers FAQs on recording sounds. It describes how to record sound and create both a ‘snd‘ resource and an AIFF file. It provides examples of doing this both synchronously and asynchronously. You will even learn how to record a sound without displaying the standard sound recording dialog.

How do I record sound in my application?

The easiest way is to call either SndRecord() or SndRecordToFile(). Both of these functions put up the standard sound recording dialog (see Figure 27-1) and record synchronously.
SndRecord() records to memory and is limited by the amount of available memory. The recorded sound is in 'snd' format. SndRecordToFile() records the data to a file and is limited only by the amount of available disk space. In both functions, by specifying the quality level of the recorded sound you determine the compression used. The resulting file is in AIF or AIF-C format.

The example below calls SndRecord() to record a file and plays it back.

```c
//****************************
void DoSimpleRecord()
{
    Handle hSound = nil;
    Point dlogPt = (100,100);
    OSErr err;

    SndRecord(nil, dlogPt, siBestQuality, (SndListResource **)&hSound);
    if (hSound != nil)
        err = SndPlay(nil, (SndListResource **)hSound, TRUE);
}
```

How do I record a sound without the standard recording dialog?

To record sound without bringing up the standard recording dialog that SndRecord() and SndRecordToFile() use, you use low level sound input routines which enable you to read data from a sound input device, get information about the sound input device, and alter the settings of the device. SPBRecordToFile() can be used to read data asynchronously from the input device into a file. It does not produce an AIF format file; it simply dumps the data without a header. SPBRecord() can be used to dump the data into a preallocated buffer. It does not create a 'snd' resource; it dumps the data into a buffer without a header.

To access a sound input device, you first open it with SPBOpenDevice(). You can request the default input device by passing nil in the deviceName parameter.
Once the device is open and you have a reference number for it, you can use SPBGetDeviceInfo() to read the input device’s settings. This provides the information necessary for recording sound:

- the number of channels (either mono or stereo)
- the sample rate (represented as a Fixed type value)
- the sample size (8 bit or 16 bit)
- the sound compression type

When we have this information we can then create a header for a 'snd' resource with SetupSndHeader(). To record sound coming in from the device to a buffer, call SPBRecord(). SPBRecord() takes a sound input parameter block. The size of buffer to contain the recorded data is passed into the bufferLength field. The count field gets set to the number of bytes to record. These two values are usually the same and, in the case of our example below, contain the amount of data allocated in the block of memory that will become the 'snd' resource minus the space required for the 'snd' resource header. You can also specify the time in milliseconds to record in the milliseconds field. The amount of data recorded is taken from the larger of the count and milliseconds fields. Once count data has been recorded or the number of milliseconds has elapsed then recording stops. After recording is completed the device is closed by SPBClose() and the 'snd' resource is completed by calling SetupSndHeader() and specifying the size of the sound data.

The example below allocates a handle and then calls RecNoDlog() to create a record sound (from the default input device) and a 'snd' resource. Then it calls SndPlay() to play the resource.

```c
//****************************
#define RECORDED_SND_SIZE 65536

void DoRecordNoDlog()
{
    Handle mySndH;

    mySndH = NewHandle( RECORDED_SND_SIZE );
    RecNoDlog( mySndH );

    HLock( mySndH );
    SndPlay( nil, (SndListResource **)mySndH, FALSE);
HUnlock( mySndH );

//************************
void RecNoDlog( Handle mySndH )
{
    SPB      mySPB;
    OSTYPE    compType;
    OSErr err;
    Fixed sampRate;
    long  myBuffSize, devRefNum;
    short numChans, sampSize, sndHdrLength;

err = SPBOpenDevice( nil,siWritePermission, &devRefNum);
if ( err != noErr )
    return;

SPBGetDeviceInfo(devRefNum, siNumberChannels, &numChans);
SPBGetDeviceInfo(devRefNum, siSampleRate, &sampRate);
SPBGetDeviceInfo(devRefNum, siSampleSize, &sampSize);
SPBGetDeviceInfo(devRefNum, siCompressionType, &compType);

err = SetupSndHeader((SndlistHandle)mySndH, numChans,
                    sampRate,  
                    sampSize, compType, kMiddleC, 0,
                    &sndHdrLength);
if ( err != noErr )
{
    SPBCloseDevice(devRefNum);
    return;
}

myBuffSize = GetHandleSize(mySndH) - sndHdrLength;
HLockHi(mySndH);

mySPB.inRefNum = devRefNum;
mySPB.count = myBuffSize;
mySPB.milliseconds = 0;
mySPB.bufferLength = myBuffSize;
mySPB.bufferPtr = (Ptr)(*mySndH + sndHdrLength);
mySPB.completionRoutine = nil;
mySPB.interruptRoutine = nil;
mySPB.userLong = 0;
The function `SPBRecordToFile()` can be used in place of `SPBRecord()` to record the information directly to file. It does not create an AIFF format file but simply dumps the recorded data into a file.

**How do I create a 'snd' resource from sampled data?**

You can create a 'snd' resource from sampled data by first setting up the header of the resource with `SetupSndHeader()` and then copy the sound data from the sampled sound source into a block of memory that will become the 'snd' resource.

First allocate a handle of 200 bytes, large enough to hold the header information of a 'snd' resource. Then call `SetupSndHeader()` with the desired rate, number of channels, and bit size but with 0 in the `numBytes` field. This will pass back the actual size of the header for this 'snd' resource. Then reallocate the handle for the new 'snd' resource to be large enough to hold the header and the size of the sound data. Call `BlockMove()` to move the sound data to your handle's memory block after the header. Call `SetupSndHeader()` again with the new size.

This sample creates a new 'snd' resource from an existing one and then plays the new one with `SndPlay()`.
//********************
void SndToSndRsrc()
{
    SndCommand theCmd;
    Handle mySndHandle, hNewSnd;
   OSErr err;
    long myOffset, length;

    // Read in 'snd' resource from resource file
    mySndHandle = GetResource ('snd', 200L);

    HLock(mySndHandle);
    length = GetHandleSize(mySndHandle);
    hNewSnd = CreateSndRsrc( (char *)(*mySndHandle), length,
                             rate22khz, 8, 1, kMiddleC );
    err = SndPlay (nil, (SndListResource **)hNewSnd, TRUE);
}

//********************
Handle CreateSndRsrc( Ptr pSample,
                        long dataLength,
                        UnsignedFixed sampleRate,
                        short bitSize,
                        short numChannels,
                        short int baseFreq)
{
    OSErr err;
    Handle theSoundHeader;
    Handle theSound = nil;
    short headerLength;
    // the sound header handle must be large enough to setup
    // the header information
    theSoundHeader = NewHandleClear(200L);
    if (theSoundHeader == NULL)
        return( NULL);

    err = SetupSndHeader((SndListHandle)theSoundHeader,
                          numChannels,
                          sampleRate, bitSize, 'NONE',
                          baseFreq, 0, &headerLength);

if (err != noErr) {
    DisposeHandle(theSoundHeader);
    return( NULL);
}

// copy the header information of headerLength size to the sound resource we are creating
theSound = NewHandle(headerLength + dataLength);
if (theSound == nil)
    { 
    DisposeHandle(theSoundHeader);
    return( NULL);
    }

HLock(theSound);
BlockMove(*theSoundHeader, *theSound, headerLength);
BlockMove(pSample, ((Byte *)*theSound) + headerLength, dataLength);
HUnlock(theSound);
DisposeHandle(theSoundHeader);

// create sound resource specifying the size of the data
err = SetupSndHeader((SndListHandle)theSound, numChannels,
    sampleRate,
    bitSize, 'NONE', baseFreq,
    dataLength, &headerLength);
if (err)
    { 
    DisposeHandle(theSound);
    return( NULL);
    }
return theSound;

How do I create a AIFF file from sampled data?

You can create a AIFF file from sampled data by creating and opening the data fork of the AIFF file, setting up the header of the resource with SetupAIFFHeader(), and then writing the sound data from the sampled sound source into the file. An important point to note: AIFF files expect the volume levels to be in the range
of -128 to 127 for 8 bit data, and -32768 to 32767 for 16 bit data with a value of 0 representing silence. The standard Macintosh 8 bit 'snd' resource has volume levels to be in the range of 0 to 256 with 128 representing silence. Before writing data to an AIFF file, the sampled data must be phased back by 128 for every 8 bits.

After you have created and opened a file, call SetupAIFFHeader() with the desired rate, number of channels, and bit size but with 0 in the numBytes field. This will move the file mark to the position in the file where the audio data should be written. Adjust the volume of the sample to follow the range of -128 to 127. Next write the audio data to the file. Finally call SetupAIFFHeader() with the proper number of bytes in the header.

The sample below creates an AIFF file from a 'snd' resource.

```c
void SndToAIFF()
{
    Handle mySndHandle, hNewSnd;
    SndCommand theCmd;
    OSErr err;
    FSSpec mySpec;
    long myOffset, length;

    mySpec.vRefNum = 0;
    mySpec.parID = 0;
    strcpy( mySpec.name, "\ptestaiiff");

    // Read in 'snd' resource from resource file
    mySndHandle = GetResource ('snd ', 200);

    HLock(mySndHandle);
    length = GetHandleSize(mySndHandle);

    Adjustvolume(*mySndHandle, length);

    CreateAIFFFFile( &mySpec, (char *)(*mySndHandle),
        length, 
        rate22khz, 8, 1);
}
```
//***************
// Adjust the volume of the sample to
// follow the range of -128 to 127
// by reducing every 8 bits by 0x80
//***************
static void Adjustvolume ( register char *p,
                         register long size)
{
    register char *endPtr;
    endPtr = p + size;
    while (p < endPtr)
    {
        *p -= 0x80;
        p++;
    }
}

//***************
OSErr CreateAIFFFile(  FSSpec *aiffFilePtr,
                Ptr pSample,
                long dataLength,
                UnsignedFixed sampleRate,
                short bitSize,
                short numChannels )
{
    OSErr err;
    long length;
    short fileRef;

    if ( (bitSize != 8) && (bitSize != 16) )
        return (badFormat);

    if ( (numChannels != 1) && (numChannels != 2) )
        return (badFormat);

    err = FSpCreate(aiffFilePtr, 'test', 'AIFF', 0);
    if (err == noErr)
    {
        err = FSpOpenDF(aiffFilePtr, fsRdWrPerm, &fileRef);
        if (err == noErr)
            //***************
// SetupAIFFHeader will move file mark to
// correct position
// to write data in file
SetFPos(fileRef, fsFromStart, OL);
err = SetupAIFFHeader(fileRef, numChannels,
sampleRate, bitSize, 'NONE',
OL, dataLength);
if (err == noErr)
{
    length = dataLength * numChannels *
            (bitSize / 8);
    err = FSWrite(fileRef, &length,
pSample);
    if (err == noErr)
    {
        SetFPos(fileRef, fsFromStart, OL);
        err = SetupAIFFHeader(fileRef,
                            numChannels, sampleRate,
                            bitSize, 'NONE',
                            length, dataLength);
    }
    FSClose(fileRef);
}
}
return err;

Related Topic

See also Chapter 8, “Files: File Manager.”

How can I determine how long an asynchronous sound has been playing or still has remaining until completion?

SndChannelStatus() passes back a pointer to an SCStatus structure in its third parameter that provides information on sound played from disk. The scEndTime field tells you the ending time in seconds. The scCurrentTime tells you the current playing time in seconds. Both of these fields are fixed data types.
Related FAQ

See also FAQ 27-14, "How can I tell when a sound playing asynchronously has completed playing?"

How can I record sound asynchronously?

You can record asynchronously by calling the low-level Sound Input Manager routines which enable you to read data from a sound input device, get information about the sound input device, and alter the settings of the device. SPBRecordToFile() can be used to read data asynchronously from the input device into a file. It does not produce an AIFF format file; it simply dumps the raw sound data without a header. SPBRecord() can be used to dump the data into a preallocated buffer. It does not create a 'snd' resource; it also simply dumps the raw sound data into a buffer without a header.

To access a sound input device, you first open it with SPBOpenDevice(). You can request the default input device by passing nil in the devicename parameter. Once the device is open and you have a reference number for it, you can use SPBGetDeviceInfo() to read the input device's settings if you want to create a 'snd' resource or create a header for an AIFF file. To record sound coming in from the device to a file, call SPBRecordToFile(). SPBRecordToFile() takes the reference number of the file it is to write to and a sound input parameter block. The count field in the parameter block gets set to the number of bytes to record. You can also specify the time in milliseconds to record in the milliseconds field. The amount of data recorded is determined by the larger of the two values: the amount of data specified in the count field and the amount of data that would be recorded in the time specified in the milliseconds field. Once this amount of data has been recorded, the recording stops. You can stop recording before all of the data has been recorded by calling SPBStopRecording(). After recording is completed, close the file that is being written to and close the device by SPBClose().

You can assign a completion routine to be executed upon completion of the recording, either by a call to SPBStopRecording() or when all of the specified data is recorded by putting a pointer to it in the completionRoutine field. The completion routine is called at interrupt time, therefore, on 68K Macs you need to set up the A5 world to access globals. The completion routine can set a global variable marking completion of the recording.

You can pause asynchronous by calling SPBPauseRecording().
The example below records data into the file RecordSound for ten seconds or until the mouse button is pressed. If the mouse button is pressed before ten seconds have elapsed the recording is manually halted by calling SPBStopRecording(). If the user does not press the mouse button within the ten second limit then the completion routine is called setting the global flag, causing the loop to be exited.

```c
******
Boolean gRecordComplete;

void DoAsyncRecordNoDlog( void )
{
    SPB mySPB;
    OSErr err;
    long devRefNum;
    short fRefNum;
    FSSpec mySpec;

    err = SPBOpenDevice( nil, siWritePermission, &devRefNum);
    if ( err != noErr )
        return;

    mySPB.inRefNum = devRefNum;
    mySPB.count = 0;
    mySPB.milliseconds = 10000;
    mySPB.completionRoutine = NewSICompletionProc(MyRecordCompletionProc);
    mySPB.completionRoutine = nil;
    mySPB.interruptRoutine = nil;
    mySPB.error = noErr;
    mySPB.unused1 = 0;
    // store A5 in userLong for access
    // by the completion routine
    mySPB.userLong = SetCurrentA5();

    gRecordComplete = FALSE;

    mySpec.vRefNum = 0;
    mySpec.parID = 0;
    pStrcpy( mySpec.name, "\pRecordSound" );
    FSpCreate( &mySpec, 'TEST', 'AIFF', smSystemScript );
    err = FSpOpenDF(&mySpec, fsRdWrPerm, &fRefNum);
```
err = SPBRecordToFile(fRefNum, &mySPB, TRUE);

// loop until user presses the mouse button or until the
// time limit has elapsed and gRecordComplete is set by
// the completion routine
while ( !gRecordComplete )
{
    // do stuff while async recording
    if ( Button() )
    {
        SPBStopRecording(devRefNum);
        gRecordComplete = TRUE;
    }
}
FSClose( fRefNum);

if ( err != noErr )
{
    SPBCloseDevice(devRefNum);
    return;
}

err = SPBCloseDevice(devRefNum);

To create either a 'snd' resource or an AIFF file you need information about the sound
to create a header. Once again, the information necessary to record sound is:

✦ the number of channels (either mono or stereo)
✦ the sample rate (represented as a Fixed type value)
✦ the sample size (8 bit or 16 bit)
✦ the sound compression type

This information can be retrieved by calling SPBGetDeviceInfo() after
SPBOpenDevice() has successfully opened the device. The calls to
SPBGetDeviceInfo() to get this information are listed below:

// get device info
SPBGetDeviceInfo(devRefNum, siNumberChannels, &numChans);
SPBGetDeviceInfo(devRefNum, siSampleRate, &sampRate);
Macworld Mac Programming FAQs

SPBGetDeviceInfo(devRefNum, siSampleSize, &sampSize);
SPBGetDeviceInfo(devRefNum, siCompressionType, &compType);

To have a completion routine executed, assign a pointer to the routine in the completionRoutine field of the sound input parameter block and pass the current value of A5 in the userLong field so it can be retrieved during interrupt time.

mySPB.completionRoutine =
    NewSICompletionProc(MyRecordCompletionProc);

// store A5 in userLong for access by the completion routine
mySPB.userLong = SetCurrentA5();

The completion routine must set up the A5 world to access globals.

pascal void MyRecordCompletionProc( SPB *mySPBPtr )
{
    long myA5;
    myA5 = SetA5(mySPBPtr->userLong);
    gRecordComplete = TRUE;
    myA5 = SetA5(myA5);
}

Using the Sound Manager

This section of the Sound chapter answers FAQs on how to use the Sound Manager. It covers questions on how to use sound commands and sound channels.

How do I execute a sound command?

Use SndDoCommand() to put a command on the sound channel queue. Use SndDoImmediate() to execute the command immediately. To affect a currently playing sound you will need to use SndDoImmediate(). If you use SndDoCommand() your command will not execute until after the sound plays.

Related FAQ

See also FAQ 27-2, “What are sound commands?”
How can I store more than 128 commands in a sound channel?

The default Sound Manager channel can hold up to 128 commands at one time. To be able to handle more than 128 commands at once you will have to create and initialize your own sound channel. To do this you will have to modify the SndChannel structure and increase the size of the field queue, the array that holds the sound commands. Next, you allocate a block of memory to hold your customized channel. Set the qLength field to the number of commands you want to hold. Finally, pass your channel to SndNewChannel().

The example below creates a channel that can handle 256 commands.

```c
//************************
#define kNewQLength 256

void CreateMyOwnChannel( SndChannelPtr myNewChannel )
{

  struct MySndChannel
  {
    struct SndChannel
    {
      SndChannelPtr nextChan;
      firstMod;

      SndCallBackUPP callBack;
      serInfo;

      long wait;

      SndCommand cmdInProgress;

      short flags;
      qLength;
      qHead;
      qTail;
      SndCommand queue[kNewQLength];
    }
  }

  OSErr err;
```
myNewChannel = (SndChannelPtr)NewPtr(sizeof(struct MySndChannel));
if (myNewChannel = nil)
    return;

myNewChannel->qLength = kNewQLength;

err = SndNewChannel( &myNewChannel, sampledSynth, 0, nil);
if (err != noErr)
    return;

---

**How do I unlock and release a sound resource and dispose of a sound channel when I play a sound asynchronously?**

The sound channel and sound resource cannot be released until the asynchronous sound completes playing. You can determine whether or not an asynchronous sound has completed playing by continuously polling the status of the sound channel with SndChannelStatus(), or by installing a callback function that gets executed upon completion of the sound playing. When the sound has completed playing you can then release the sound resource and dispose of the sound channel.

**Related FAQ**

See also FAQ 27-14, "How can I tell when a sound playing asynchronously has completed playing?"

---

**Can I dispose of my sound handle and sound channel in my completion routine?**

No. The callback procedure installed in a sound channel and the completion routine passed to SndStartPlay() are called at interrupt time. Consequently they cannot move memory. Have the routine set a global flag and free your memory in your main application in response to the global flag.
How can I calculate the sample rate of a sound playing on a sound channel from the value returned from the getRateCmd sound command?

See FAQ 27-10, "How can I change the playback rate, to slow down or speed up the sound playing?"

Why can't I get a ‘snd’ resource? I know it is in my resource file.

The sound resource is a ‘snd’ resource not ‘snd’. Make sure you are including the blank space at the end.

Related Topic

See also Chapter 25, "Resources."

When is the callback routine for my sound channel executed?

Unlike the completion routine of SndStartFilePlay() which gets executed automatically after the sound completes playing, the callback routine assigned to a sound channel does not get executed until you tell it to. You do this by sending a callBackCmd sound command to the channel.
The Macintosh excels as a multimedia machine. Its ability to display realistic colors, play sounds, and run QuickTime movies are just some of the most frequently noted examples of this. In the summer of 1993 Apple added yet another dimension to the Macintosh's interface capabilities: the ability to incorporate synthesized speech into an application. Apple introduced the Speech Manager which gives your application an easy way to convert text to speech. In this chapter, you'll learn how to use the Speech Manager to speak a buffer of text, how to change voices, how to tell when your speech segment is finished, how to synchronize your speech, how to build a list of all available voices, and how to stop speech on command.

How can I have Speech Manager speak a buffer of text?

There are three routines to speak text: SpeakBuffer(), SpeakText(), and SpeakString(). You can simply call SpeakString() with Pascal string and the Speech Manager will speak it. The other two routines require a speech channel to be created and passed to them.
The sample below allocates a speech channel, begins speaking the text by calling `SpeakText()`, and stops speech when the mouse button is pressed.

```c
//***************************
void SpeakSomeText()
{
    SpeechChannel mySpeechChan = nil;
    Str255 myStr = {"\pSpeak this text to me"};
    OSErr err;

    err = NewSpeechChannel(nil, &mySpeechChan);
    SpeakText(mySpeechChan, (char *)&myStr[1], myStr[0]);

    while ( SpeechBusy() )
    {
        if ( Button() )
            StopSpeech( mySpeechChan );
    }

    DisposeSpeechChannel(mySpeechChan);
}
```

**How can I stop a channel from speaking once it starts synthesizing text?**

You can stop the speech on a channel by calling `StopSpeech()` and `StopSpeechAt()`. You can also call `PauseSpeechAt()` to stop the speech, allowing a call to `ContinueSpeech()` to resume speaking.

**How can I tell when my speech channel has completed speaking text?**

The call `SpeechBusy()` returns true if any speech channels are currently synthesizing speech. To determine if a particular speech channel is done, implement a call back routine that is executed when the channel has completed synthesizing the text.
**How can I get a list of all of the available voices?**

You can iterate through all of the voices installed on a system by using CountVoices() to get the number of installed voices and GetIndVoice() to get the voice associated with an index. The function GetVoiceDescription() passes back a voice description which can be used to get information about the voice. Among the information available in the voice description is the name, gender, and approximate age in years of the voice in the name, gender, and age fields.

The sample below iterates through all of the voices and gets each voice description.

```c
void GetAllVoices()
{
    OSErr err;
    VoiceSpec voice;
    VoiceDescription voicedesc;
    short voiceCount, i;
    err = CountVoices(&voiceCount);
    for (i = 1; i <= voiceCount; i++)
    {
        err = GetIndVoice(i, &voice);
        if ( err == noErr )
            err = GetVoiceDescription(&voice, &voicedesc, sizeof(voicedesc));
        if ( err == noErr )
            // do something with voice
    }
}
```

**How can I select a voice other than the default voice to speak?**

When you create a channel with NewSpeechChannel() and you specify nil as the first parameter, the default voice is selected. To speak with a voice other then the default voice, pass the voice specification of another voice to NewSpeechChannel(). All text spoken on that channel will use the voice.
The sample below finds all of the boys' voices and speaks with them.

```c
//*************************
void AllBoysSpeak()
{
    OSErr err;
    VoiceSpec voice;
    VoiceDescription voicedesc;
    Str255 myStr = "\pHi, this is a test.";
    SpeechChannel chan;
    short voiceCount, i;
    err = CountVoices(&voiceCount);
    for (i = 1; i <= voiceCount; i++ )
    {
        err = GetIndVoice(i, &voice);
        if ( err == noErr )
            err = GetVoiceDescription(&voice, &voicedesc,
                                         sizeof(voicedesc));
        if ( err == noErr )
        {
            if ( voicedesc.age < 16 &&
                 voicedesc.gender == kMale)
            {
                NewSpeechChannel(&voice, &chan);
                SpeakText(chan, (char *)&(myStr[1]),
                           myStr[0]);
                while ( SpeechBusy() );
                DisposeSpeechChannel(chan);
            }
        }
    }
}
```

**How can I synchronize speech from a speech channel with other actions?**

The Speech Manager allows you to implement callback procedures that can be executed as the result of a number of different actions. You assign a callback procedure for a specified action to a speech channel with the SetSpeechInfo() which is used to change the settings of a speech channel. The function is defined as
SetSpeechInfo( SpeechChannel chan, OSType selector, Ptr SpeechInfo)

where selector specifies the type of information being changed, and SpeechInfo is a pointer to a value dependent upon selector. There is a special selector, soSyncCallback, which can be generated at any point during the speech processing to synchronize to an external action. The selector values used to set a callback procedure are:

- soErrorCallback: called whenever an error is encountered
- soPhonemeCallback: called every time the Speech Manager is about to generate a phoneme
- soSpeechDoneCallback: called when the Speech Manager has finished generating speech on a channel
- soSyncCallback: called whenever the Speech Manager encounters a synchronization command either within an embedded speech command or within the text to be synthesized
- soTextDoneCallback: called when Speech Manager has finished synthesizing speech
- soWordCallback: called every time the Speech Manager is about to generate a word

If a sound callback routine is to access an application global, it must set up the application’s A5 world.
Strings, Bit Manipulation, and Random Numbers

Every program contains at least some code that performs operations on strings and bits, whether it is retrieving input from a user in a TextEdit field or testing a bit flag returned from a Gestalt call. In this chapter you'll learn how to integrate the Pascal style strings from the Toolbox into your C code, get a handle to a string for use with some functions that require them, and other string manipulation functions. You'll also learn how to test and set the values of individual bits. And, added here for your viewing pleasure, are a few FAQs on random number generation.

There are three subsections in this chapter:

• Strings
• Bit Manipulation
• Random Number Generation
Strings

This section of the Strings, Bit Manipulation, and Random Numbers chapter answers FAQs about manipulating strings. The Toolbox calls receive and return Pascal style strings. In C programs this always causes the extra headache of having to convert C strings to Pascal strings and vice versa. This section shows how to do this as well as how to convert strings to and from other data types.

How do I convert a C string to a Pascal string and vice versa?

C strings are a series of characters terminated by the null character, '\0'. Pascal strings are a series of characters whose length is stored in the first byte. Consequently Pascal strings have a maximum size of 255 characters, the maximum number that can be represented by a byte.

There are two functions that convert a C string to a Pascal string and two functions that convert a Pascal string to a C string.

```c
StringPtr c2pstr(char *aStr);
pascal StringPtr C2PStr(Ptr cString);

char *p2cstr(StringPtr aStr);
pascal Ptr P2CStr(StringPtr pString);
```

If you want to do this yourself, to convert from C to Pascal, you can use the BlockMove() function to shift the characters down one byte and set the first byte to the size. If you are converting from Pascal to C, use the BlockMove() function to shift the characters up one byte and set the last byte to null. An example of this is in the routine below.

```c
//***********************
char *PtoCStringConvert( Str255 thePString)
{
    int    len;

    len = thePString[0];
    BlockMove( &thePString[1], &thePString[0], len);
    thePString[len] = '\0';
    return( (char *)thePString );
}
```
Related Topic

See also Chapter 4, "Development Environment and Language Issues."

**How can I convert between a Str255 and an OSType?**

An OSType is a four byte value used to identify file types and file creator types. To convert an OSType to a Pascal string:

```pascal
OSType myOSType;
Str255 osTypeString;

osTypeString[0] = 4;
osTypeString[1] = ((char *) &myOSType)[0];
osTypeString[2] = ((char *) &myOSType)[1];
osTypeString[3] = ((char *) &myOSType)[2];
osTypeString[4] = ((char *) &myOSType)[3];
```

To convert a Pascal string to an OSType:

```pascal
myOSType = *((OSType *) (osTypeString + 1));
```

**How do I get the handle to a string?**

To get a handle from a Pascal string, use the NewString() function. It takes a Pascal string and returns a StringHandle:

```pascal
StringHandle hPString;
Str255 theString;

hPString = NewString(theString);
```

If you want to convert a C string to a handle, create a handle to a block of memory with NewHandle() and then copy the string into it with PtrToHand():

```pascal
Handle hCString;
char *theCString;
hCString = NewHandle( strlen(theCString) );
PtrToHand( theCString, &hCString, strlen(theCString) );
```

If you are going to be doing any manipulation of the string, it is a good idea to set the handle size to the maximum potential string length, in most cases 255, with
How do I convert a string to all uppercase or all lowercase?

UpperString() converts lowercase letters in a Pascal string to uppercase. There is no corresponding LowerString() call.

LowercaseText() converts uppercase characters in a text string to lowercase. UppercaseText() converts lowercase characters in a text string to uppercase. Note that these two functions accept pointers to a string of text and the number of characters to convert and not a Pascal string.

Where can I find functions like strcpy and strcat for Pascal strings?

You can write them yourself. Use BlockMove().

```
#define MIN(a,b) (((a)<(b))?(a):(b))

char *pStrcat( unsigned char *dest, unsigned char *src)
{
    long sLen = MIN(*src, 255 - *dest);
    BlockMove(src + 1, dest + *dest + 1, sLen);
    *dest += sLen;
    return ((char *)dest);
}

char *pStrcpy( unsigned char *dest, unsigned char *src)
{
    BlockMove(src, dest, (long) *src + 1);
    return ((char *)dest);
}
```

Related Topic

See also Chapter 4, "Development Environment and Language Issues."
How do I convert strings to numbers and numbers to strings?

For strings and long integers:

- Call `StringToNum()` to convert a string to a long.
- Call `NumToString()` to convert a long to a string.

To convert a string to a floating point, first call the routine `StringToExtended()` to change the number into type `extended80`, which is the 80-bit floating point type. `StringToExtended()` takes as one of its parameters a number format specification string which is the result of the call `StringToFormatRec()`. The number format specification string establishes the output format of the string. It also handles the different types of number parts separators that are present on different systems (the United States uses the ‘,’ and ‘.’ as the thousands separator and decimal point separator while European conventions reverse the two) through the a number parts table retrieved by the `GetInt1ResourceTable()` call. By modifying the format specification string appropriately, you can specify specialized formats. For example, you can put parenthesis around negative numbers.

To convert a floating point to a string, you use the same procedure as converting a string to a floating point, except you use the `ExtendedToString()` call.

Below is an example that takes a string containing the number 12,345.123 and converts it into a float. It then takes the float and converts it into another string.

```c
//***************
void StrToFloat()
{
    Handle        it1Handle;
    NumFormatStringRec strFormatRec;
    Str255        numStr = "\p12,345.123", newStr;
    extended80    newNumber;
    long          offset, length;

    GetInt1ResourceTable (smCurrentScript, smNumberPartsTable, 
                          &it1Handle, &offset, &length);

    StringToFormatRec ("\p##,###.0##", 
                        (NumberParts *)(*it1Handle + offset), &strFormatRec);

    StringToExtended (numStr, &strFormatRec,
```
(NumberParts *)(*it1Handle + offset),
&newNumber);
ExtendedToString (&newNumber, &strFormatRec,
(NumberParts *)(*it1Handle + offset), newStr);
}

Another, and much easier, approach is to use the ANSI library command sscanf. Below is an example that converts the contents of a string, myString, into the float, myFloat.

```c
float myFloat;
char *myString[64];
sscanf( myString, "%f", &myFloat);
```

You can use sprintf to turn a float into a string:

```c
sprintf( myString, "%f", myFloat);
```

## Bit Manipulation

This section of the Strings, Bit Manipulation, and Random Numbers chapter shows how to set and test individual bits and perform other bit manipulation functions.

### How can I test if a single bit is set? How can I set a single bit?

BitTest() can be used to test the state of a single bit in a bit string. BitSet() can be used to set a single bit in a bit string. BitClr() can be used to clear a single bit in a bit string. These three functions use a reversed bit numbering scheme; they count the bits from left to right starting at the byte specified by the passed in address. This scheme is the reverse of the bit numbering scheme used by the MC680x0 processors.

A reversed bit numbering scheme appears this way:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ...
```

A MC680x0 processor bit numbering scheme appears this way:

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 15 14 ...
```
To test bit number 6 in a byte you would call `BitTst` since bit number 6 in the byte is in position number 1 in reversed numbering scheme. To get the reverse numbering position of bit number subtract the bit number from 7 (7 - 6 = 1).

Masks can also be used in conjunction with bit operations to get and set bit values. To clear a bit, do a bitwise AND (&) with a mask that has the bit cleared. To set the bit, do a bitwise OR (I) with a mask that has the bit set. To test the value, do a bitwise AND (&) with a mask that has the bit set:

- `bitsValue & c = 0xFFF0`: clears bits 0 to 3
- `bitsValue = 0x000F`: sets bits 0 to 3
- `bitsValue & 0x0001`: tests the first bit

There are Toolbox routines to perform these logical bit operations. They are `BitAnd()`, `BitOr()`, `BitNot()`, `BitShift()`, and `BitXor()`.

Below is an example of using `BitTst()` to interpret a bit field response from Gestalt. The defined constants are based off of the normal bit numbering scheme. To examine them with `BitTst()`, we must subtract the value from 31 (the number of bits in the long response). The constant bit flag, `gestaltTempMemSupport` (which equals 4), is used to indicate if temporary memory is available.

```cpp
//************************
Boolean IsTempMemSupported()
{;
    short     myBit, myErr;
    long      theResponse;

    Gestalt(gestaltOSAttr, &myFeature);
    myBit = gestaltTempMemSupport;
    if (BitTst(&myFeature, 31-myBit))
        return (true);
    else
        return (false);
}
Random Number Generation

This section of the Strings, Bit Manipulation, and Random Numbers chapter answers FAQs about generating random numbers. It covers how to get random numbers returned that are within a range other than the standard -32767 to 32767. It also shows you how to avoid getting random numbers that repeat a pattern.

**Random() returns values from -32767 to 32767.**

**How can I get random numbers in a smaller specified range?**

To return a random number that falls within the range, upperlimit to lowerlimit, use the formula:

```
number = (Random() % (upperlimit - lowerlimit)) + lowerlimit;
```

**How are random numbers generated on the Mac?**

The Random function in QuickDraw is based on the formula:

```
randSeed := (randSeed * 16807) MOD 2147483647
```

It returns a value between -32767 to 32767 and updates the unsigned 32-bit low-memory global randSeed to generate another random number for the next call.

**Why aren't the numbers generated by Random() random?**

The randSeed global variable must be set to different values to generate different sequences of the random numbers. If you set randSeed to 1 and then call Random() ten times you will get the same sequence of numbers that you would if you reset randSeed to 1 and call Random() ten more times. One way around this is to set randSeed to the value returned by TickCount() before each call to Random().
**How can I get random numbers greater than 32767?**

The `Random()` function restricts you to a range of numbers -32767 to 32767. To get true random numbers greater than 32767, call `Random()` and place the 16 bit returned value in the low word of a long and then call `Random()` again and place the returned value into the high word of the long.

Multiplying the results of two calls to `Random()` won't produce truly random numbers since prime numbers will never be generated and numbers with many factors will get produced often.
System Extensions and Patching Traps

The System extension (INIT) and trap patching mechanisms will likely change dramatically once the Copland Operating System is released. Bearing that in mind, this chapter will answer questions about both of these complex topics. System extensions are special pieces of code that run at startup time, before any applications are up and running. Patches are pieces of code that are called in place of a specified Toolbox function, usually calling the original Toolbox code either right before or immediately after the patch executes. Writing an extension is usually not difficult. Patching traps is easy to do wrong and difficult to do right. Avoid patching traps. Don't do it. There. Now, if you insist, here are the questions.

There are three subsections in this chapter:

- Definitions
- System Extensions
- Traps
Definitions

This section of the System Extensions and Patching Traps chapter answers FAQs about some important terms that are discussed in the following sections. It provides detailed descriptions of traps and System extensions.

What is a trap?

Here's what Ultimate Mac Programming has to say about traps:

“In the days before the Macintosh, the most popular personal computer was the Apple II. Like most computers, the Apple II had a ROM. As had become the industry practice the Apple II ROM contained the code needed to start up the computer, as well as some nice utility routines. Savvy developers knew where these routines lived in ROM and accessed them directly whenever necessary.”

As they were planning the Macintosh, Apple wanted to follow this design. However, there were several problems with programs jumping directly into the ROM to access utility routines. First off, during development the ROMs were constantly changing, so jumping directly into ROM just wasn’t practical. In addition, with only 64K of RAM to work with (this was bumped to 128K just before the Mac was released), space was at a premium, and at 6 bytes of code per call, jumping directly into ROM was RAM-intensive. Finally, some parts of the OS changed so frequently that they never made it into ROM, so jumping to these routines was out of the question. Apple needed an alternative.

The solution Apple arrived at was the trap, a mechanism that allowed developers to access Macintosh OS routines without having to know where they were. By using traps, Apple was able to replace what would have been 6-byte long JSR or JMP instructions directly into ROM with 2-byte long “A-line instructions.” For example, without using traps, calling the QuickDraw routine MoveTo() would have required something akin to “JSR $40814EBA.” On the other hand, while using traps the A-line instruction $A893 performs the same function in 1/3 the space.

Traps are made possible by the Motorola 680x0 exception handling architecture. Every instruction executed by the 680x0 has the potential to generate an exception. Examples of 680x0 exceptions are the bus error, address error, illegal instruction, and zero divide. When an exception occurs the processor saves a small amount of information about what caused the exception on the stack, looks the exception up in a special table known as the exception vector table, and retrieves the address of a routine designed to handle that exception. It then jumps to that routine.
CHAPTER 30: System Extensions & Patching Traps

The exception we are interested in is the line 1010 emulator exception. This exception occurs whenever an instruction that starts with the four bits 1010 is executed. Since 1010 is equivalent to the hex digit A, these instructions are known as A-line instructions, and the exceptions they cause are known as A-line exceptions. Since the term, "trap" and "exception" are often synonymous, both the instructions and the exceptions they generate are sometimes known as A-traps, or just plain traps.

As is the case with any other exception, there is an entry in the exception vector table for the A-line exception. When your Mac starts up, the Operating System places the address of the routine it wants to have handle all A-line exceptions into the A-line exception vector. This routine is known as the trap dispatcher.

The trap dispatcher maintains two tables known as traps dispatch tables. These tables take the information embedded in a trap instruction and return the address of a routine that corresponds to that instruction. One of these tables contains a list of 256 Operating System routines and the other a list of 1,024 Toolbox routines. Each address in the table is 32 bits long and may point to either ROM or RAM. Once the trap dispatcher retrieves the address of the routines that corresponds to the current trap, control is transferred to that routine.

For more information check out the “Patching Traps” chapter in Ultimate Mac Programming and (if you can get your hands on it) Motorola’s M68000 16/32-bit microprocessor programmer’s reference manual. Both of these are definitely worth the read.

What is the Unimplemented() routine?

The Unimplemented() routine is a special routine whose address is stored in both of the trap dispatch tables any place where there is no routine defined to go along with a particular trap number. If a trap is called that has the address of the Unimplemented() function associated with it, the System error 12, dsCoreErr, is triggered which crashes the currently running application.

Related FAQs

See also FAQ 11-1, “How can I tell if Gestalt Manager is available?”

See also FAQ 30-15, “How can I tell if a System routine is available in my System?”
What is a System extension (‘INIT’)?

A System extension, also referred to as an INIT, is a file of type ‘INIT’ that contains an ‘INIT’ code resource that is loaded into memory and is executed at System startup. At the end of the startup process, the System folder is searched for files of type ‘INIT’. These files are then opened and any ‘INIT’ resources that they contain are executed. The period of the System startup process during which this occurs is called INIT time. Applications do not load until after INIT time. The first application that is loaded is usually the Finder.

Under System 7, the first INITs that are loaded are those that exist in the Extensions folder. They are loaded in alphabetical order. Then any INITs that exist in the Control Panels folder are loaded in alphabetical order. Then the INITs that exist in the root level of the System folder are loaded in alphabetical order.

One of the most common functions of an INIT is to install a patch into the System. Since INITs are loaded before all applications, their enhancements can affect all processes that are executed. Apple often uses INITs to install System enhancements. Two examples of these are the Drag-and-Drop Manager and the Speech Manager.

What is the difference between a head patch and a tail patch?

A head patch is a routine that performs its added functionality and then calls the existing trap. It does not regain control after the execution of the original System software routine.

A tail patch is a routine that performs its added functionality after calling the existing trap. It regains control after the execution of the original System software routine.

There is another type of patch called a come-from patch, or a System patch, which is used by Apple to replace existing code or add capabilities not already in ROM. When a come-from patch is executed, it checks the return address on the stack to determine where it was called from. If it was called from code in ROM that needs to be fixed, then the modifying code is applied. Otherwise the come-from patch executes the next routine as usual.
System Extensions

This section of the System Extensions and Patching Traps chapter answers FAQs about creating and executing System extensions. It covers having the extension display an icon at System startup, how you can communicate with and control the extension, and other items.

How do I create a System extension?

To write a System extension that simply beeps when it gets executed at System startup is trivial. Here it is:

```c
#include <sys.h>

void main(void) {
    SysBeep(10);
}
```

To have this code executed at INIT time there are some additional steps you must follow when creating the extension.

1. When you create the project, set it to be a code resource of type ‘INIT’.
2. Set the resource attribute “System Heap” and “Locked” flags. This is only required if your extension is going to remain in the System heap.
3. Set the filetype of the built code resource file to ‘INIT’ (you can set the creator signature to any value). This allows the extension to be executed at INIT time.
4. Assign a file name to the extension.

To have the extension executed, drag the built file into the System folder. The Finder should place it in the Extensions folder.

While beeping may be fun, you will probably want to have your System extension do a little more. If you want to have the System extension access global variables in the 68K environment, you will have to setup the A4 register. Note that this A4 register setup is not necessary on the PowerMac. The code below shows how to setup the A4 register in CodeWarrior:

```c
long oldA4;

// Set up A4, so we can access our globals.
```
oldA4 = SetCurrentA4();

    // Restore the value of A4 on the way out.
    SetA4(oldA4);

    In Think C the A4 register is set up like this:

    // Set up A4, so we can access our globals.
    RememberAO();
    SetUpA4();

    // Do your stuff

    // Restore the value of A4 on the way out.
    RestoreA4();

Most of the time you will want to have your code in your System extension code to remain in memory after INIT time. An example of this is an extension that installs a patch to a trap. The key here is the call to DetachResource(). The routine that is used to patch the SystemEvent() routine, MySystemEvent(), is contained in the ‘INIT’ resource. This code is part of a resource file and gets removed from memory when its resource files are closed. This happens when the main routine exits. DetachResource() dissociates the code resource from the resource file and prevents this. It is important to remember that the code resource must be loaded into the System heap or it will get destroyed when the extension’s temporary heap is destroyed.

 электро

 pascal short (*gOldSystemEvent)(EventRecord *theEvent);
pascal short MySystemEvent(EventRecord *theEvent);

void main(void)
{
    long    oldA4;
    THz    oldZone;
    // Set up A4, so we can access our globals.
    oldA4 = SetCurrentA4();

    oldZone = GetZone();
    SetZone( SystemZone() );

    // detach our code, so that we stay around.
    DetachResource( GetlResource('INIT', -4048) );

    // Patch SystemEvent.
    gOldSystemEvent = (void *) GetToolTrapAddress(0xA9B2);
SetToolTrapAddress( NewRoutineDescriptor((ProcPtr)
   &MySystemEvent,uppSystemEventProcInfo,
   kPowerPCISA), 0xA9B2 );

   // Restore the old zone again
SetZone( oldZone );
   // And restore the value of A4 on the way out.
SetA4( oldA4 );

There are some other considerations to beware of. If your 68K System extension code extends beyond a single segment, you will have to do additional work.

**How can my System extension tell when INIT time is over?**

It may be necessary to prohibit a System extension from processing until after the System has completed the execution of all extensions. One way to do this is to post a notification request to the Notification Manager since the Notification Manager doesn't usually start processing requests until INIT time is over. Be aware, however, that, if other extensions display windows during INIT time, it is possible that events will occur before INIT time is over. This may cause your notification request to be executed prematurely. An additional check is needed. The Process Manager is not available until after INIT time. If Gestalt reports that the Process Manager is not available, then INIT time is not over. You will need to reinstall your Notification Manager request to check again.

**How do I get my System extension to show its icon at startup?**

There is code that displays the icon of an System extension at startup time, called ShowInitIcon (written by Peter N. Lewis, Jim Walker, and Francois Pottier). It is available from America Online and at various ftp sites. This code can be linked into your System extension or called as a separate code resource.

ShowInitIcon takes the ID of the icon resource and displays it at the appropriate position in the screen. If it detects that System 7 or greater is running and Color QuickDraw is available, then it calls PlotIconID to draw the icon. If it detects that System 6 is running, then it draws an ICN# member of the icon family. ShowInitIcon properly handles the wrapping around of icons if more than one row has been displayed.
If your extension cannot be installed, then it is the convention to display your icon with a red X through it. To do this, create a separate icon to display this image.

**How can I control the execution of my System extension?**

You can control a task that is started by a System extension with a control panel.

**Related FAQ**

See also FAQ 30-9, “How do I get my extension and control panel to communicate or share data?”

**How do I get my extension and control panel to communicate or share data?**

You will frequently find yourself wanting to establish communications between a control panel and an extension. For example, you might have an extension that loads code in the System heap that waits for a specific keypress, beeping when that key is pressed. The extension might have an associated control panel that allows you to turn this code on or off. In other words, the extension provides the background code and the control panel provides the user interface. This question concerns the ability to communicate between the user interface code in the control panel and the extension code that performs the requested function.

There are several ways to establish a link between an extension and a control panel. Some folks use a custom resource as a go-between. The control panel opens the resource and, as the user changes the control panel settings, writes the resource back out to disk. When the extension loads, it uses the data in the resource to establish its initial settings.

Another approach is to create a Gestalt selector function that returns a pointer to a block of shared global data in the System heap. This is definitely the preferred approach, since it deals nicely with the case where the control panel is open while the extension code is still loaded in memory. There is a great piece of code (written by Jorg Brown) in the Patching Traps chapter in *Ultimate Mac Programming* that demonstrates this technique.
Related FAQs

- See also FAQ 11-4, "How do I register new selectors in Gestalt?"
- See also FAQ 11-6, "How can I use Gestalt to communicate between processes?"

**Why do I have problems getting a PicHandle at INIT time?**

The application heap during INIT time gets reinitialized for each INIT. To load a picture properly, load it into the System heap. You can do this by setting the System heap zone to be the current zone with `SetZone()` before you call `GetResource()`, or you can set the `SysHeap` resource attribute of the picture resource.

**Traps**

This section of the System Extensions and Patching Traps chapter answers FAQs about patching traps. It covers such topics as determining the address of a patch, finding the trap number for a routine, and determining whether a routine is an Operating System trap or a Toolbox trap. It also discusses different ramifications of patching traps.

**What does patching a trap mean?**

When a trap is patched, its entry in the trap dispatch table containing the address of the routine that is called is replaced with the address of another function. When the trap is called, the new function is executed. The functions `NSetTrapAddress()`, `SetToolTrapAddress()`, and `SetOSTrapAddress()` are used to install pointers to custom functions in the Toolbox trap dispatch table and the OS trap dispatch table.
How do I patch a trap?

When patching a trap, there are a lot of points that you need to consider.

- Which routine do you want to patch?
- Is the routine a Toolbox trap or an Operating System trap? A register-based trap or a stack-based trap?
- Does the routine move memory?
- Do you want to create a head patch or a tail patch?
- Do you want the patch to run on a 68K Macintosh or do you want to create a native patch?

Answers to each one of these questions affect how the patch is written.

A trap is patched when the function pointer in its entry in the trap dispatch table is replaced by a pointer to a new function. To do this you need first to get the trap number of the routine you want to patch. Then you must determine whether the trap exists in the Operating System trap dispatch table or in the Toolbox trap dispatch table. This is because there are different routines used to get and set the address of a routine depending upon the dispatch table in which the routine exists. With this information you can set the address of the trap. If you were patching a Toolbox trap you would patch the trap with the routines GetToolTrapAddress() and SetToolTrapAddress() like this:

```pascal
short (*gOldSystemEvent)(EventRecord *theEvent);

gOldSystemEvent = (void *) GetToolTrapAddress(0xA9B2);
SetToolTrapAddress(NewRoutineDescriptor((ProcPtr)
    &BeeperSystemEvent, uppSystemEventProcInfo,
    kPowerPCISA), 0xA9B2);
```

Patching an Operating System trap is done similarly except you would use GetOSTrapAddress() and SetOSTrapAddress(). You can also use NGetTrapAddress() and NSetTrapAddress() to get and set the trap addresses in the dispatch tables. They both have a parameter that specifies which table to access.

Now that you know how to install a patch you need to write the patch routine itself. Your patch routine should call the original function that it replaced in the dispatch table. If it calls the original function in the before it performs any of its processing, it is known as a “tail patch.” If it performs its processing before calling the original
function, it is known as a “had patch.” The address of the function being replaced
is returned by the address setting functions. Your patch should use this to execute
the function it replaces.

If the routine that you are patching does not move or purge memory, then your
patch should not call any routines that perform these functions either. Like any code
resource, a patch must setup the A4 register to access globals.

One of the most common, and easiest traps to patch is SystemEvent(). Patching
SystemEvent() will allow you to examine every event that occurs in the System.
The following example is a piece of code (written by Jorg Brown) in the Patching
Traps chapter in Ultimate Mac Programming. It shows SystemEvent() being
patched by a System extension. The patch that is installed looks for key-down
events and calls a routine to handle them.

```c
void main(void)
{
    long oldA4;
    THz oldZone;

    // Set up A4, so we can access our globals.
    oldA4 = SetCurrentA4();

    // Set the current zone to the system zone.
    // In the 680x0 case, this
    // is not necessary, but it's not a bad idea
    // and it keeps us out of
    // trouble when traps that we don't expect to have side effects
    // unexpectedly allocate memory from the current zone.
    // One example of this is the NewRoutineDescriptor routine.
    oldZone = GetZone();
    SetZone(SystemZone());

    // We need to detach our code, so that we stay around.
    DetachResource(GetResource('INIT', -4048));

    // Remember the old implementation of SystemEvent.
    gOldSystemEvent = (void *) GetToolTrapAddress(0xA9B2);

    // Patch ourselves in.
    SetToolTrapAddress(NewRoutineDescriptor((ProcPtr)
        &BeeperSystemEvent,
```
You can install a trap either from an application or from a System extension. If you install a trap from an application it will only affect that application. If you install the patch from a System extension at startup time, it will affect all applications.
There are several features of SystemEvent() that contribute to the ease of patching it: it is a stack-based trap; it is documented as not preserving any scratch registers; and it can move memory. Patching OS traps or register-based Toolbox traps and writing native patches usually requires Assembly code and other special steps.

For more information on patching traps check out *Ultimate Mac Programming* by Dave Mark and *Power Macintosh Programming Starter Kit* by Tom Thompson.

**Where can I find the trap numbers for routines?**

The universal header file, Trap.h, contains the trap numbers assigned to their corresponding functions. The individual header files for the different managers also map the trap numbers to the function declarations.

Below is an excerpt from Trap.h:

```c
enum {
    /* Palette Manager */
    _InitPalettes = 0xAA90,
    _NewPalette = 0xAA91,
};
```

Below is an excerpt from Palette.h:

```c
extern pascal void InitPalettes(void)
    ONEWORDINLINE(0xAA90);

extern pascal PaletteHandle NewPalette(short entries,
    CTabHandle srcColors, short srcUsage, short srcTolerance)
    ONEWORDINLINE(0xAA91);
```

**How do I get the address of a trap?**

To get the address of a trap you can call NGetTrapAddress(), GetOSTrapAddress(), and GetToolTrapAddress(). GetOSTrapAddress() is used with OS traps. GetToolTrapAddress() is used with Toolbox traps. NGetTrapAddress() can be used with both with the second parameter specifying the trap type. The type of the trap can be determined by bit number 11 (0 if OS trap, 1 if Toolbox trap).
For example if you want to get the address of SystemEvent you first need to find its trap number. Traps.h has the entry:

```
_SYSTEMEvent = 0xA9B2
```

The trap number of SystemEvent() is 0xA9B2. Since this is greater than 0xA7FF, it is a Toolbox routine. To get its address we call:

```
pascal short (*gOldSystemEvent)(EventRecord *theEvent);
gOldSystemEvent = NGetTrapAddress( 0xA9B2, ToolTrap);
```

or we can call:

```
pascal short (*gOldSystemEvent)(EventRecord *theEvent);
gOldSystemEvent = GetToolTrapAddress( 0xA9B2 );
```

To find the address of an Operating System routine, call NGetTrapAddress() with the second parameter set to OSTrap or call GetOSTrapAddress().

### How can I tell if a System routine is available in my System?

You can test if a System routine is available in your System by calling the NGetTrapAddress() and comparing the returned address with that of the Unimplemented() procedure. At startup time the address of Unimplemented() is stored in both the Operating System Trap dispatch table and the Toolbox trap dispatch table any place where there is no routine defined to go along with a particular trap number. If the address returned by the NGetTrapAddress() call for your function is the same address as Unimplemented(), then the function is not available on your System. Before testing the trap against the Unimplemented() address, test if the trap has a trap number greater than the total number of traps on the machine. If it does, then the trap is unimplemented. The total number of traps is determined by testing the __InitGraf trap. If the trap dispatch table contains more than 0x0200 entries, the 0xAA6E always points to either Unimplemented() or something else, but never to __InitGraf.

```c
//**************************************************************************
Boolean IsRoutineAvailable( short theTrap )
{
    TrapType    trapType;
    short       numTraps;
    Boolean     isAvailable;
```
// test if toolbox or os trap
if (( theTrap & 0x0800 ) > 0 )
    trapType = ToolTrap;
else
    trapType = OSTrap;

if ( trapType == ToolTrap )
{
    theTrap = theTrap & 0x07FF;

    // get number of traps in the trap table
    if ( NGetTrapAddress(_InitGraf, ToolTrap ) ==
        NGetTrapAddress( OxAA6E, ToolTrap ) )
        numTraps = 0x0200;
    else
        numTraps = 0x0400;

    if ( theTrap >= numTraps )
        theTrap = _Unimplemented;
}

isAvailable = NGetTrapAddress( theTrap, trapType ) !=
    NGetTrapAddress( _Unimplemented, ToolTrap );

return(isAvailable);

Related FAQs

- See also FAQ 11-1, “How can I tell if Gestalt Manager is available?”
- See also FAQ 30-2, “What is the Unimplemented() routine?”
Macworld Mac Programming FAQs

How can I tell if a routine is an Operating System or a Toolbox trap?

Bit 11 of the trap number indicates if the trap belongs to the Operating System dispatch table or the Toolbox dispatch table. If bit 11 of the trap number is set, then the trap is a Toolbox trap. If it is 0 then the trap is an Operating System trap. This means trap numbers 0xA000 to 0xA7FF are Operating System traps. Trap numbers 0xA800 to 0xAFFF are Toolbox traps.

```c
/*-----------------------------*
OSorToolbox(int trapWord)
{
    if ( (trapWord & 0x800) == 0 )
        return ( OSTrap );
    else
        return ( ToolTrap );
}

There are also some other differences between traps.

Traps 0xAC00 through 0xAFFF:

These traps are shorthand for jumping directly to a stack-based Toolbox routine. The trap dispatcher handles them as if they were a JMP instruction. For example

0xAC73

is equivalent to

JMP SetPort

Traps 0xA800 through 0xABFF:

These traps are shorthand for calling a stack-based Toolbox routine. The trap dispatcher handles them as if they were a JSR instruction. For example
CHAPTER 30: System Extensions & Patching Traps

| 1 | 0 | 1 | 0 | 0 | F | F | 1 | Trap Number (0 - 255) |

0xA873

is equivalent to

JSR SetPort

Traps 0xA100-0xA1FF, 0xA300-0xA3FF, 0xA500-0xA5FF, 0xA700-0xA7FF:

These traps are shorthand for calling a register-based OS routine. The trap dispatcher saves registers A2, D2, D1, and A1 onto the stack, then calls the specified routine with JSR, restores the registers, and returns.

0xA51E

is equivalent to

| 1 | 0 | 1 | 0 | 0 | F | F | 0 | Trap Number (0 - 255) |

movem.l D1 - D2/A1 - A2, -(a7)
move.w #$A51E, D1
JSR NewPtr
    movem.l (a7)+, D1 - D2/A1 - A2

Traps 0xA000-0xA0FF, 0xA200-0xA2FF, 0xA400-0xA4FF, 0xA600-0xA6FF:

These traps are shorthand for calling a register-based OS routine. The trap dispatcher saves registers A2, D2, D1, A1, and A0 onto the stack, then calls the specified routine with JSR, restores the registers, and returns.

0xA41E

is equivalent to

movem.l D1 - D2/A0 - A2, -(a7)
move.w #$A41E, D1
JSR NewPtr
    movem.l (a7)+, D1 - D2/A0 - A2

If I patch a trap from my application, will it affect all
applications?

Each application can be thought of as having its own copy of the two trap dispatch tables (more on this later): the Operating System trap dispatch table, which contains a list of 256 OS routines; and the Toolbox trap dispatch table, which contains a list of 1,024 Toolbox routines. The application accesses these copies to get the address of a desired routine. When the application patches a trap, it only modifies its own copies of the dispatch tables. Therefore, a patch only affects the application that installed it.

When a patch is installed by a System extension during System startup, before any applications are launched, it is installed into the tables maintained by the System. When applications get launched they will copy these tables and the patches, installed by the System extensions. Therefore, a patch installed at startup time will affect all applications.

Now back to the dispatch tables. Each application does not actually have its own copy of the trap table. The Process Manager tracks what patches which traps by being hooked into SetTrapAddress. Part of each context switch is unhooking one app’s patches and hooking up another app’s patches. This is so an application’s patches only run in the context of the patching application. When an application quits, the Process Manager unhooks any patches the application has installed.

How can I have a patch to a trap and have it affect all applications?

For a patch to affect all applications, it must be installed by a System extension during System startup, before any applications are launched.

See FAQ 30-17, “If I patch a trap from my application, will it affect all applications?”

Can two applications patch the same trap at the same time?

Yes, the same trap can be patched multiple times. However the effects of these patches depend upon when they are installed.

A patch that is installed by an application affects only the copies of the dispatch
CHAPTER 30: System Extensions & Patching Traps 543

tables maintained by that application. This means that multiple applications can patch the same trap without affecting each other.

A patch that is installed by a System extension at System startup, before any applications are launched, affects the dispatch tables maintained by the System. So, what happens if two extensions patch the same trap? Part of the functionality of a routine that is installed as a patch is to execute the code that the patched trap previously called, the code pointed to by the address that was in the trap dispatch table. This results in a chaining effect of called routines. For example, if an extension patches SystemEvent() with the function TrapFunctionOne(), TrapFunctionOne() will call the function whose address was in the trap table at the trap number 0xA9B2, in this case the original SystemEvent(). If a second extension then patches SystemEvent() with TrapFunctionTwo(), TrapFunctionTwo() will call the function whose address was in the trap table at that time - that function is now TrapFunctionOne(). So when an application calls the SystemEvent() routine to be executed (by calling WaitNextEvent()) the sequence of functions called is:

TrapFunctionTwo() → TrapFunctionOne() → SystemEvent()

Can my patch routines move memory?

When you patch a trap that is defined as not moving or purging memory, your patch cannot move or purge memory and can only call other traps that do not move or purge memory. If you patch a trap that is defined as not moving or purging memory and is callable from an interrupt, your patch should call only those other traps that are defined that way.

If you patch a trap that is defined as moving or purging memory, you are not under any such restrictions.

How can I examine keydown and mouse events in all applications?

In order to get all of the keydown and mouse events generated by a user, you will have to intercept the events before they get sent to the target application. Two methods of doing this are patching SystemEvent() and installing a GNE filter. SystemEvent() is called by WaitNextEvent() and GetNextEvent() to determine if an event should be handled by the calling application or by the Operating System. A GNE filter is a procedure that is called from GetNextEvent()
and `WaitNextEvent()` just before those traps return to an application. Both of these methods will allow you to examine all events and perform a corresponding action.

Related FAQs

- See also FAQ 7-12, "How do I capture keystrokes in all applications?"
- See also FAQ 7-30, "How do I create and install a GNE filter?"

Related Topics

- See also Chapter 7, "Events."
Though most of your work with text will consist of calls to `DrawText()` or `DrawChar()`, there will be times when you need to go a step or two further. This chapter defines the various font families and clarifies the differences between outlined and bitmapped fonts. As you read this chapter, you'll learn how to draw dimmed (grayed-out) text, how to work with text modes, and more.

There are three subsections in this chapter:

- Definitions
- Fonts
- Drawing Text
Definitions

This section of the Text: Fonts and Drawing chapter defines several terms relating to fonts. It describes the two basic fonts, the System and application fonts, the font family, and the difference between outlined and bitmapped fonts.

What are the System and the application fonts?

The System font and application font are two special fonts that are recognized by the System. The System font is the font used to display text in menus, dialog boxes, and alerts. It is Chicago 12-point font. The application font is the default font used by applications. This is typically Geneva 12-point font. To use the System or application font, you pass a special font designator to `Text Font()`. For the System font, this is the `systemFont` constant which equals 0. For the application font, this is the `appFont` constant which equals 1. These values are not the true font family IDs. To get the true font family ID for the System font call `GetSysFont()`; for the application font call `GetAppFont()`.

What is a font family?

A font family is a collection of multiple fonts of the same typeface. The fonts in a single font family can be either bitmapped or outlined. Bitmapped fonts in the same family can be of different styles or sizes. The font family resource type is ‘FOND’.

What is the difference between an outlined font and a bitmapped font?

A bitmapped font is designed at a fixed point size for a particular display device. An individual bitmap is provided for each style and size. If a particular style and size combination is requested by the user, but is not available, QuickDraw alters a bitmapped font of a different size. This often produces fonts with jagged edges or irregular shapes. Bitmapped fonts can exist in ‘FONT’ or ‘NFNT’ resource type.

An outlined font is represented by a description of lines and curves that make up the font. There are no point size or display device characteristics attached to it. There is one representation for each typeface and style. From these thousands of
different point sizes can be generated to produce clean and identical looking fonts. TrueType fonts are a type of outline font that use the Apple TrueType format. The outline font resource type is ‘sfnt’.

Fonts

This section of the Text: Fonts and Drawing chapter answers FAQs about fonts. You will learn how to get the font family ID and the name of a font. This section also discusses how to get a list of all available fonts in the System along with other font related issues.

How do I get the font family ID of a font from the name of the font?

To get the font family ID of a font from the font name, call GetFNum(). There is one thing to watch out for. If the font specified by GetFNum() is not present then GetFNum() will return a 0. This is the ID of the System font. If GetFNum() returns a 0, compare the System font name with the name of your font.

The example below gets the font family ID and compares the name against the System name if 0 is returned.

```c
//***************
short GetFontID(Str255 fontName)
{
    short fontFamilyID;
    Str255 systemFontName;

    GetFNum( fontName, &fontFamilyID );
    if (fontFamilyID == 0)
    {
        GetFontName(O, systemFontName);
        if ( !EqualString(fontName, systemFontName, false,
                         false) )
            return -1;
    }

    return fontFamilyID;
}
```
How do I get the name of a font family from the font family ID?

The function GetFontName() returns the name of a font family from a font family ID.

How can I get all available fonts in the System?

You can call GetIndResource() to iterate through all of the font families, 'FOND', available in your system. By calling GetResInfo() you can determine the font names.

How can I tell if a graphics port will use a bitmapped font or an outline font?

The call IsOutline(numer, denom) determines if an outline font will be chosen for the current graphics port to meet the scaling factors specified by the Point parameters numer and denom. By default the Font Manager prefers bitmapped fonts over outline fonts. You can call SetOutlinePreferred() to make outline fonts preferred over bitmapped fonts. To determine what the preference is, call GetOutlinePreferred().

Why does text drawn with a font ID sometimes appear different on different machines?

Many fonts on the same machine may share the same family ID. Even if the desired font is present in the System file, another font with the same ID may appear in the font menu. It is also possible for a font family to have different IDs on different computer systems. If this is the case, the desired font may not be found and another one will be substituted. To avoid these problems, always specify the name of the font rather than the font family ID.
How do I determine the height in pixels of a font?

The height of a font is the sum of its ascent, the measurement in pixels from the baseline to the highest point, and its descent, the measurement in pixels from the baseline to the lowest point. GetFontInfo() returns the ascent, descent, and a third value — the leading of the current font in the current graphics port. Leading is the number of pixels between the descent line and the ascent line.

![Figure 31-1: The ascent, decent, and leading of a font](image)

Drawing Text

This section of the Text: Fonts and Drawing chapter answers questions about the drawing of text. It shows how to gray-out text and rotate text. It also discusses drawing text using transfer modes.

How do I draw text in my window?

There are four functions that draw text in a GrafPort. They each draw at the current pen location and use the text characteristics of the GrafPort. They are:

- **DrawChar**: draws a single character
- **DrawString**: draws a Pascal string
- **DrawText**: draws a block of text from a buffer
- **DrawJustified**: draws a block of text from a buffer and allows scaling, condensing, and extending of text
The text characteristics that can be set are:

- font
- size
- style
- spacing
- drawing mode

They are set by the calls:

- TextFont()
- TextSize()
- TextFace()
- SpaceExtra() and CharExtra()
- TextMode()

respectively. The color of the text can also be set by setting the foreground color.

If you want to draw text and provide text editing capabilities as well as automatic wrapping of text within a rectangular area, you will want to use the features provided by TextEdit.

How do I change the font, style, and size of the text that gets drawn?

The Toolbox contains routines that allow you to manipulate the characteristics of a font. The routines always work on the current graphics port. You can set the current graphics port by calling SetPort().

The routines that allow you to set the font characteristics are listed below.

- TextFont() selects the text font.
- TextFace() sets the style of the font. Table 31-1 lists the valid styles.
Table 31-1:
Font Styles

<table>
<thead>
<tr>
<th>Style</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bold</td>
<td>1</td>
</tr>
<tr>
<td>italic</td>
<td>2</td>
</tr>
<tr>
<td>underline</td>
<td>4</td>
</tr>
<tr>
<td>outline</td>
<td>16</td>
</tr>
<tr>
<td>shadow</td>
<td>16</td>
</tr>
<tr>
<td>condense</td>
<td>32</td>
</tr>
<tr>
<td>extend</td>
<td>64</td>
</tr>
</tbody>
</table>

These values are bit flags that can be logically combined to add or remove an individual style. To draw with a plain font, call TextFace(0).

TextSize() sets the size of the text drawn. You can set the size from 1 to 127. A value of 0 specifies the System font size.

How can I determine the current text characteristics (font, size, mode, and style) that a graphics port will use to draw text?

There are no Toolbox routines that return the text characteristics of a graphics port. To determine these values you can examine the appropriate fields in the graphics port's GrafPort structure. The names of the fields that contain this information are:

- txFont: the font number of the text
- txFace: the text style
- txMode: the text-drawing transfer modes
- txSize: the text size in points

Below is a routine that returns the current text size of a window.

```c
//**************
short GetWindowTextSize( WindowPtr pWindow )
```
return( WindowPtr->txSize );

Related Topic

See also Chapter 23, “QuickDraw: Drawing.”

**Is there any way to determine the length of styled text that is drawn given the font, size, style, etc?**

There are four Toolbox calls that return the widths of text. They all measure the text in the text characteristics of the current GrafPort.

- **CharWidth()**: returns the width of a character
- **TextWidth()**: returns the width of a block of characters in a buffer
- **StringWidth()**: returns the width in pixels of a Pascal string
- **MeasureText()**: fills an array with the pixel offsets corresponding to the width of each character in a specified text buffer

**Can I use transfer modes when drawing text?**

You can use transfer modes with text. However, these are not the same transfer modes as the pattern modes that are used to draw lines and shapes. The transfer modes for text are the source transfer modes. The Toolbox function **TextMode()** is used to select the source transfer mode. The modes are: **srcCopy**, **srcOr**, **srcXor**, **srcBic**, **notSrcCopy**, **notSrcOr**, **notSrcXor**, **notSrcBic**.

**Why doesn’t the text that I draw respond to the mode settings I select with PenMode?**

**PenMode()** affects the pen drawing and painting operations but not text operations. **Use TextMode()** to select a drawing mode for text drawing operations. The valid modes are: **srcCopy**, **srcOr**, **srcXor**, **srcBic**, **notSrcCopy**, **notSrcOr**, **notSrcXor**, **notSrcBic**.
How can I get the current textMode, so I can change it and then restore it when I'm done?

The current text mode can be obtained from the current GrafPort's txMode field. The QuickDraw global qd.thePort represents the current GrafPort. The current text mode is qd.thePort->txMode. Save this before you change the text mode. Restore it by calling TextMode() with the stored value.

```c
//**************
void GetAndRestoreTextMode()
{
    short tMode;

    // get current text mode
    tMode = qd.thePort->txMode;

    // change mode and do your stuff
    // restore old text mode
    TextMode( tMode );
}
```

How do I draw dimmed (grayed-out) text?

To draw dimmed text, call TextMode() to change the text mode to grayishTextOr.

The example below saves the current text mode, changes the text mode to grayishTextOr, draws the text, and then restores the original text mode.

```c
//*******************
void DrawGrayedText()
{
    short tMode;

    tMode = qd.thePort->txMode;

    TextMode(grayishTextOr);
    DrawString("\pDraw me in dimmed text.");
    TextMode( tMode );
}
```
Related FAQs

- See also FAQ 2-10, "What RGB value does the System use for dimmed buttons, menus, and window titles?"
- See also FAQ 6-16, "How do I get my controls to dim with light gray instead of bitmapped gray?"
- See also FAQ 6-28, "How do you deactivate (gray-out) static text items and edit text items in a dialog?"
- See also FAQ 23-37, "How can I make text or an object appear dimmed or grayed-out without redrawing it?"

How can I draw rotated text?

There is no ToolBox call or easy method to draw rotated text. The only way to get rotated text drawn on screen is to draw the text to an offscreen bitmap, manipulate the bitmap yourself, and then call CopyBits() to transfer it to your graphics port.

QuickDraw GX and PostScript calls to PostScript supporting printers rotate text at any angle for you.

The sample below draws a string rotated counter-clockwise 90 degrees on a window. The function DisplayRotatedString() creates a window, calls RotateString90() to get a bitmap containing the rotated text, and then calls CopyBits() to copy the bitmap to the window.
The routine `RotateString90()` is where all of the fun is. First it stores the font information of the window it needs in order to draw the text on the bitmaps. It then creates a graphics port in which to draw the text in order to create the source bitmap. After setting the new graphics port to the window’s font characteristics, it gets the font information about the string and uses it to determine how much memory needs to be allocated to the source bitmap. The string is then drawn onto it. Next the destination bitmap is created. The orientation of the bitmap is determined by the degree of rotation. Since our string is rotated 90 degrees, its height and width are the width and height of the source bitmap. This is represented by the bounds field of the destination bitmap. Finally, the bits of the source bitmap are transferred to the destination bitmap, translated by our rotation algorithm. This particular algorithm rotates the bitmap counter clockwise by 90 degrees.

```c
//********************************
void DisplayRotatedString()
{
    WindowPtr window;
    BitMap destMap;
    Rect destRect;

    window = NewWindow( nil, &windowRect, "$pRotate String", 
            true, 
            noGrowDocProc,(WindowPtr)-1, true, nil);
    SetPort(window);
    TextSize(16);

    // get bitmap that contains string
    // rotated 90 degrees to the right
    RotateString90( "$pRotated String", &destMap);

    // copy rotated string bitmap to window
    destRect = destMap.bounds;
    OffsetRect(&destRect, 20,20);
    CopyBits( &destMap, &window->portBits, &destMap.bounds, 
        &destRect, srcCopy, nil);

    while( !Button());
    DisposePtr(destMap.baseAddr);
    DisposeWindow( window );
}
```
OSErr RotateString90( Str255 str, BitMap *destMap) 
{
    GrafPort srcPort;
    GrafPtr oldPort;
    BitMap srcMap;
    FontInfo fontInfo;
    short fontHeightBits, fontHeightBytes;
    short fontWidth, fontWidthBytes;
    short destWidth, destHeight;
    short row, column;
    short texFont, texFace, texMode, texSize;
    short err;
    char *srcPtr, *destPtr;

    GetPort(&oldPort);

    // get font characteristics of current port
    // string will be drawn in it.
    texFont = qd.thePort->txFont;
    texFace = qd.thePort->txFace;
    texMode = qd.thePort->txMode;
    texSize = qd.thePort->txSize;

    //************
    // create source graphics port. string
    // will be drawn in here
    // to create src bitmap for rotation
    //************
    OpenPort( &srcPort);
    SetPort(&srcPort);

    TextFont(texFont);
    TextFace(texFace);
    TextMode(texMode);
    TextSize(texSize);
    GetFontInfo( &fontInfo);

    fontHeightBits = fontInfo.ascent + fontInfo.descent +
                    fontInfo.leading;
    fontHeightBytes = ((fontHeightBits + 7) & 0x7FF8) >> 3;
    fontWidth = StringWidth( str);
fontWidthBytes = ((fontWidth + 7) & 0x7FF8) >> 3;

srcPort.portBits.baseAddr = NewPtrClear(fontWidthBytes * fontHeightBits);
if(err = MemError())
  return err;
srcPort.portBits.rowBytes = fontWidthBytes;
SetRect(&srcPort.portBits.bounds, 0, 0, fontWidth, fontHeightBits);

// draw string in source bitmap
RectRgn(srcPort.visRgn, &srcPort.portBits.bounds);
MoveTo(0, fontInfo.ascent + fontInfo.leading);
DrawString(str);

//***************
// create the destination bit map. make sure it is
// oriented the right way
//***************
destWidth = fontHeightBytes;
destHeight = fontWidthBytes << 3;
destMap->baseAddr = NewPtrClear(destWidth * destHeight);
if(err = MemError())
{
  DisposePtr(srcPort.portBits.baseAddr);
  return err;
}
destMap->rowBytes = destWidth;
SetRect(&destMap->bounds, 0, 0, destWidth * 8, destHeight);

//***************
// perform the counter clockwise rotation.
// set bits from src to dest bitmap
//***************

// start at top left of src
srcPtr = srcPort.portBits.baseAddr;

// start at bottom left of destination
destPtr = destMap->baseAddr + ((destHeight - 1) * destWidth);
// traverse through all row and column pixels
// for each set src bit set dest bit
for( column = 0; column < fontWidth; column++)
{
    for( row = 0; row < fontHeightBits; row++)
    {
        // Go to the first even byte of row
        // get shifted bit position from column
        if( *(srcPtr + (row * fontWidthBytes) +
            (column >> 3))
            & (0x80>>(column & 7)) )
        {
            // go up to each row (column of src)
            // from destPtr
            // set a bit at the shifted position
            *(destPtr - (column * destWidth) +
                (row >> 3))
                |= (0x80 >>(row & 7));
        }
    }
}
DisposePtr(srcPort.portBits.baseAddr);
ClosePort(&srcPort);
SetPort(oldPort);
return noErr;

Related FAQs

- See also FAQ 18-12, “How can I print rotated text?”
- See also FAQ 23-17, “How do I flip a pixmap (put up its mirror image)?”
- See also FAQ 23-19, “How do I rotate a pixmap?”
- See also FAQ 24-17, “How do I rotate a picture?”
CHAPTER 31: Text: Fonts & Drawing

Topic-Related FAQs

☑ See also FAQ 6-30, "How can I set the fonts and colors of my dialog items?"

☑ See also FAQ 16-17, "How can I change the style, size, and font of a menu?"

☑ See also FAQ 16-18, "How do I make a menu of available fonts?"

☑ See also FAQ 32-6, "How can I determine the line height of a TextEdit field?"
Macworld Mac Programming FAQs
If you plan on adding editable text to your application, chances are you'll take advantage of the TextEdit routines built into the Toolbox. In this chapter, you'll learn how to draw blocks of text in a designated rectangle and have it automatically handle word wrapping. You'll learn how to put up large editable text fields and manipulate them, giving your application TextEditing capabilities. You will also see how to retrieve, save and restore styled text as well as solve many of those annoying little TextEdit problems.

There are three subsections in this chapter:

- Using TextEdit
- Scrolling in TextEdit
- The TextEdit Cursor
Using TextEdit

This section of the Text: TextEdit chapter answers FAQs about putting TextEdit capabilities into an application. It covers how to retrieve both the text and styles from a TextEdit field. It shows how to get various pieces of information about the text such as the height of an individual line (even in a multi-styled TE field) and how to get the start of each line. It also covers problems with word wrapping after characteristics of the TextEdit field have been modified.

How can I draw a block of text in a specified rectangular area?

If the text is static and won't be edited, use TETextBox().

How can I display static text of more than 255 characters?

To display more than 255 characters of static text call TETextBox(). TETextBox() fits the text into a destination rectangle in the current GrafPort and automatically handles line wrapping.

The sample below calls TETextBox() to put static text in the window and then frames the window by calling FrameRect().

```c
//*************
void DoTextBox()
{
    Rect textRect = {30,30, 100, 100}, outlineRect;
    char *myText = {"The string that I want displayed"};

    outlineRect = textRect;
    InsetRect(&outlineRect, -1, -1);
    FrameRect(&outlineRect);

    TETextBox(myText, strlen(myText), &textRect, teCenter);
}
```
**How do I get the text of a TextEdit field?**

The text of a TextEdit field is stored as a handle in the hText field of the TERec structure. To extract the text into a usable form you can call TEGetText() or GetIText(). TEGetText() takes a handle to a TextEdit record and returns a CharsHandle, a handle to an array of characters. The array is not null terminated so before you use it in any routine expecting a C string, make sure that you terminate it with the null character. The number of characters is stored in the teLength field of the TERec structure. Another note, the CharsHandle returned to you is not a copy for you to spindle and mutilate. Use it to make a copy to work with. The code below creates a C string of text from an edit text field.

```c
TEHandle hTE;
CharsHandle text;
char *myCString;

...  
text = TEGetText(hTE);
myCString = NewPtr( (*hTE)->telength + 1 );
BlockMove( *text, myCString, (*hTE)->telength);
myCString[ (*hTE)->telength ] = '\0';
```

GetDialogItemText(), formally known as GetIText(), can be used to copy the text from a TextEdit field into a Pascal string. Unlike C strings, Pascal strings are limited in size. Make sure that you do not use GetDialogItemText() on a TextEdit field that contains more that 255 characters. The code below copies the contents of a TextEdit field into a Str255 variable.

```c
void TEToStr255 (TEHandle hTE, Str255 str)
{
    GetDialogItemText((*hTE)-> hText, str);
}
```

**Related FAQ**

See also FAQ 6-27, “How do I get the text from an editable text item in a dialog?”

**How do I get the text and the style from a multistyle TE field?**

To save and restore text properly from a multistyled edit record, you should save all of the associated character information along with the text itself. A handle to the text of the TextEdit field can be obtained by calling TEGetText(). The style of a
block of text in a TextEdit field can be obtained by calling GetStyleScrap(). This character attribute information can then be reassigned to the block of text when it is restored by calling SetStyleScrap(). GetStyleScrap() takes the handle to an edit record as a parameter. It returns a handle to a style scrap record a (StScrpRec structure) of the text specified by the current selection range.

To copy the contents of a TextEdit field along with its character attribute information to a file, you first select the entire contents of the field by calling TESetSelect(), specifying a range of 0 to 32767. Then you call GetStyleScrap() to get the character attribute information. This style data is then written as a resource. The text data can be written to the data fork of a file. The code below writes the text to the data fork of a file and the character attribute information as a ‘styl’ resource to the file’s resource fork.

```plaintext
TESetSelect(0, 32767, hTE);
myStyle = GetStyleScrap(hTE);
AddResource((Handle)myStyle, 'styl', 128,
    "pmy style record");
WriteResource((Handle)myStyle);
ReleaseResource((Handle)myStyle);
FSWrite( myDataFile, (*hTE)->teLength, (*hTE)->hText);
```

To restore the text and its styles into a TextEdit field, read the data from the data fork and store it in the TextEdit field with TESetText(). Then read its ‘styl’ resource and call SetStyleScrap() to assign it to the text with SetStyleScrap(). The code below does this.

```plaintext
GetEOF(myDataFile, textLength);
myBuffer = NewPtr( textLength );
FSRead( myDataFile, textLength, myBuffer);
TESetText( myBuffer, textLength, hTE );

myStyle = GetResource( 'styl', 128);
DetachResource((Handle) myStyle );
SetStyleScrap( 0, textLength, myStyle, true, hTE);
```

The TextEdit clipboard routines TECopy() and TECut() copy both the ‘TEXT’ element and a ‘styl’ element. When you want to paste the text and its character attribute information, use TESStylePaste().
How can I determine if a TextEdit field is an old type TextEdit field or a style-aware TextEdit field?

You can determine if a TextEdit field is an old type TextEdit field or a style-aware TextEdit field by examining the tSize field of a TRec structure. If it is a negative value, then the TextEdit field is style aware. If it is positive or zero, then the TextEdit field is of the original type.

How can I determine the line height of a TextEdit field?

The method used for determining the height of a TextEdit field depends upon whether or not the TextEdit field is the original type or the style-aware type. If it is the original type, one that supports only a single text style, then this is simple. The line height is stored in the lineHeight field of the TRec structure.

(*myTEHandle)->lineHeight

Determining the line height of a style-aware TextEdit field is more complicated. The height of every line is stored in a table that is part of the style information of the textedit field. To access this table, first call GetStyleHandle() to get a handle to the TextEdit field’s style record. The table is stored in the lTab field. Each line element in the table is a structure of type LHElement and contains the height of the line and ascent of the tallest character in the line.

The code below gets the height of a specified line in the TextEdit field.

```c
//************
short GetLineHeight( TEHandle hTE,
                     short lineNumber,
                     short *lineHeight)
{
  LHHandle hLH;
  LHPtr pLH;
  TESTyleHandle hTEStyle;

  hTEStyle = GetStyleHandle( hTE);
  hLH = (*hTEStyle)->lTab;
  pLH = *hLH;
  *lineHeight = pLH[lineNumber].lhHeight;
}```
Related Topic

See also Chapter 31, "Text: Fonts and Drawing."

How do I process a mouse click in a TextEdit field?

When your application gets a mouseDown event inside the viewRect of a TERec, pass the click on to TextEdit by calling TEClick(). TEClick() takes the mouse point, a boolean indicating that the shift key is down, and the TEHandle as parameters. Convert the mouse location from the mouseDown event where field to local coordinates and pass it as the mouse point. Then determine the state of the shift key by examining the modifiers key of the event as shown below:

\[
is\text{ShiftDown} = (\text{theEvent.modifiers} \& \text{shiftKey}) \neq 0;
\]

Pass the result in the boolean parameter. TEClick() will then properly process the mouseDown event. If the shift key is down, it will extend the selection. If it was not down, it will move the insertion point. TEClick() will also process mouse drags in the TextEdit field.

How do I set the style of the text in a TextEdit field?

There are two types of TextEdit fields, the original monostyle edit record created by TENew() and the multistyle edit record created by TESetStyleNew(). If you want to have text of different styles in your TextEdit field, you must use the multistyle edit record created by TESetStyleNew().

To set the style of all the text in a monostyle TextEdit field, you can edit the fields txFont, txFace, txMode, and txSize in the TERec structure of the TextEdit field. After you modify the style of a TextEdit field you must call TECalText() to recalculate the beginnings of all of the lines of text to have proper word wrapping.

A multistyle edit record gives you much more flexibility when assigning styles to its text. You can set one block of text to one style and another block of text in the same TextEdit field to another style. The call TESetStyle() allows you to set the style of the text that is in the current selection range. It is defined below:

\[
\text{TESetStyle} (\text{short mode, TextStyle newStyle, Boolean redraw, TEHandle hTE})
\]

If the redraw parameter is set to true, then the beginnings of all of the lines are automatically recalculated before they are being redrawn.
How can I get the start of each line in my TE field?

TextEdit keeps track of the start of each actual line in a table. This table is a one-dimensional array of integers and is located in the lineStarts field of the TextEdit field's TERec structure. Each element represents an offset into the text data to the start of the corresponding line.

Why doesn't changing the sizes of my destination and view rectangles affect the word wrapping in my TextEdit field?

If you want to resize your TextEdit field, the first and most obvious steps to take are to alter the destination and view rectangles. This will affect the sizes of the rectangle but it will not affect the array that contains the offset to the start of each line. Consequently your word wrapping will remain unchanged. To force a recalculation word wrapping based on a new destination rectangle, you must call TECalText().

Why doesn't my text change when I call TESetText?

TESetText() does not affect the text that is drawn in the destination rectangle on the screen. You must call InvalRect(), passing it the destination rectangle, for the text to be drawn.

How can I draw text with TETextBox() so that the background behind the text is maintained allowing me to draw over a picture?

The problem with trying to have TETextBox() draw over a picture or other type of background that you want to preserve is that TETextBox() calls EraseRect() before it draws. One solution around this problem is to use TETextBox() to draw the text offscreen and then use CopyBits() to transfer it to your window. If you use CopyBits() with the srcOr transfer mode, the background of your window will remain under the text. This method may slow things down somewhat, but it enables you to display the text in a number of different ways. Another solution is
to override the standard low-level rectangle function, StdRect(), immediately before you call TETextBox(), so that it does nothing. Then when you call TETextBox() the text is drawn but the area is not erased. After the text is drawn, restore the low-level bottleneck routine.

**How do I get TextEdit to display more than 32K of text?**

You cannot use TextEdit to display more than 32K of text properly. There is a TextEdit replacement, TE32k, which can handle more than 32K of text and has an interface equivalent to TextEdit. It does not handle styled text. TE32k is available from various download sites.

**Scrolling in TextEdit**

This section of the Text: TextEdit chapter answers FAQs about putting scroll bars in a TextEdit field. It covers how this is done and shows how you can get your scroll bars to update automatically when you scroll by dragging the cursor.

**How do I put scroll bars in my TextEdit field?**

TextEdit does not allow you to assign a scroll bar to a TextEdit field automatically, the way the List Manager allow you to assign lists to the TextEdit field. If you want a scroll bar to be attached to your TextEdit field, you must create the scroll bar and handle mouse clicks in it yourself.

The first step is to create a new scroll bar by calling NewControl(). The rectangle of the scroll bar, defined in the second parameter, should be just outside the viewRect rectangle of the TextEdit field. A vertical scroll bar should be the same height as the TextEdit field and have a width of 16 pixels.

```c
myTEScrollBar = NewControl( myTEWindow, &scrollBarRect, "\p", 
    TRUE, 0, 0, 0, scrollBarProc, OL);
```

mouseDown events in the scroll bar are handled the same way as with a scroll bar attached to a window. When the user clicks the mouse in the content area of the window, then FindControl() is called to determine which on part, if any, of the scroll bar was clicked. If the part was inPageUp or inPageDown (the area in the
scroll bar between the thumb and the scroll bar arrow buttons), then the TextEdit field should be scrolled by the number of pixels in the viewRect. If the part was inUpButton or inDownButton, then the TextEdit field should be scrolled by the height of the line. Determining the height of the line is another tricky issue. If the TextEdit field is an old type, single style TextEdit field then the height of a line is stored in the lineHeight field of TRec structure. Otherwise it must be extracted from the line height table. If the part was inThumb, then the TextEdit field should be scrolled according to the movement of the thumb. Once you have determined the number of pixels that the TextEdit field should be scrolled, call TEScroll() to do the actual scrolling.

Related FAQs

See also FAQ 6-21, “How do I put scroll bars in my window?”

See also FAQ 32-15, “How do I update the scroll bars in response to automatic scrolling caused by dragging outside the window?”

Related Topic

See also Chapter 6, “Dialogs and Controls.”

How do I update the scroll bars in response to automatic scrolling caused by dragging outside the window?

Automatic scrolling occurs when the user clicks the mouse in the viewRect of a TextEdit field and drags the mouse outside of it. If there is a scroll bar associated with the TextEdit field, you have no way of telling TextEdit to scroll the scroll bar in tandem with the scrolling text.

You can work around this problem by installing your own routine into the clickLoop field of the TRec structure. The routine pointed to by clickLoop is called continuously, as long as the mouse is held down in the viewRect. By default, clickLoop points to a routine that works well with TEAutoView(); it will automatically scroll the text in the TextEdit field. However, it doesn’t know about the scroll bars. You can install a routine to replace the default clickLoop, and handle the manipulation of scroll bars, by calling TESetClickLoop(). Don’t forget that you are replacing the routine that did the actual scrolling of text. So, along with manipulating the scroll bars, you will have to scroll the text yourself with TEScroll() inside your clickLoop.
The `clickLoop` routine is defined below.

```pascal
Boolean NewClickLoop()

To install it, call `TESetClickLoop()`

`TEClickLoopUPP `uppClickLoop;

`uppClickLoop = NewTEClickLoopProc(NewClickLoop);
`TESetClickLoop(uppClickLoop , myTE );
```

Related FAQ

See also FAQ 32-14, “How do I put scroll bars in my TextEdit field?”

### The TextEdit Cursor

This section of the Text: TextEdit chapter answers FAQs about manipulating the cursor in a TextEdit field. It covers how you can get your text caret to blink and how you can adjust its blinking rate. It also shows how you can set the position of the cursor and a selection range.

**How do I set the selection range in a TextEdit field?**

Some TextEdit functions work on a selected range of text in an edit text field. The clipboard routines `TECopy()` and `TECut()` and the style setting routine `TESetStyle()` are some examples. TextEdit automatically handles the selection of text by the user clicking and dragging in its text. Sometimes it is necessary for the application to define the selection range.

The range of selected text is specified by two fields in the `TERec` structure of a TextEdit field, `selStart` and `selEnd`. These two values can be manually changed or set with `TESetSelect()`. `TESetSelect()` removes the highlighting of the old selection range and highlights the new one. If `selEnd` is equal to `selStart`, the selection range is an insertion point and the caret is displayed. To move the caret to the beginning of the TextEdit field, call:

```pascal
TESetSelect(hTE, 0,0);
```
Sometimes you may want to adjust the selection range without having the newly selected text become highlighted. This can be accomplished by manually setting the selStart and setEnd fields of the TERec structure. If you want to call TESetSelect() to set the selection range but you want to avoid the highlighting, call TDEactivate() to render the edit record inactive before calling TESetSelect().

**How do I set the cursor position in a TE field?**

You can display the blinking cursor at a particular position in a TE field with the TESetSelect() call. The selStart and setEnd parameters should both be set to the new cursor position.

**Why won’t my TE caret blink?**

To make the TextEdit caret blink you must call TEIdle() each time WaitNextEvent() returns a null event. The blinking speed of the caret is determined by the value stored in the low memory global, CaretTime, at the address 0x02F4. The function, GetCaretTime() retrieves this value for you.

**How can I modify the caret blinking rate of the TextEditing insertion point caret?**

The blinking rate of the caret is determined by the value stored in the low memory global variable CaretTime. This value represents the number ticks, 1/60th second, between blinks. The Toolbox function GetCaretTime() returns this value. The universal headers define two macros that allow you to access this global variable directly. LMGetCaretTime() also returns the number of ticks between caret blinks. LMSetCaretTime() allows you to set this value and consequently adjust the blinking rate.

**Topic-Related FAQ**

See also FAQ 26-15, “Why doesn’t GetScrap() and PutScrap() from a TextEdit field or DialogCopy() from a dialog affect the desk scrap?”
On Macintoshes and PCs, the window is such an integral part of the interface that it is taken for granted. Yet, when the Macintosh was introduced over a decade ago, the concept of windows was one of the key elements that revolutionized how we work with computers. Being able to have multiple windows displayed on the screen, each one representing its own workspace, enables us view several distinct pieces of information at once. The coordination of the different windows is an important job that is handled by the Window Manager. Displaying, hiding, ordering, and updating properly, are just some of its functions. This chapter focuses on getting the most from the Window Manager. You'll learn how to force an update event, how to determine the location of your window in global coordinates, how to get from a WindowPtr to a WindowRecord, how to get the GrafPort of the desktop, how to copy the contents of one window to another, how to create non-rectangular windows, how to access the current update region, how to determine the height of a window's title bar, and more.
There are five subsections in this chapter:

- Definitions
- Displaying Windows
- Updating and Drawing on Windows
- Windows and Scroll Bars
- Windows and Your Code

Definitions

This section of the Windows chapter answers FAQs about some important terms that are discussed in the following sections. It compares the different types of coordinates systems and defines the WDEF.

What is the difference between local and global coordinates?

Global coordinates represent the array of pixels that exist on a screen. The coordinate \((0, 0)\) is always located at the upper right corner of the screen. Local coordinates of a window represent the area inside that window's boundaries, the coordinates of the windows graphics port. By default, \((0, 0)\) is in the upper left corner of the window's drawing space, the area under its title bar. By calling SetOrigin(), you can reset the coordinates that are located at this position. QuickDraw does all of its drawing relative to the point set as the origin of a window.
Figure 33-1: Local coordinates are based off of the origin of the window. Global coordinates are based off of the upper-left corner of the screen. The point located at local coordinate (50,50) in the window is located at global coordinate (150,150).

Related FAQs

- See also FAQ 33-4, “How do I find the position of my window on the screen?”
- See also FAQ 33-14, “Is the origin of a window the upper-left corner of the window, or the upper-left corner of the drawing area, below the title bar?”

What is a window definition procedure (‘WDEF’)?

A window definition procedure is a function called by the Window Manager that defines the appearance and behavior of a window. It is used to draw the window’s frame, determine what region of the window the cursor is in, draw the size box and zoom box of the window, move and resize the window, and calculate the window’s content and structure regions.
Displaying Windows

This section of the Windows chapter contains FAQs about how to manipulate windows on the screen. It covers how to alter a window’s position and size. It also deals with the organization of multiple windows in an application. As one window becomes active and moves to the front of other windows, the window that was on top now must get deactivated. This section also presents a simple method of putting a floating window in your application.

How do I set the appearance of a window?

The Window Manager supports nine different window types from which to choose. The fifth parameter in NewWindow() and the sixth parameter in NewCWindow() (usually referred to as theProc), accept a window definition ID. The standard ID values are:

- documentProc = 0; [standard document window, no zoom box]
- dBoxProc = 1; [alert box or modal dialog box]
- plainDBox = 2; [plain box]
- altDBoxProc = 3; [plain box with shadow]
- noGrowDocProc = 4; [movable window,]
- movableDBoxProc = 5; [movable modal dialog box]
- zoomDocProc = 8; [standard document window]
- zoomNoGrow = 12; [zoomable, nonresizable window]
- rDocProc = 16; [rounded-corner window]

If you create a ‘WIND’ resource with ResEdit or Resorcerer you can select the window type in the dialog used to define the window.

You can display a customized window type by creating your own window definition procedure (‘WDEF’).
How do I find the position of my window on the screen?

To get the position of the window, get the bounding box of the `strucRgn` field. This represents the rectangle containing the content region plus the window frame in global coordinates. It is located in `rgnBBox` field of the `strucRgn` field of the `WindowRecord` in global coordinates.

```
    rWindow = ((WindowPeek)pWindow)->strucRgn)->rgnBBox;
```

If the window is hidden, the `strucRgn` is empty. You can then find the position of a window's content region on the screen and subtract the window's frame and title bar height. To get the position of the content region, call `SetPort()` to make the window's GrafPort active. Then get the origin of the window from the `portRect` field of the window pointer and call `LocalToGlobal()` on that point. The global location of the window's origin represents the location of the window.

```
/*--------------------------*
 void CalcWindowContentPosition(
   WindowPtr pWindow,
   Point *windowPt )
{
    GrafPtr oldPort;
    SetPtCwindowPt, pWindow->portRect.left,
    pWindow->portRect.top);
    GetPort( &oldPort );
    LocalToGlobalCwindowPt);
    SetPort( oldPort );
}

Related FAQs

- See also FAQ 33-1, “What is the difference between local and global coordinates?”
- See also FAQ 33-14, “Is the origin of a window the upper-left corner of the window, or the upper-left corner of the drawing area, below the title bar?”

How do I get the current size of my window?

To get the size of the window, get the bounding box of the `strucRgn` field. This represents the rectangle containing the content region plus the window frame in
global coordinates. It is located in the rgnBBox field of the structRgn field of the WindowRecord.

\[ rWindow = (f((WindowPeek)pWindow)->strucRgn)->rgnBBox; \]

The width is the difference between right and left coordinates. The height is the difference between bottom and top coordinates.

If the window is hidden, the structRgn is empty. You can then get the size of the window's content region from the portRect field. You will have to add the height of the window's title bar. The code below determines the size of the content region.

```c
void GetWindowContentSize( WindowPtr pWindow,
                     short *width,
                     short *height )
{
    *height = pWindow->portRect.bottom - pWindow->portRect.top;
    *width = pWindow->portRect.right - pWindow->portRect.left;
}
```

**How can I get all of the open windows in my application? Is there a list of all open windows somewhere?**

There is a nextWindow field in the structure of each window that points to the next window in the window list. To traverse the entire list of windows, start at the top window, returned by frontWindow(), and move to the next window pointed to by the nextWindow field. The last window in the list will have a nextWindow value of nil.

nextWindow is a field in a window's WindowRecord structure. A pointer to a WindowRecord structure is called a WindowPeek. frontWindow() returns a WindowPtr. A WindowPtr is a pointer to a GrafPort, not to a WindowRecord. However, since the first field of a WindowRecord is the window's GrafPort, you can cast a WindowPtr to a WindowPeek. Now you can access the nextWindow field.

Hidden windows are also kept in the list.
```c
//F***************************************************************************
void TraverseWindowList()
{
    WindowPtr pTempWindow;

    pTempWindow = FrontWindow();
    while (pTempWindow)
    {
        DoYourWindowAction();
        pTempWindow = (WindowPtr)
        ((WindowPeek)pTempWindow)->nextWindow;
    }
}
```

**Related FAQ**

See also FAQ 33-10, “How do I open a new window behind an already open window in my application?”

**How do I center a window on the screen?**

To center a window in a screen you need to get the dimensions of the screen and the window first. The dimensions of the screen are stored in the gDRect field of the screen's GDevice structure. You can get the GDevice structure of the main screen by calling GetMainDevice(). To get the GDevice structure of other screens in your system, you will have to traverse the device list. If the window is visible, you get the actual size of the window from the bounding box of the window’s strucRgn. If the window is not visible, you can use the window’s portRect field (which actually gives you the rect of the content region — that’s pretty close). Then you can get the coordinate of the upper left corner of the centered window by:

```c
OffsetRect( &windRect, screenRect.left - windRect.left,
            screenRect.top - windRect.top);
OffsetRect( &windRect, (screenRect.right - windRect.right)/2,
            (screenRect.bottom - windRect.bottom)/2);
```

Now that you have the new window rectangle call MoveWindow().
**How do I get the top window of my application?**

`frontWindow()` returns a pointer to the frontmost window in your application. If your window is in the background and windows from another application are on top, `frontWindow()` still returns the frontmost window of your application.

**What effect does hiding a window have on the window’s position in the application’s window list?**

The Window Manager maintains a linked list of all windows in an application through the `nextWindow` field of each window’s `WindowRecord` structure. The active window is the top of the list, the last window’s `nextWindow` field is 0.

If there is more than one window in an application, and the top window is hidden with `HideWindow()`, it is moved to the second position in the list and the next visible window is made active and moved to the top of the window list. When the hidden window is made visible with `ShowWindow()` it remains in the second position. If a non-active window is hidden and then made visible with `ShowWindow()` it remains in its original position.

`SelectWindow()` makes a window visible and also moves it to the top of the list.

`ShowHide()` allows you to show or hide a window. It does not change the highlighting or ordering of windows and does not cause any activate events to be generated.

**How do I open a new window behind an already open window in my application?**

Both `GetNewWindow()` and `NewWindow()` have a parameter, of type `WindowPtr`, that allows you to specify where the new window will be positioned for your application. This parameter is the third parameter in `GetNewWindow()` and the sixth parameter in `NewWindow()`. If you put a `WindowPtr` in the behind parameter, the new window will appear behind that window. If you pass a -1 in the behind parameter, the window becomes the frontmost window. If you pass a 0 in the behind parameter, the window goes to the back of the window list.

If you create a new window behind another window but set the visible flag (the fourth parameter in `NewWindow()` and a resource flag in the ‘WIND’ resource
retrieved by GetNewWindow() to false, calling ShowWindow() will open the window behind the specified window. Do not call SelectWindow() since it will both show and activate the window.

Related FAQ

See also FAQ 33-6, “How can I get all of the open windows in my application? Is there a list of all open windows somewhere?”

**How do I create a floating window or palette, a window that always stays on top of my windows, in my program?**

A floating window is a window that always remains on top of all other windows, except for other floating windows. When a normal document window is active, and appears active, the floating window is still the top window.

To handle a floating window in your program you must: open new windows under the floating window, replace SelectWindow() when the user clicks in the content of a document window with calls that make the window appear active but do not make it the front window, replace DragWindow() with calls that move the window but do not overwrite the floating window, and manually activate and deactivate your top document window in response to suspend and resume events.

The majority of the work will involve the handling of mouse-down events. When the user clicks in the content region, instead of calling SelectWindow(), call SendBehind() to move the window directly behind the floating window. This will position the window properly, but it will not make the window look active. To do this call HilitWindow() with the second parameter set to true. You must also deactivate the document window that was previously active. To do this call HilitWindow() with the second parameter set to false. You will have to keep track of the top non-floating window to deactivate.

A more complicated case to handle is responding to the dragging of a window. When a window is dragged, a dotted line frame of the window gets moved around the screen. This should not appear over the floating window. To prevent this you will have to create a clipping region for the desktop that does not include the area over the floating window. You can get the Window Manager port by calling GetWMgrPort(). Then call DragGrayRgn() to drag the dotted outline of the rectangle. When the dragging is completed you must call MoveWindow() to move the window to the new position.
The following sample program puts up three windows, two document windows and a small floating window. The program is exited upon the hitting of any key and assumes that the three windows are always open throughout the program.

```
/* InitToolbox */
void InitToolboxCvoid);
void InitProg(void);
void EventLoop( void );
void HandleEvent( EventRecord *eventPtr);
void HandleMouseDown( EventRecord *eventPtr);
void DoUpdate( EventRecord *eventPtr);
void DoOSEvent( EventRecord *eventPtr);

WindowPtr pWindow1, pWindow2, // Document windows
      floatWindow, // the floating window
activeDocWindow = nil; // the top document window

Boolean gDone = FALSE;

void main()
{
    InitToolbox();
    InitProg();

    EventLoop();
}

InitToolbox()
{
    InitGraf(&qd.thePort);
    InitFonts();
    FlushEvents(everyEvent, 0);
    InitWindows();
    InitMenus();
    InitDialogs(OL);
    InitCursor();
    MaxApplZone();
}
```
/**************************
void InitProg()
{
    Rect r1 = {40, 40, 440, 340};
    Rect r2 = {60, 60, 460, 360};
    Rect rfloat = {80, 80, 180, 280};

    pWindow1 = NewCWindow( nil, &r1, "\pWindow 1",
        false, noGrowDocProc, 0,
        true, nil );
    pWindow2 = NewCWindow( nil, &r2, "\pWindow 2",
        false, noGrowDocProc, 0,
        true, nil );
    floatWindow = NewCWindow( nil, &rfloat, "\pFloater",
        true, noGrowDocProc, (WindowPtr)-1,
        true, nil );

    ShowWindow( pWindow1);
    ShowWindow( pWindow2);

    ShowWindow( floatWindow);
    SelectWindow( floatWindow );
    HiliteWindow( pWindow1, TRUE );
    activeDocWindow = pWindow1;
}

/**************************
void EventLoop( void )
{
    EventRecord event;

    gDone = FALSE;
    while ( gDone == FALSE )
    {
        if ( WaitNextEvent( everyEvent, &event, 30, 0L) )
        {
            HandleEvent( &event );
        }
    }
}
```c
void HandleEvent( EventRecord *eventPtr )
{
    switch ( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;

        case updateEvt:
            DoUpdate( eventPtr );
            break;

        case osEvt:
            DoOSEvent( eventPtr );
            break;

        case keyDown:
            gDone = TRUE;
            break;

        default:
            break;
    }
}

void DoOSEvent( EventRecord *eventPtr )
{
    // test for suspend or resume messages
    if (eventPtr->message & 0x01000000 )
    {
        // manually activate appearance of top doc
        // window on resume event, deactivate on suspend
        if ( eventPtr->message & 0x00000001 )
            HiliteWindow( activeDocWindow, TRUE );
        else
            HiliteWindow( activeDocWindow, FALSE );
    }
```
```c
//**************************
void DoUpdate( EventRecord *eventPtr )
{
    WindowPtr window;
    GrafPtr oldPort;
    Rect r = { 20,20, 300, 300};
    int i;

    GetPort( &oldPort );
    window = (WindowPtr)eventPtr->message;
    SetPort( window );
    BeginUpdate( window );
    for ( i = 1; i< 50; i++)
    {
        InvertRect( &r );
        InsetRect(&r, 2, 2);
    }

    EndUpdate( window );
    SetPort( oldPort );
}

//**************************
void HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr window;
    GrafPtr oldPort, windowManagerPort;
    RgnHandle oldclip, diffclip, dragRegion,
               windowContentRegion;
    long dragResult;
    short thePart;
    short newHorizontalWindowPosition,
            newVerticalWindowPosition;
    short horizontalOffset, verticalOffset;

    thePart = FindWindow( eventPtr->where, &window );
    switch ( thePart )
    {
    case inContent:
        if ( window !=floatWindow)
        {
```
SendBehind( window, floatWindow );
if (window != activeDocWindow)
{
    HiliteWindow( activeDocWindow, FALSE );
    HiliteWindow( window, TRUE );
    activeDocWindow = window;
}
break;

case idDrag:
    if ( window == floatWindow )
    {
        DragWindow( window, eventPtr->where, &qd.screenBits.bounds );
        break;
    }
    else
    {
        // a document window
        if (window != activeDocWindow)
        {
            HiliteWindow( activeDocWindow, FALSE );
            HiliteWindow( window, TRUE );
            activeDocWindow = window;
        }
    }
oldclip = NewRgn();
diffclip = NewRgn();
dragRegion = NewRgn();
CopyRgn(((WindowPeek)window)->strucRgn, dragRegion);

GetPort(&oldPort);
GetWMgrPort(&windowManagerPort);
SetPort(windowManagerPort);

GetClip(oldclip);
DiffRgn(GetGrayRgn(), ((WindowPeek)floatWindow)->strucRgn, diffclip);
CHAPTER 33: Windows

SetClip( diffclip );

    dragResult = DragGrayRgn(dragRegion,
            eventPtr->where,
            &qd.screenBits.bounds,
            &qd.screenBits.bounds,
            noConstraint, nil);

SetClip(oldclip);
SetPort(oldPort);

// move and redraw the window
if (dragResult != 0)
{
    horizontalOffset = dragResult & 0xFFF;
    verticalOffset = dragResult >> 16;

    windowContentRegion =
            ((WindowPeek)window)
                ->contRgn;
    newHorizontalWindowPosition =
            (**windowContentRegion).
                rgnBBox.left + horizontalOffset;
    newVerticalWindowPosition =
            (**windowContentRegion).
                rgnBBox.top + verticalOffset;

    MoveWindow(window,
                newHorizontalWindowPosition,
                newVerticalWindowPosition, false);
}

SendBehind( window, floatWindow );

DisposeRgn( dragRegion );
DisposeRgn( oldclip );
DisposeRgn( diffclip );
break;

default:
    break;
}
}
How can I tell when a window becomes activated or deactivated?

When a window becomes activated or deactivated, the Event Manager posts an activateEvt. The modifiers field of the event record defines if the activate event is signalling that the window is becoming activated or deactivated. Do a bitwise AND between the modifiers field and the activeFlag.

```c
void HandleEvent( EventRecord *eventPtr )
{
    Boolean becomingActive;

    switch ( eventPtr->what )
    {
        case activateEvt:
            becomingActive = ( (eventPtr->modifiers &
                                activeFlag) != 0 );
            if (becomingActive)
            {
                // window is becoming active
            }
            else
            {
                // window is becoming deactive
            }
            break;
    }
}
```

Is it possible to have windows of shapes other than the standard rectangular shape?

Yes, by creating a customized window definition procedure, you can create non-rectangular windows. The System window definition procedure gets passed messages to calculate the contRgn, content region, and the strucRgn, content region plus the frame, and to draw the window parts. This allows you to define the shape of the window.
Updating and Drawing On Windows

A window does not do us any good if it does not display any data. This section answers FAQs about updating the contents of a window. It covers how to set up a window for drawing and even tells you how to get the graphics port of the most important window of them all, your Desktop.

Is the origin of a window the upper-left corner of the window, or the upper-left corner of the drawing area, below the title bar?

A window's origin is located in the upper-left corner of the window's content region. This region is below and does not include the window's title bar.

Related FAQs

- See also FAQ 33-1, "What is the difference between local and global coordinates?"
- See also FAQ 33-4, "How do I find the position of my window on the screen?"

Related Topic

- See also Chapter 23, "QuickDraw: Drawing."

How can I force my window to redraw when I want to make a change?

To redraw a window you should force an update event to be generated. You do this by calling InvalRect() or InvalRgn() with the window to be redrawn set as the current port. The rectangle or region passed as a parameter of the Inval routine defines the area of the window that is redrawn (clipped by BeginUpdate() before drawing).
How can I set up custom colors in my window?

The colors of the different parts of a window are determined by a color table. Each ColorSpec entry in the color table determines the colors for a single part. You can control the colors used by a window by assigning a window color table, 'wctb' resource, to the window. There are two ways to accomplish this. The first is to define a 'wctb' resource with a resource editor and assign it to a window by giving it the same id. The second is to assign a color table to a window by calling SetWinColor().

When a window is created, the Window Manager looks for a 'wctb' resource with the same id as the 'WIND' resource used for the window. If it finds one, it creates a color table based on the 'wctb' resource information for the window. If it does not find one, it uses the default System window color table that is loaded into the application heap when the application starts up.

The value field of each ColorSpec record specifies a constant that determines which part of the window the color controls. Below is a list of the different possible values and the corresponding part of the window.

- wContentColor = 0 content region background
- wframeColor = 1 window outline
- wTextColor = 2 window title and button text
- wHiliteColor = 3 reserved
- wTitleBarColor = 4 reserved
- wHiliteColorLight = 5 lightest stripes in title bar and lightest dimmed text
- wHiliteColorDark = 6 darkest stripes in title bar and darkest dimmed text
- wTitleBarLight = 7 lightest parts of title bar background
- wTitleBarDark = 8 darkest parts of title bar background
- wDialogLight = 9 lightest element of dialog box frame
Related FAQs

- See also FAQ 6-6, “How do I create a color dialog?”
- See also FAQ 6-30, “How can I set the fonts and colors of my dialog items?”

Related Topic

- See also Chapter 2, “Color.”

How do I get the GrafPort of the desktop?

Call GetCWMgrPort() or GetWMgrPort() to get a pointer to the Window Manager's GrafPort, the GrafPort of the Desktop. GetCWMgrPort() is called for systems which support Color QuickDraw. GetWMgrPort() is called for systems with the original monochrome QuickDraw.

```c
//********************************************************************************
void GetDeskTopGrafPort( GrafPtr *wMgrPort )
{
    OSErr err;
    long response;

    err = Gestalt( gestaltQuickdrawVersion, &response);
    if ( (err == noErr) && (response >= gestalt8BitQD) )
        GetCWMgrPort((CGrafPtr *)wMgrPort);
    else
        GetWMgrPort(wMgrPort);
}
```
Related FAQs

- See also FAQ 12-11, "How can I get the image of a screen (as in a screen capture)?"
- See also FAQ 23-38, "Is there a way to draw outside a window on the Desktop?"

Related Topic

- See also Chapter 23, "QuickDraw: Drawing."

**Why can I only draw in the eight basic colors in my window?**

You are probably not using a color window. The calls, `NewWindow()` and `GetNewWindow()`, create a window in a basic graphics port. This will limit you to the eight colors in the original QuickDraw, leading you to believe that you do in fact have a color graphics port. You can use `NewCWindow()`, `GetNewCWindow()`, `NewCDialog()`, and `GetNewCDialog()` to create color windows and dialogs.

Related FAQ

- See also FAQ 6-6, "How do I create a color dialog?"

Related Topic

- See also Chapter 2, "Color."

**How can I copy the contents of one window to another?**

You can copy the contents of one window to another with the `CopyBits()` command. Specify the `portBits` or the `portPixMap` fields of the two windows as the source and destination fields in `CopyBits()`. The following code copies the entire contents of `srcWindow` into `destWindow`. 
When I call SelectWindow() to activate a window I cannot always draw to it. My drawing affects the window that was previously the active window even though the new one is now on top and active. Why?

SelectWindow() activates a window, it does not set the window that was just selected to be the active graphics port. You must call SetPort() yourself.

How can I tell what part of my window needs to be redrawn in response to an update event?

In response to an update event being posted, your program will call BeginUpdate(). BeginUpdate() saves a copy of the window’s visible region and then sets the visRgn field of the window to the union of the original visRgn and the updateRgn. EndUpdate() restores the original visRgn value.

Related Topic

See also Chapter 23, “QuickDraw: Drawing.”

If there are multiple windows in my application, how do I select which windows to draw to?

All QuickDraw activities are performed on the current port. To draw on a window, it must be set to be the current port. The call SetPort() allows you to set the current port and draw to the specified window. It does not make the window the top-most window. SelectWindow() sets the current port and activates the window.

How can I put a color background in my window?

The background color of a window can be set by creating a window color table, ‘wctb’ resource, for the window. The first entry in the color table represents the content region background color. You can define a ‘wctb’ resource with a resource
editor and assign it to a window by giving it the same ID. You can also assign a ‘wctb’ to a window by calling SetWinColor().

Resorcerer allows you specify the color of the content region background in the window resource’s definition.

Windows and Scroll Bars

This section of the Windows chapter answers FAQs on putting scroll bars into a window. It covers how to get the scroll bars to draw properly and how to display the grow box in the lower right of your screen. It provides a sample that puts a scroll bar in an application and handles all of the necessary processing.

How do I draw the grow box in my window?

To draw the grow box in a window, call DrawGrowIcon() after receiving an update or activate event. DrawGrowIcon() automatically draws the appropriate image based on the window’s type and style. If the window is active and of the proper type, DrawGrowIcon() draws the size box in the lower right corner of the window and draws the outline of the scroll bars. If the window is inactive, DrawGrowIcon() draws the outline of the scroll bars and the outline of the size box and erases the image of the size box.

DrawGrowIcon() does not erase the scroll bar areas. This means you will have to erase these areas yourself to remove the scroll bars, the scroll bar outlines, and the size box when you resize the window.

Figure 33-2: The grow box is located at the lower right corner of the window.
**How can my application determine a window title bar’s height in pixels?**

*Inside Macintosh: Macintosh Toolbox Essentials* states that in the Roman script system, the height of the standard document title bar is 20 pixels high. This is incorrect, it is actually 19 pixels high.

You can not always count on the height of the title bar to be the same on all systems. Some international versions may have a different size. To determine the height of the window title bar, find the difference between the top of the rectangle of the window’s content area, which is below the title bar, and the top of its structure region. Make sure that the window is visible when you check these values. The structure and content regions are not valid if the window is not visible. The following routine returns the title bar height of a specified window.

```c
//******************
short GetTitleBarHeight( WindowPtr myWindow) 
{
    short titleBarHeight;
    titleBarHeight =
        ( *( ((WindowPeek)myWindow)->contRgn ) )->rgnBBox.top 
        - ( *( ((WindowPeek)myWindow)->strucRgn ) )->rgnBBox.top;
    return( titleBarHeight );
}
```

If you need the title bar height before you display the window on the screen, then show it off screen with the `ShowHide()` command, and check the regions.

**Why don’t my scroll bars get refreshed when my window comes to the front?**

Scroll bars and other controls are not automatically redrawn when a portion of your window needs to be updated. You have to draw them manually. To do this call the `UpdateControls()` call in response to an update event for your window.
How do I draw a Macintosh grow box without the scroll bars?

To draw the window's size box without drawing the lines that delimit the scroll bars, set the clipping region of the window's graphics port to be a 15 pixel by 15 pixel rectangle in the lower right corner of the window and then call DrawGrowIcon().

The sample below draws the size box of the window passed to the function without drawing the scroll bar outline.

```c
//***************
#define kGrowBoxWidth 15
void MyDrawGrowIcon(WindowPtr window)
{
    GrafPtr savePort;
    RgnHandle saveRgn;
    Rect growRect;

    GetPort(&savePort);
    SetPort(window);
    growRect = window->portRect;
    growRect.top = growRect.bottom - kGrowBoxWidth;
    growRect.left = growRect.right - kGrowBoxWidth;

    saveRgn = NewRgn();
    GetClip(saveRgn);
    ClipRect(&growRect);

    DrawGrowIcon(window);

    SetClip(saveRgn);
    DisposeRgn(saveRgn);
    SetPort(savePort);
}
```

How do I prevent scroll bars from getting drawn on?

To prevent the scroll bars from getting drawn on, you must manually set the clipping area so that it does not include the area of the scroll bars. The vertical scroll bar has a width of 16 pixels and a height equal to the height of the window's
portRect. The horizontal scroll bar has a height of 16 pixels and a width equal to the width of the window's portRect. Don't forget to expand the clipping region in order to draw the size box if necessary.

The code below creates a rectangle of the window's content area minus the scroll bar areas.

```c
#define kScrollBarWidth 15

windowRect = window->portRect;
windowRect.right -= kScrollBarWidth;
windowRect.bottom -= kScrollBarWidth;
ClipRect( &windowRect );
```

Windows and Your Code

This section of the Windows chapter answers FAQs about putting windows in your code. It covers a few important fields in a Windows code structure, the WindowRecord.

I want to access fields in a window's WindowRecord structure but all I have are WindowPtrs which are pointers to GrafPorts. How can I access a window's WindowRecord from a WindowPtr?

A pointer to a WindowRecord structure is called a WindowPeek. A WindowPtr is a pointer to a GrafPort, not to a WindowRecord. However, since the first field of a WindowRecord is the window's GrafPort, you can cast a WindowPtr to a WindowPeek and access the fields of the window's WindowRecord.

```c
typedef WindowRecord FWindowPeek;
struct WindowRecord
{
    GrapPort  port;
    short windowKind;
};
To access the `windowKind` field from a `WindowPtr`:

```c
(WindowPtr)((WindowPeek)pTempWindow)->windowKind
```

### Can I store my own data in the refCon field of a window or dialog?

Yes, the refCon field of a dialog or window is available for an application to store data in. Two routines, `SetWRefCon()` and `GetWRefCon()` set and get the data in this field.

Related Topic

See also Chapter 6, "Dialogs and Controls."

### How can I attach data to a window structure for my own use?

There are two ways to do this. The first is to use the refCon field of the WindowRecord structure which Apple provided for your own use. This field is a long integer. If you need to store more data than that, create a structure and store a handle to it. You can use `SetWRefCon()` and `GetWRefCon()` to set and get this field.

The second method is to "piggyback" data onto your window. To implement this method, a structure is created whose first field is a WindowRecord. Subsequent fields in the structure contain the other pieces of information that you want associated with that window.

```c
typedef structure
{
    WindowRecord theWindow;
    Handle myFirstHandle;
    Handle mySecondHandle;
    Handle myThirdHandle;
}
MyWindowRecord, FMyWindowPeek;
```
Next we have the Window Manager store the window record of a window into this structure. We do this by making use of the wStorage parameter in the window creation routines. This is the second parameter of GetNewWindow() and GetNewCWindow() and the first parameter of NewWindow() and NewCWindow(), the parameter that always made you ask, “Why do I need that?”

```c
Ptr wStorage;
WindowPtr mySpecialWindow;

wStorage = NewPtr( sizeof(MyWindowRecord) );
mySpecialWindow = GetNewCWindow( 128, wStorage, -1 );
```

You can pass mySpecialWindow to any function that expects a WindowPtr. You can also use this pointer to access your private data fields by casting mySpecialWindow to your structure type. For example to access the myThirdHandle field you would do the following:

```c
((MyWindowPeek)mySpecialWindow)->myThirdHandle
```

**Topic-Related FAQs**

- See also FAQ 2-10, “What RGB value does the System use for dimmed buttons, menus, and window titles?”
- See also FAQ 2-24, “How can I get the palette of a window?”
- See also FAQ 2-25, “How do I assign a color palette to a window?”
- See also FAQ 2-26, “How do I set up a default palette for all windows in my application?”
- See also FAQ 2-27, “Does a palette that is attached to a window get disposed of when the window is closed?”
- See also FAQ 2-28, “Why is my window not immediately affected when I change my palette?”
- See also FAQ 2-29, “How can I get a picture that does not use the default palette to display properly in a window?”
- See also FAQ 6-22, “What is the size of the scroll bars?”
See also FAQ 7-18, "How can I tell what part of my window the user clicked in?"

See also FAQ 7-23, "When my program receives an activate event is issued, how can I tell if the window is activated or deactivated?"

See also FAQ 12-11, "How can I get the image of a screen (as in a screen capture)?"

See also FAQ 12-15, "If my window overlaps multiple screens with different depths, how can I draw the contents of the window properly on all screens?"

See also FAQ 12-16, "How can I determine which screens my window overlaps and which screen the majority of my window exists on?"

See also FAQ 16-29, "How can I draw to a window while a menu is pulled down obscuring the window?"

See also FAQ 23-16, "Why does my drawing trash the screen whenever I draw in response to an update event for a window that is not the frontmost window?"

See also FAQ 23-36, "How do I implement a ZoomRect (a thin gray rectangle that 'zooms out' when a window opens or closes, as when you open one of the finder's directory windows)?"

See also FAQ 24-10, "How do I create a picture of the contents of a window?"
Using the Companion CD-ROM

The CD-ROM that's packaged with the book contains a folder named *Mac Programming FAQs*. In this folder you'll find three main folders:

- **FAQ Code Projects**: The FAQ Code Projects folder contains CodeWarrior and Symantec projects that enable you to build applications based on the longer — and more interesting — sample code in the book. The contents of the FAQ Code Project folder include subfolders that represent different chapters in the book. The contents of each chapter subfolder include other subfolders that contain the individual projects. Each project includes the complete source, resource, and project files that are necessary to build executables, as well as a pre-built executable.

- **FAQ Code Snippets**: The contents of the FAQ Code Snippets folder include 27 SimpleText files that contain sample code from the book. Each file represents a different chapter; the code is organized by the associated FAQ.

- **FAQ Database**: The FAQ Database folder contains a database of the FAQs in the book and the FAQ DB Viewer program, which allows you to view the contents of the database. The database files are stored in a folder named *Tables Folder*. 
The FAQ DB Viewer program enables you to view all of the questions and answers in the FAQ database. The true benefit of the program is that it enables you to search FAQs by keyword or by topic.

**Starting the FAQ DB Viewer Program**

To run the FAQ DB Viewer program, follow these steps:

1. Place the CD-ROM in the CD drive.
2. Double-click the CD icon. The Mac Programming FAQs folder appears.
3. Double-click the folder.
4. Double-click the FAQ Database folder.
5. Double-click the FAQ DB Viewer icon. The Main FAQ appears, displaying the information from the current FAQ, FAQ 1-1, as shown in Figure A-1.

![FAQ Screen](image)

**Figure A-1:** The Main FAQ Screen displays the number, question, answer, topic, and subtopic of a FAQ. At the bottom left of the screen are the navigation buttons. At the bottom right of the screen are the current FAQ position and total view FAQ counts.
The Main FAQ screen is used to display the information of the current FAQ. The screen displays the FAQ's number, question, answer, topic, and subtopic. The screen also contains information about the current view of FAQs. A view is a collection of FAQs; it can either contain all of the FAQs in the database or a subset of them.

At the lower right of the screen are two numbers, the position of the current FAQ in the current view, and the total number of FAQs in the current view. At the lower left of the screen are the four navigation buttons that allow you to move to the first FAQ, the FAQ immediately before the current FAQ, the FAQ immediately after the current FAQ, and the last FAQ in the current view.

The FAQ menu and the navigation buttons at the bottom of the main screen allow you to travel through the FAQs in the current view.

Menus

The database program includes three menus:

- File menu
- FAQ menu
- View menu

File Menu

The File menu contains only one menu item, Quit, which you choose to exit the program.

FAQ Menu

The FAQ menu contains four menu items that you use to navigate through the current view of FAQs. They are:

- First FAQ
- Previous FAQ
- Next FAQ
- Last FAQ

The functions associated with these four menu items can also be executed from the navigation buttons at the bottom of the Main FAQ screen.
View Menu

The View menu contains four menu items:

- **Search FAQs**: This selection brings up the Search FAQs screen, which allows you to search the questions and the answers for specified text and selected FAQs of a specified topic and subtopic. You can select the matching FAQs to create a new view.

- **View FAQs**: This selection brings up the View FAQs screen, and displays all of the FAQs in the current view. When you select one of these FAQs, it becomes the current FAQ in the Main FAQ screen.

- **Go To FAQs**: This selection brings up the Go To FAQ Screen (see Figure A-2), which enables you to specify a FAQ by its number. If the FAQ appears in the current view, it will become the current FAQ displayed in the Main FAQ screen.

- **View Related FAQs**: This selection brings up the View FAQ screen and displays any related FAQs of the current FAQ.

![Go to FAQ Number](Figure A-2: The Go To FAQ Screen contains two text edit fields in which the two components of the FAQ number are entered. The first field represents the chapter number. The second field represents the FAQ position in that chapter.)

The Go To FAQ screen allows you to specify the FAQ number of a FAQ in the current view and make it the current FAQ. The first text field is the chapter number and the second text field is the FAQ number inside the chapter. These FAQ numbers are the same as in the book. The FAQ must be in the current view.

Using the FAQ DB Viewer

You can create different views by choosing Search FAQs from the View menu and then displaying the Search FAQs screen (see Figure A-3), which searches for FAQs based upon text and topics you enter.
The Search FAQs screen allows you to specify the chapter and subsection of a FAQ along with text to search for in the question or answer.

**Conducting a Search**

To search for a specific entry, enter a text string in the Search String field and select the Question option. To search for an answer, select the Answer option. When you type a string in the Search String field and you specify a chapter, the program will only search for those FAQs that are included in that chapter. After you enter the selection criteria, click the Do Search button. All the FAQs that match the selection criteria will appear in the list window on the left. You can then make these FAQs the current view by clicking the Select FAQs button.

To create a view that contains all of the FAQs in the database, choose <ALL> from the Chapter pull-down menu and deselect the Question and Answer options. Click the Do Search button to displays all of the FAQs. Click the Select FAQs button to load all of the FAQs into the current view.
You can also select the related FAQs of the current FAQ by choosing View Related FAQs from the View menu, which displays the View FAQs Screen that's shown in Figure A-4.

![View FAQs Screen](image)

**Figure A-4:** The View FAQs Screen is used to display FAQs in the current view when called by the View FAQs command and related FAQs of the current FAQ when called by the View Related FAQs command.

The View FAQs screen displays the FAQs in the current view. You can make any one of these FAQs the current FAQ by selecting it and then pressing the Go To button. The program will then return to the Main FAQ screen and display the selected FAQ.

### Using the FAQ Code Projects

To open the sample code projects, follow these steps:

1. Place CD-ROM in the CD drive.
2. Double-click the CD icon. The Mac Programming FAQs folder appears.
3. Double-click the folder.
5. Open the folder of the chapter whose projects you want to access and then open the folder of the project itself.
6. You can run the pre-built application by double-clicking its icon, or you can open the project by double-clicking the project icon.

Each sample project includes the complete source, resource, and project files that are necessary to build executables.

Using the FAQ Code Snippets

To view the sample code snippets, follow these steps:

1. Place CD-ROM in the CD drive.
2. Double-click the CD icon. The Mac Programming FAQs folder appears.
3. Double-click the folder.
5. Double-click the file icon of the chapter whose code you want to see. This action starts the Simple Text application. You can also view code files with any text editor.

Each file represents a different chapter. The code is organized by the associated FAQ.
This appendix lists the Apple-defined Gestalt Manager selector codes. We've attempted to describe the formats of their responses and list the constants that are defined for their return values.

You pass a selector code when you call Gestalt to specify the kind of information you need. Apple defines two distinct kinds of selector codes: environmental selectors, which supply information you can use to control the behavior of your application, and informational selectors, which supply information you can't use to determine what hardware or software features are available.

The selector code constants use a set of suffixes that indicate what format the response value will take. Selectors with the suffix Attr, for example, return a 32-bit response value in which the individual bits represent specific attributes. The constants listed for these response values represent bit numbers. See FAQ 11-2, "How do I use Gestalt to get information?" for more information.
Environmental Selectors

The following sections list the environmental selector codes.

Addressing Mode Attributes

The gestalt32BitAddressing attribute indicates that the machine started up with 32-bit addressing. The gestalt32BitSysZone attribute indicates that the system heap has 32-bit clean block headers (regardless of the type of addressing the machine started up in). See the book *Inside Macintosh: Memory* for more information about 32-bit addressing.

\[
\text{gestaltAddressingModeAttr} = \text{'addr'};
\]

- [booted in 32-bit mode]
  \[
  \text{gestalt32BitAddressing} = 0;
  \]
- [32-bit compatible system zone]
  \[
  \text{gestalt32BitSysZone} = 1;
  \]
- [machine is 32-bit capable]
  \[
  \text{gestalt32BitCapable} = 2;
  \]

Alias Manager Attributes

\[
\text{gestaltAliasMgrAttr} = \text{'alis'};
\]

- [Alias Manager is present]
  \[
  \text{gestaltAliasMgrPresent} = 0;
  \]
- [Alias Manager knows about remote AppleTalk]
  \[
  \text{gestaltAliasMgrSupportsRemoteAppleTalk} = 1;
  \]

Apple Events Attribute

\[
\text{gestaltAppleEventsAttr} = \text{'evnt'};
\]

- [Apple events available]
  \[
  \text{gestaltAppleEventsPresent} = 0;
  \]
- [OSL in system]
  \[
  \text{gestaltOSLInSystem} = 2;
  \]

The version number of the AppleTalk driver (in particular, the .MPP driver) currently installed. The version number is placed into the low-order byte of the result; ignore the three high-order bytes. If an AppleTalk driver is not currently open, the response parameter is 0.
gestaltAppleTalkVersion = 'atlk';

The version number of the AppleTalk driver, in the format introduced with AppleTalk
version 56. The version is stored in the high 3 bytes of the return value.

Byte 3 contains the major revision number, byte 2 contains the minor revision number, and
byte 1 contains a constant that represents the release stage.

For example, if you call Gestalt with the 'atkv' selector when AppleTalk version 57 is loaded,
you receive the long integer response value $39008000. Byte 0 always contains 0.

gestaltATalkVersion = 'atkv'

{gestaltATalkVersion release stage constants}
development = $20;
alpha = $40;
beta = $60;
final = $80;
release = $80;

The version of A/UX if it is currently executing. The result is placed into the low-order word
of the response parameter. If A/UX is not executing, Gestalt returns gestaltUnknownErr.

gestaltAUXVersion = 'a/ux';

Code Fragment Manager Attributes

gestaltCFMAattr = 'cfrg';

{Code Fragment Manager present}
gestaltCFMPresent = 0;

CloseView Attributes

gestaltCloseViewAttr = 'BSDa';

gestaltCloseViewEnabled = 0;
gestaltCloseViewDisplayMgrFriendly = 1;

Color Picker

gestaltColorPickerVersion = 'cpkr'
Component Manager

getstaltComponentMgr = 'cpnt';

Connection Manager Attributes

The gestaltConnMgrCMSearchFix bit flag indicates that the fix is present that allows the CMAddSearch routine to work over the mAttn channel.

gestaltConnMgrAttr = 'conn';

{Connection Manager present}
gestaltConnMgrPresent = 0;
{CMAddSearch fix present}
gestaltConnMgrCMSearchFix = 1;
{has CMGetErrorString}
gestaltConnMgrErrorString = 2;
{has CMNewIOPB, CMDisposeIOPB,}
{CMPBRead, CMPBWrite, and CMPBIOKill}
gestaltConnMgrMultiAsyncIO = 3;

Communications Resource Manager Attributes

gestaltCRMAAttr = 'crm ';

{Communication Resource Manager present}
gestaltCRMPresent = 0;
{fix for persistent tools}
gestaltCRMPersistentFix = 1;
{tool resource calls available}
gestaltCRMToolRsrcCalls = 2;

The version number of the Communications Toolbox (in the low-order word of the return value).

gestaltCTBVersion = 'ctbv';

Data Access Manager Attribute

gestaltDBAccessMgrAttr = 'dbac';
gestaltDBAccessMgrPresent = 0;
Appendix B: Gestalt Manager Selector Codes

Dictionary Manager Attributes

\[
\text{gestaltDictionaryMgrAttr} = \text{'dict'};
\]
\[
\text{gestaltDictionaryMgrPresent} = 0;
\]

Display Manager Attributes

\[
\text{gestaltDisplayMgrAttr} = \text{'dply'};
\]
\[
\text{gestaltDisplayMgrPresent} = 0;
\]

Dialog Manager Extensions Attributes

If this flag bit is TRUE, then the Dialog Manager extensions included in System 7 are available. See the book *Inside Macintosh: Macintosh Toolbox Essentials* for details about the Dialog Manager.

\[
\text{gestaltDITLExtAttr} = \text{'ditl'};
\]
\[
\text{gestaltDITLExtPresent} = 0;
\]

Drag Manager Attributes

\[
\text{gestaltDragMgrAttr} = \text{'drag'};
\]
\[
\text{gestaltDragMgrPresent} = 0;
\]

Easy Access Attributes

\[
\text{gestaltEasyAccessAttr} = \text{'easy'};
\]
\[
\text{gestaltEasyAccessOff} = 0;
\]
\[
\text{gestaltEasyAccessOn} = 1;
\]
\[
\text{gestaltEasyAccessSticky} = 2;
\]
\[
\text{gestaltEasyAccessLocked} = 3;
\]

Edition Manager Attributes

\[
\text{gestaltEditionMgrAttr} = \text{'edtn'};
\]
\[
\text{gestaltEditionMgrPresent} = 0;
\]
\[
\text{gestaltEditionMgrTranslationAware} = 1;
\]
The base address of the second half of the Toolbox trap table if the table is discontiguous. If the table is contiguous, this selector returns 0.

\[
\text{gestaltExtToolboxTable} = 'xttt';
\]

**Finder Attributes**

\[
\text{gestaltFinderAttr} = 'fndr'
\]

{Finder recognizes drop event}
\[
\text{gestaltFinderDropEvent} = 0;
\]

{Finder supports magic icon placement}
\[
\text{gestaltFinderMagicPlacement} = 1;
\]

{Finder calls AEProcessAppleEvent}
\[
\text{gestaltFinderCallsAEProcess} = 2;
\]

{Finder is scriptable and recordable}
\[
\text{gestaltOSLCompliantFinder} = 3;
\]

{Finder handles 4GB volumes}
\[
\text{gestaltFinderSupports4GBVolumes} = 4;
\]

{Finder handles Code Fragment Manager errors}
\[
\text{gestaltFinderHandlesCFMFailures} = 5;
\]

{Finder supports Drag Manager clipping files}
\[
\text{gestaltFinderHasClippings} = 6;
\]

**FindFolder Function Attribute**

\[
\text{gestaltFindFolderAttr} = 'fold';
\]

\[
\text{gestaltFindFolderPresent} = 0;
\]

**First Physical Slot**

\[
\text{gestaltFirstSlotNumber} = 'sltl';
\]

**Font Manager Attribute**

\[
\text{gestaltFontMgrAttr} = 'font';
\]

\[
\text{gestaltOutlineFonts} = 0;
\]

A constant that represents the type of floating-point unit currently installed, if any.

\[
\text{gestaltFPUPType} = 'fpu';
\]
gestaltNoFPU = 0;
gestalt68881 = 1;
gestalt68882 = 2;
gestalt68040FPU = 3;

**File System Attributes**

gestaltFSAtt = 'fs ';

[new HFSDispatch available]
gestaltFullExtFSDispatching = 0;
[has FSSpec calls]
gestaltHasFSSpecCalls = 1;
[has File System Manager]
gestaltHasFileSystemManager = 2;
[supports dynamic loading]
gestaltFSMDoesDynamicLoad = 3;
[supports 4 gigabyte volume]
gestaltFSSupports4GBVols = 4;
[has extended disk initialization calls]
gestaltHasExtendedDiskInit = 6;

**File Transfer Manager Attributes**

gestaltFXfrMgrAttr = 'fxfr';

[File Transfer Manager present]
gestaltFXfrMgrPresent = 0;
[supports FTSend and FTReceive]
gestaltFXfrMgrMultiFile = 1;
[supports FTGetErrorString]
gestaltFXfrMgrErrorString = 2;

**Help Manager Attribute**

gestaltHelpMgrAttr = 'help';

gestaltHelpMgrPresent = 0;
gestaltHelpMgrExtensions = 1
Icon Utilities Attribute

```plaintext
gestaltIconUtilitiesAttr = 'icon';
gestaltIconUtilitiesPresent = 0;
```

Image Compression Manager

```plaintext
gestaltCompressionMgr = 'icmp'
```

Keyboards

If the Apple Desktop Bus (ADB) is in use, there may be multiple keyboards or other ADB devices attached to the machine. The `gestaltKeyboardType` selector identifies only the type of the keyboard on which the last keystroke occurred.

You cannot use this selector to find out what ADB devices are connected. For that, you can use the Apple Desktop Bus Manager, described in *Inside Macintosh: Devices*. Note that the ADB keyboard types described by Gestalt do not necessarily map directly to ADB device handler IDs.

Future support for the `gestaltKeyboardType` selector is not guaranteed. To determine the type of the keyboard last touched without using Gestalt, check the system global variable `KbdType`, documented in *Inside Macintosh: Devices*.

If the Gestalt Manager does not recognize the keyboard type, it returns an error.

```plaintext
gestaltKeyboardType = 'kbd';
```

```plaintext
gestaltMacKbd = 1;
gestaltMacAndPad = 2;
gestaltMacPlusKbd = 3;
gestaltExtADBKbd = 4;
gestaltStdADBKbd = 5;
gestaltPrtblADBKbd = 6;
gestaltPrtblISOADBKbd = 7;
gestaltStdISOADBKbd = 8;
gestaltExtISOADBKbd = 9;
gestaltADBKbdII = 10;
gestaltADBKbdII = 11;
gestaltPwrBookADBKbd = 12;
gestaltPwrBookISOADBKbd = 13;
gestaltAppleAdjustKeypad = 14;
```
Appendix B: Gestalt Manager Selector Codes

gestaltAppleAdjustADBKbd = 15;
gestaltAppleAdjustISOKbd = 16;
gestaltJapanAdjustADBKbd = 17;
gestaltPwrBkExtISOKbd = 20;
gestaltPwrBkExtJISKbd = 21;
gestaltPwrBkExtADBKbd = 24;

The logical page size. This value is defined only on machines with the MC68010, MC68020, MC68030, or MC68040 microprocessors. On a machine with the MC68000, Gestalt returns an error when called with this selector.

gestaltLogicalPageSize = 'pgsz';

The amount of logical memory available. This value is the same as that returned by gestaltPhysicalRAMSize when virtual memory is not installed. On some machines, however, this value might be less than the value returned by gestaltPhysicalRAMSize because some RAM may be used by the video display and the Operating System.

gestaltLogicalRAMSize = 'lram';

The size (in bytes) of the low-memory area. The low-memory area is used for vectors, global variables, and dispatch tables.

gestaltLowMemorySize = 'lmem';

Information about miscellaneous pieces of the Operating System or hardware configuration.

gestaltMiscAttr = 'misc';
gestaltScrollingThrottle = 0;
gestaltSquareMenuBar = 2;

Mixed Mode Manager

The name gestaltMixedModeVersion for the 'mixd' selector is misleading. It has renamed gestaltMixedModeAttr to properly reflect the Inside Mac: PowerPC System Software documentation. gestaltMixedModeVersion will be removed in a future release of the Interfaces.

For the first version of Mixed Mode the 'mixd' selector returns 0x00000001. In subsequent versions of Mixed Mode, the 'mixd' selector will respond with 32 attribute bits with various meanings, instead of an increasing version number.

gestaltMixedModeVersion = 'mixd';
gestaltMixedModeAttr = 'mixd';

gestaltPowerPCAware = 0

A constant that represents the type of MMU currently installed:

gestaltMMUType = 'mmu';

gestaltNoMMU = 0;
gestaltAMU = 1;
gestalt68851 = 2;
gestalt68030MMU = 3;
gestalt68040MMU = 4;
gestaltEMMU1 = 5;

Native CPU Type

Note, to check whether the native system architecture is a MC680x0 or a PowerPC microprocessor, use the gestaltSysArchitecture selector.

gestaltNativeCPUtype = 'cput';

gestaltCPU68000 = $000;
gestaltCPU68010 = $001;
gestaltCPU68020 = $002;
gestaltCPU68030 = $003;
gestaltCPU68040 = $004;
gestaltCPU601 = $101;

Notification Manager Attribute

gestaltNotificationMgrAttr = 'nmgr';

gestaltNotificationPresent = 0;

A bitmap that describes the NuBus™ slot connector locations. On a Macintosh II, for example, the return value would have bits 9 through 14 set, indicating that 6 NuBus slots are present, at locations 9 through 14.

gestaltNuBusConnectors = 'sltc';

General Operating System attributes, such as whether temporary memory handles are real handles. The low-order bits of the response parameter are interpreted as bit flags. A flag is set to 1 to indicate that the corresponding feature is available. Currently, the bits below are significant.
Appendix B: Gestalt Manager Selector Codes

See the book *Inside Macintosh: Memory* for a full explanation of the temporary memory features, and see the book *Inside Macintosh: Processes* for a full explanation of the launch control features.

\[
\text{gestaltOSTAttr} = \text{'os'};\\
\text{gestaltSysZoneGrowable} = 0;\\
\text{gestaltLaunchCanReturn} = 1;\\
\text{gestaltLaunchFullFileSpec} = 2;\\
\text{gestaltLaunchControl} = 3;\\
\text{gestaltTempMemSupport} = 4;\\
\text{gestaltRealTempMemory} = 5;\\
\text{gestaltTempMemTracked} = 6;\\
\text{gestaltIPCSupport} = 7;\\
\text{gestaltSysDebuggerSupport} = 8;
\]

**Operating System Trap Dispatch Table**

The base address of the Operating System trap dispatch table is:

\[
\text{gestaltOSTable} = \text{'ostt'};\]

**Parity-Checking Features**

Note that parity is not considered to be enabled unless all installed memory is parity RAM.

\[
\text{gestaltParityAttr} = \text{'prty'};\\
\text{gestaltHasParityCapability} = 0;\\
\text{gestaltParityEnabled} = 1;
\]

**PC Exchange Attributes**

\[
\text{gestaltPCXAttr} = \text{'pcxg'};\\
\text{gestaltPCXHas8and16BitFAT} = 0;\\
\text{gestaltPCXHasProDOS} = 1;
\]
Physical RAM

The number of bytes of physical RAM currently installed:

```
   gestaltPhysicalRAMSize = 'ram';
```

Pop-Up Control Definition

The attribute of the pop-up control definition:

```
   gestaltPopupAttr = 'pop!';
   {pop-up 'CDEF' is present}
   gestaltPopupPresent = 0;
```

Power Manager Attributes

```
   gestaltPowerMgrAttr = 'powr';
   {Power Manager is present}
   gestaltPMgrExists = 0;
   {CPU can idle}
   gestaltPMgrCPUIdle = 1;
   {Power Manager can stop SCC clock}
   gestaltPMgrSCC = 2;
   {Power Manager can turn off sound power}
   gestaltPMgrSound = 3;
   {Power Manager dispatch exists}
   gestaltPMgrDispatchExists = 4;
```

Processor Attributes

A constant that represents the type of microprocessor currently running:

```
   gestaltProcessorType = 'proc';
   gestalt68000 = 1;
   gestalt68010 = 2;
   gestalt68020 = 3;
   gestalt68030 = 4;
   gestalt68040 = 5;
```
Appendix B: Gestalt Manager Selector Codes

Program-to-Program Communication (PPC) Toolbox Attributes

Note that these constants are defined as masks, not bit numbers.

\[
\text{gestaltPPCToolboxAttr} = 'ppc';
\]

{PPC Toolbox is present; PPCInit has been called}
\[
\text{gestaltPPCToolboxPresent} = \$0000;
\]
{supports real-time delivery}
\[
\text{gestaltPPCSupportsRealTime} = \$1000;
\]
{accepts sessions from remote computers}
\[
\text{gestaltPPCSupportsIncoming} = \$0001;
\]
{can initiate sessions with remote computers}
\[
\text{gestaltPPCSupportsOutgoing} = \$0002;
\]

QuickDraw Features

\[
\text{gestaltQuickdrawFeatures} = 'qdrw';
\]

{Color QuickDraw present}
\[
\text{gestaltHasColor} = 0;
\]
{graphics worlds can be deeper than 1 bit}
\[
\text{gestaltHasDeepGWorlds} = 1;
\]
{PixMaps can be direct (16- or 32-bit)}
\[
\text{gestaltHasDirectPixMaps} = 2;
\]
{supports text mode grayishTextOr}
\[
\text{gestaltHasGrayishTextOr} = 3;
\]
{supports video mirroring using the Display Manager}
\[
\text{gestaltSupportsMirroring} = 4;
\]

The version of QuickDraw, encoded as a revision number in the low-order word of the return value. The high-order byte represents the major revision number, and the low-order byte represents the minor revision number. For example, version 1.3 of 32-Bit QuickDraw represents QuickDraw revision 2.3; its response value is \$0230.

Values having a major revision number of 1 or 2 indicate that Color QuickDraw is available, in either the 8-bit or 32-bit version. These results do not, however, indicate whether a color monitor is attached to the system. You must use high-level QuickDraw routines to obtain that information.

\[
\text{gestaltQuickdrawVersion} = 'qd ';
\]
\[
\text{gestaltOriginalQD} = \$000;
\]
\[
\text{gestalt8BitQD} = \$100;
\]
gestalt32Bit0DQ = $200;
gestalt32Bit0DQ11 = $210;
gestalt32Bit0DQ12 = $220;
gestalt32Bit0DQ13 = $230;

QuickTime Version

gestaltQuickTimeVersion = 'qtim';

PowerPC QuickTime glue library is present:

gestaltQuickTimeFeatures = 'qtrs',
gestaltPPCQuickTimeLibPresent = 0

Realtime Manager Attributes

gestaltRealtimeMgrAttr = 'rtmr';
gestaltRealtimeMgrPresent = 0;

Resource Manager Attribute

gestaltResourceMgrAttr = 'rsrc';

{partial resources supported}
gestaltPartialRsrcs = 0;

Scrap Manager Attributes

gestaltScrapMgrAttr = 'scra';

gestaltScrapMgrTranslationAware = 0;
gestaltTranslationMgrHintOrder = 1;

Script Manager

The version number of the Script Manager (in the low-order word of the return value):

gestaltScriptMgrVersion = 'scri';
Appendix B: Gestalt Manager Selector Codes

Script Systems

The number of script systems currently active:

gestaltScriptCount = 'scr#';

Serial Hardware Attributes

Serial hardware attributes of the machine, such as whether or not the GPI a line is connected and can be used for external clocking.

gestaltSerialAttr = 'ser';

[GPI connected to DCD on port A]
gestaltHasGPIToDCDa = 0;

[GPI connected to RTx C on port A]
gestaltHasGPIToRTxCa = 1;

[GPI connected to DCD on port B]
gestaltHasGPIToDCDb = 2;

Slot Manager Attributes

gestaltSlotAttr = 'slot'

[Slot mgr exists]
gestaltSlotMgrExists = 0;

[NuBus slots are present]
igestaltNuBusPresent = 1;

[gestaltSESlotPresent]
igestaltSESlotPresent = 2;

[SE/30 slot present]
igestaltSE30SlotPresent = 3;

[Portable's slot present]
igestaltPortableSlotPresent = 4;

Sound Attributes

If the bit gestaltStereoCapability is TRUE, the available hardware can play stereo sounds. The bit gestaltStereoMixing indicates that the sound hardware of the machine mixes both left and right channels of stereo sound into a single audio signal for the internal speaker. The gestaltSoundIOMgrPresent bit indicates that the new sound input routines are available, and the gestaltBuiltInSoundInput bit indicates that a built-in sound input device is available. The gestaltHasSoundInputDevice bit indicates that some sound input device is available.
Note bits 7 through 12 are not defined for versions of the Sound Manager prior to version 3.0.

```
gestaltSoundAttr = 'snd';

{stereo capability present}
gestaltStereoCapability = 0;
{stereo mixing on internal speaker}
gestaltStereoMixing = 1;
{sound input routines present}
gestaltSoundIOMgrPresent = 3;
{built-in input device present}
gestaltBuiltInSoundInput = 4;
{sound input device present}
gestaltHasSoundInputDevice = 5;
{built-in hardware can play and record simultaneously}
gestaltPlayAndRecord = 6;
{sound hardware can play and record 16-bit samples}
gestalt16BitSoundIO = 7;
{sound hardware can record stereo}
gestaltStereoInput = 8;
gestaltLineLevelInput = 9;
{SndPlayDouble buffer present}
gestaltSndPlayDoubleBuffer = 10;
{multiple channel support}
gestaltMultiChannels = 11
{16-bit audio data supported}
gestalt16BitAudioSupport = 12;
```

**Speech Manager Attributes**

```
gestaltSpeechAttr = 'ttsc';

{Speech Manager present}
gestaltSpeechMgrPresent = 0;
{Speech Manager has native PPC glue for API}
gestaltSpeechHasPPCGlue = 1;
```

**Standard File Package Attributes**

If the `gestaltStandardFile58` flag bit is set, you can call the four new procedures—`StandardPutFile`, `StandardGetFile`, `CustomPutFile`, and `CustomGetFile”—introduced with System 7. (The name of the constant reflects the enabling of selectors 5 through 8 on the trap macro that handles the Standard File Package.)
Appendix B: Gestalt Manager Selector Codes

```
gestaltStandardFileAttr = 'stdf';
gestaltStandardFile58 = 0;
gestaltStandardFileTranslationAware = 1;
gestaltStandardFileHasColorIcons = 2;
gestaltStandardFileUseGenericIcons = 3;
gestaltStandardFileHasDynamicVolumeAllocation = 4;
```

**StandardNBP Function**

Information about the StandardNBP (Name-Binding Protocol) function.

```
gestaltStdNBPAttr = 'nlup';
gestaltStdNBPPresent = 0;
```

**System Architecture**

The native system architecture. If the gestalt68k flag bit is set, the native microprocessor is a MC680x0 chip. If the gestaltPowerPC flag bit is set, the native microprocessor is a PowerPC chip.

```
gestaltSysArchitecture = 'sysa';
gestalt68k = 1;
gestaltPowerPC = 2;
```

**TextEdit Attributes**

```
gestaltTEAttr = 'teat';

{TextEdit has TEGetHiliteRgn}
gestaltTEHasGetHiliteRgn = 0;
{TextEdit does Inline Input}
gestaltTESupportsInlineInput = 1;
{TextEdit does Text Objects}
gestaltTESupportsTextObjects = 2;
```

**Terminal Manager Attributes**

```
gestaltTermMgrAttr = 'term';
gestaltTermMgrPresent = 0;
gestaltTermMgrErrorString = 2;
```
TextEdit

A constant that indicates which version of TextEdit is present.

gestaltTextEditVersion = 'te';

[TextEdit in MacIIci ROM]

gestaltTE1 = 1;

[TextEdit with 6.0.4 Script Systems on MacIIci]

(gestaltTE2 = 2;

[TextEdit with 6.0.4 Script Systems all but MacIIci]

gestaltTE3 = 3;

[TextEdit in System 7.0]

gestaltTE4 = 4;

(TextWidthHook available in TextEdit)

gestaltTE5 = 5;

[TextEdit in System 8.0]

gestaltTE6 = 6

Thread Manager Attributes

gestaltThreadMgrAtt = 'thds';

[Thread Mgr is present]

gestaltThreadMgrPresent = 0;

(bit true if Thread Mgr supports exact match creation option)

gestaltSpecificMatchSupport = 1;

(bit true if Thread Mgr shared library is present)

gestaltThreadsLibraryPresent = 2

Time Manager

A constant that indicates which version of the Time Manager is present.

gestaltTimeMgrVersion = 'tmgr';

gestaltStandardTimeMgr = 1;

gestaltRevisedTimeMgr = 2;

gestaltExtendedTimeMgr = 3;
Appendix B: Gestalt Manager Selector Codes

Toolbox

The base address of the Toolbox trap dispatch table.

\[
gestaltToolboxTable = 'tbtt';
\]

Translation Manager Attributes

\[
\begin{align*}
gestaltTranslationAttr &= 'xlat'; \\
gestaltTranslationMgrExists &= 0;
\end{align*}
\]

Text Services

\[
gestaltTSMgrVersion = 'tsmv';
\]

Version of Gestalt Manager

The version of the Gestalt Manager (in the low-order word of the return value). The current version is 1, corresponding to a returned value of $0001.

\[
gestaltVersion = 'vers';
\]

\{version of gestalt where gestaltValue is implemented\}

\[
gestaltValueImplementedVers = 5
\]

Virtual memory Attributes

\[
\begin{align*}
gestaltVMAttr &= 'vm ' ; \\
gestaltVMPresent &= 0;
\end{align*}
\]

Informational Selectors

The following sections list the informational selector codes.

Low-Level Hardware Configuration Attributes

The \texttt{gestaltHasSCSI} bit means the machine is equipped with a SCSI implementation based on the \texttt{53C80} chip, which was introduced in the Macintosh Plus. This bit is 0 on computers with a different SCSI implementation. Those computers set the \texttt{gestaltHasSCSI961}
or `gestaltHasSCSI962` bit to report a SCSI implementation based on the 53C96 chip installed on an internal or external bus, respectively.

The `gestaltHasSCC` bit is normally returned as 0 on the Macintosh II fx and Macintosh Quadra 900 computers, which have intelligent I/O processors that isolate the hardware and make direct access to the SCC impossible. However, if the user has used the Compatibility Switch control panel to enable compatibility mode, `gestaltHasSCC` is set.

```plaintext
gestaltHardwareAttr = 'hdwr';
{has VIA1 chip}
gestaltHasVIA1 = 0;
{has VIA2 chip}
gestaltHasVIA2 = 1;
{has Apple sound chip}
gestaltHasASC = 3;
{has SCC}
gestaltHasSCC = 4;
{has SCSI}
gestaltHasSCSI = 7;
{capable of software power off}
gestaltHasSoftPowerOff = 19;
{has 53C96 SCSI on internal bus}
gestaltHasSCSI961 = 21;
{has 53C96 SCSI on external bus}
gestaltHasSCSI962 = 22;
{has universal ROM}
gestaltHasUniversalROM = 24;
```

**Icon Family Resource ID for the Current Type of Macintosh**

```plaintext
gestaltMachineIcon = 'micn';
```

**A Constant that Indicates the Model of Computer**

To obtain a string containing the machine's name, you can pass the returned value to the `GetIndString` procedure as an index into the resource of type 'STR#' in the System file having the resource ID defined by the constant `kMachineNameStrID (-16395)`.

```plaintext
gestaltMachineType = 'mach';

gestaltClassic = 1;
gestaltMacXL = 2;
```
Appendix B: Gestalt Manager Selector Codes

gestaltMac512KE = 3;
gestaltMacPlus = 4;
gestaltMacSE = 5;
gestaltMacII = 6;
gestaltMacIIx = 7;
gestaltMacIIcx = 8;
gestaltMacSE030 = 9;
gestaltPortable = 10;
gestaltMacIIci = 11;
gestaltMacIIfx = 13;
gestaltMacClassic = 17;
gestaltMacIIsi = 18;
gestaltMacLC = 19;
gestaltQuadra900 = 20;
gestaltPowerBook170 = 21;
gestaltQuadra700 = 22;
gestaltClassicII = 23;
gestaltPowerBook100 = 24;
gestaltPowerBook140 = 25;
gestaltQuadra950 = 26;
gestaltMacLCIII = 27;
gestaltPowerBookDuo210 = 29;
gestaltMacCentris650 = 30;
gestaltPowerBookDuo230 = 32;
gestaltPowerBook180 = 33;
gestaltPowerBook160 = 34;
gestaltMacQuadra800 = 35;
gestaltMacLCII = 37;
gestaltPowerBookDuo250 = 38;
gestaltMacIivi = 44;
gestaltPerforma600 = 45;
gestaltMacIivx = 48;
gestaltMacColorClassic = 49;
gestaltPowerBook165c = 50;
gestaltMacCentris610 = 52;
gestaltMacQuadra610 = 53;
gestaltPowerBook145 = 54;
gestaltMacLC520 = 56;
gestaltMacCentris660AV = 60;
gestaltPerforma46x = 62;
gestaltPowerMac8100_80 = 65;
gestaltPowerBook180c = 71;
Macworld Mac Programming FAQs

gestaltPowerMac6100_60 = 75;
gestaltPowerBookDuo270c = 77;
gestaltMacQuadra840AV = 78;
gestaltPerforma550 = 80;
gestaltPowerBook165 = 84;
gestaltMacTV = 88;
gestaltMacLC475 = 89;
gestaltMacLC575 = 92;
gestaltMacQuadra605 = 94;
gestaltQuadra630 = 98;
gestaltPowerMac6100_66 = 100;
gestaltPowerBookDuo280 = 102;
gestaltPowerBookDuo280c = 103;
gestaltPowerMac7100_66 = 112;
gestaltPowerBook150 = 115;

Size of the installed ROM, in Bytes

The value is returned in only one word.

    gestaltROMSize = 'rom ';

Version Number of the Installed ROM

The version number of the installed ROM (in the low-order word of the return value).

    gestaltROMVersion = 'romv';

Version Number of the System File

The version number of the currently active System file, represented as four hexadecimal digits in the low-order word of the return value. For example, if your application is running in version 7.0.1, then Gestalt returns the value $0701. Ignore the high-order word of the returned value.

    gestaltSystemVersion = 'sysv';
INDEX

A

A5 world. See memory
AIFF files. See sound
alerts, 114–117
customizing text in, 116
displaying and closing, 114
displaying custom icons in, 114
incrementing and resetting alert stage value, 116
See also controls; dialog boxes
Apple events. See events
arrow cursor. See cursors
Assembly code
in CodeWarrior C projects, 61–62, 63
PowerPC versus 68K Assembly code, 61
in Symantec C projects, 62, 63
See also C language; Pascal

B

bit manipulation, 520–521
bitmaps. See QuickDraw
“bottleneck” functions. See QuickDraw
buttons. See controls

C

C language
Assembly code in CodeWarrior C projects, 61–62, 63
Assembly code in Symantec C projects, 62, 63
Code Warrior Gold, Academic, Bronze, and Discover differences, 64
converting C strings and Pascal strings, 516–517
ONEWORDINLINE(), TWOWORDINLINE(),
THREEWORDINLINE() macros in universal header files, 60–61
versus Pascal, 58–59
Symantec C++ version 8.0, 65
See also Assembly code; Pascal
CD-ROM (book's), 601–607
FAQ DB Viewer program, 602–606
folders, 601–602
opening and using sample code projects, 606–607
click loops. See List Manager
color
Color Manager documentation, 24
color settings for dialog box items, 104–105
crashes caused by color cursors, 44
declarations, 12–16
drawing in color, 16–23
48-bit RGB values on 16-bit screens, 23
Basic QuickDraw RGB values, 22
with colors at specific locations in color palettes, 26–27
converting from one color model to another, 17
creating grayscale ramps, 17–18
drawing in highlight colors with drawing commands, 21–22
inverting colors in 256-color systems, 19
retrieving and setting highlight color values, 20–21
RGB values for dimmed buttons, menus, and window titles, 19–20
with RGBForeColor(), 23
HSV color model, 12–14
pmCourteous, pmTolerant, and pmExplicit colors, 14–15
in QuickDraw
assigning custom colors to GWorlds, 406–407
color distortion using CopyBits() in GWorlds, 406
displaying selection rectangles in color, 412–413
drawing in color patterns, 389–390
drawing shapes filled with solid colors, 386
passing pixmaps of color windows to CopyBits(), 400
retrieving most frequently used colors in pixmaps, 398
in windows
creating color backgrounds for windows, 593–594
drawing in color, 592
passing pixmaps of color windows to CopyBits() in
QuickDraw, 400
RGB values for dimmed window titles, 19–20
setting custom colors, 590–591
color lookup tables (CLUTs), 14–16, 23–38, 442–443
default color table IDs, 34–35
defined, 14
and indexes of RGB values, 32–33
inverse tables, 15–16
loading color tables into palettes and palettes into color tables, 27
monitoring color tables in indexed devices, 35
restoring color environments when applications go to background, 33–34
retrieving color tables of pictures, 442–443
retrieving color tables of screens, 31
color palettes, 14, 23–38, 442–443
animated colors and, 35–37
changing, 25
converting RGB color into indexes to, 25–26
creating, 24–25
color palettes (continued)
creating floating palettes, 581-587
defined, 14
defining default palette for applications, 29
displaying pictures not using default palette, 30-31
drawing with colors at specific locations in, 26-27
getting screens to use, 26
loading color tables into color palettes and color palettes into color tables, 27
Palette Manager documentation, 24
retrieving default application palette, 27-28
retrieving palettes of pictures, 442-443
windows and
applying changes to windows, 30
assigning to windows, 28-29
disposing of color palettes attached to windows, 30
retrieving window color palettes, 28
controls, 76, 83-99
activating, 93
changing button colors, 89
color cursor crashes, 44
dimming with light gray instead of bittmappad gray, 89-91
moving buttons with MoveControl(), 92
refreshing when windows come to front, 92-93
setting default items and default cancel items, 88
See also alerts; dialog boxes; scroll bars
cursors, 39-56, 570-571
animated cursors, 40, 49-56
changing cursor shape over areas of windows, 42-43
changing shape of, 40-42
color cursor crashes, 44
hiding, 44-45
mouse events and hidden cursors, 45
moving to specific locations, 46-49
resetting cursor shape when application returns to foreground, 44
resetting to arrow cursor, 42
retrieving cursor location, 46
ShowCursor() and HideCursor(), 45
in TextEdit fields, 570-571
dialog boxes, 76-83, 99-105
creating
color dialog boxes, 80-81, 82
modal dialog boxes, 77-78
modeless dialog boxes, 79-80
movable modal dialog boxes, 79
deactivating static text items and edit text items, 102-103
disabling edit text fields with HiliteControl(), 102
drawing on, 81
event filter procedures in, 106-113
activating default items with Return key, 113
creating event filters, 106-109
password entries in edit text items, 111-112
restricting edit text items to numeric input, 109-110
update events in event filters, 113
finding items with FindDialogItem(), 101
font and color settings for items, 104-105
icon types displayable in, 238
modal dialog boxes
creating, 77-78
creating movable modal dialog boxes, 79
and disk-insert events, 82
putting lists in, 247-250
Page Setup and Print dialog boxes
creating, 317
saving options set in, 320-321
putting PICT items in backgrounds, 81
retrieving text from editable text items, 102
storing data in refCon fields, 598
text-entry field background settings, 105
user items
defined, 76
drawing in dialog boxes with, 100-101
user-interface guidelines for item placement, 82-83
variables in static text fields, 103-104
See also alerts; controls; menus; scroll bars; Standard File Package; windows
directories. See File Manager
drawing. See color; QuickDraw; text; windows
drives. See File Manager

--- D ---

Desktop, 154, 196, 199-204, 591-592
accessing root directory on, 154
attaching icons to applications, 201-202
changing application icons, 201
Desktop database
accessing, 199-200
defined, 196
displaying bundled icons, 203
drawing outside windows on the Desktop in QuickDraw, 417-419
retrieving applications that open files with a given creator, 200
retrieving file icons of a given type and creator, 202-203
retrieving GraFXPort of, 591-592
See also Finder; icons
development environments. See Assembly code; C language; Pascal
F

FAQ DB Viewer program, 602-606
fat binaries. See Power Macintosh programming
File Manager, 143-181
definitions, 144-147
creator IDs and file types, 146
default versus current directory and volume, 145-146
drive queues, 147
FSSpec (file system specification), 144-145
resource and data forks of files, 147
working directories, 146-147
disk drive operations, 175-181
determining free space on volumes, 180
determining if disk is locked, 176-177
determining if diskette is in drive, 175-176
determining if volumes are floppy disks or hard disks, 177-180
referencing files, folders, and volumes, 147-155
accessing files located in same folder with applications, 150
accessing root directory of volumes and on the Desktop, 154
converting working directories to volume reference numbers and directory IDs, 155
current directory and volume in Standard File Package calls versus File Manager calls, 153-154
generating full path name from vRefNum, parent directory ID, and file name, 149-150
maximum size of file, directory, and volume names, 148-149
obtaining folder names from dirID and vRefNum, 152
obtaining folder's directory ID from full path name, parent folder ID, and directory name, 151-152
retrieving directory and volume of application files, 154-155
retrieving and setting default volume and folder, 150
retrieving vRefNum of startup volumes, 153
specifying path names and separating volume, directory, and file names, 148
WDRefNums versus vRefNums, 153
searching files, folders, and volumes, 155-162
finding all files in a directory, 156
finding all mounted volumes, 158-159
finding invisible files and folders in directories, 155-156
reversing through all files and folders in directories, 156-158
searching volumes for files matching search criteria, 159-152
system folders, 162-165
accessing the System folder, 162-163
finding System and Preferences folders, 164-165
where to store temporary files, 163-164
working with files, folders, and volumes, 165-175
changing files without changing the time stamp, 169-170
compiler errors when accessing ioNameptr field of HParamBlockRec structures, 173
creating invisible files, 170-171
deleting folders, 169
deleting resource or data fork only, 168-169
determining if files are open or which fork is open, 167-168
file and allocation block size on hard drives, 172
interchanging file reference numbers from data forks and resource forks, 167
opening files in current directory with FSOpen(), 172
opening files with names beginning with period (.), 172
reading and writing to files, 166-167
retrieving files' Finder label menu color and label string, 174-175
See also Standard File Package filters. See dialog boxes; GNE filters
Finder, 5-6, 174-175, 196-199
drawing icons in different Finder states, 239-240
events and, 5-6
forcing folder updates, 197
forcing window updates, 198-199
getting applications to respond to, 196
icon types displayable in, 238
retrieving files' Finder label menu color and label string, 174-175
See also Desktop folders. See File Manager; Finder fonts, 546-549
application fonts, 546
changing font, style, and size of drawn text, 550-551
determining font height in pixels, 549
determining if graphics ports will use bitmapped or outline fonts, 548
determining width of drawn text from font attributes, 552
font families defined, 546
font settings for dialog box items, 104-105
identifying available fonts, 548
outline versus bitmapped fonts, 546-547
retrieving font family ID from font name, 547
retrieving font family name from font family ID, 548
System fonts, 546
why text drawn with font IDs looks different on different computers, 548
See also strings; text
G

Gestalt Manager, 205-209, 609-630
communicating between processes with, 209
determining availability of, 205-206
retrieving information with, 206
selector codes
equipment selector codes, 610-627
global variables in, 208-209
informational selector codes, 627-630
registering new selectors, 207-208
GNE filters, 120, 128, 139-142
creating and installing, 139-141
defined, 120
detecting OS events with, 141-142
keyboard events and, 128
See also events
graphics devices, 211-236
Gamma tables defined, 212
main graphics devices defined, 212
manipulating, 217-233
capturing screen images, 227-232
fading direct color device screens to black, 223-227
graphics devices (continued)
fading screens to black, 218–222
pixel depth and color mode settings, 232–233
producing grays with SetDepth(), 233
retrieving device information, 213–217
determining maximum pixel depth supported by
screens, 213
determining number of monitors attached to a
computer, 214–215
determining which is main monitor, 214
retrieving monitors' pixel size, 217
retrieving screen with greatest color depth, 214
retrieving screens' pixel depth and color mode,
215–217
working with, 233–236
determining window/screen intersections, 234–236
drawing window contents on multiple, overlapping
screens with different depths, 234
See also printers; QuickDraw
gray shades. See color
GWorlds. See QuickDraw
—H—
handles. See memory
heap. See memory
hierarchical menus. See menus
—I—
icons, 201–203, 237–241
adding to menus, 277–279
creating masks for icons with different bit
depths, 240
creating menus with icons in menu title, 290–291
on Desktop, 201–203
attaching to applications, 201–202
changing application icons, 201
displaying bundled icons, 203
retrieving file icons of a given type and creator,
202–203
displaying custom icons in alert dialog boxes, 114
displaying icon for System extensions at startup,
531–532
drawing in different Finder states, 239–240
icon families and suites defined, 237–238
icon types displayable in dialog boxes and
menus, 238
icon types displayable in Finder, 238
notifying users by blinking icons, 331
PlotClcon() function, 240–241
speeding up 'icon' icon display for animation, 241
why only part of icon draws, 240
See also Desktop
INITs. See System extensions
—K—
keyboard events, 120–128
capturing keystrokes in applications, 128
detecting
arrow key presses, 125
keydown events, 120–121
multiple simultaneous keystrokes, 123
Shift, Control, Command, or Option key presses,
123–124, 126
determining if Cancel button in a dialog box is pressed,
185
passing back key values with GetKeys(), 122
reading keys without retrieving events from event queue,
121–122
retrieving character code and virtual key code of pressed
keys, 127–128
retrieving keydown events in all applications with traps,
543–544
WaitNextEvent() and key-up events, 126–127
See also events
—L—
languages. See Assembly code; C language; Pascal
List Manager, 243–257
click loops, 244, 254–257
defined, 244
displaying cell information when mouse drags over
text fields, 257
installing in lists, 255–256
why click loops won't work, 256
displaying lists, 250–252
changing column widths, 250
changing text size and style, 251–252
creating lists of icons or pictures, 252
using different data types in lists, 251
list definition procedure (LDEF) defined, 244
list information, 252–254
determining if cells are selected, 253–254
determining number of rows and columns in lists, 254
determining which cells are visible, 254
retrieving currently selected cells, 253–254
retrieving lists' current display area size, 253
working with lists, 244–250
detecting double-clicks in cells, 245
disabling flashing in changed lists, 245–246
list size limit, 245
preventing multiple cell selection, 246–247
putting lists in modal dialog boxes, 247–250
unrecognizable characters in front of strings
explained, 245
why lists won't draw or reflect changes, 247
—M—
marquees. See QuickDraw
memory, 259–270
A5 world, 261–262, 267–269, 311
A4 register in code resources, 269
and accessing applications' global variables, 311
accessing from functions called at interrupt time,
267–268
defined, 261–262
when to explicitly set up and restore, 268–269
determining memory available to applications, 266
freeing allocated memory when application exits, 267
handles and pointers, 260–261, 262–265
changing handle size, 264
copying data between handles, 264
determining if handles are already locked before
locking, 263
See also System extensions
determining if purgeable blocks are valid, 263
handles defined, 260–261
passing pointers to memory block pointers where handles are needed, 262
retrieving size of data referenced by, 265
returning handles to blocks of zeroed-out memory, 265
unlocking locked handles, 262
using handles versus pointers, 264–265
when to lock handles, 263
moving memory with patch routines, 543
System heap
allocating memory in System heap versus application heap, 266–267
defined, 261
loading resources into, 447–449
temporary memory, 261, 270
menus, 19–20, 174–175, 238, 271–303
adding
Apple menu to applications, 275–276
application menu to menu bar, 277
check marks to menu items, 280–281
icons and Command-key equivalents, 278–279
icons next to menu item text, 277–278
menus to applications, 273–275
changing
font, style, and size, 282
menu titles after installation, 283–284
counting number of items in, 282
creating
customized menu definition procedures (MDEF), 291–298
with icons in menu title, 290–291
menus of available fonts, 283
menus of pictures, 302–303
using metacharacters, 282
determining if menus or items are disabled, 279–280
determining when menus are about to be opened, 284
dividers in, 277
drawing windows while menus are pulled down, 300–302
ellipsis (…) in, 280
enabling and disabling menu items past position 31, 281
GetMenu() and memory leaks, 285
hiding menu bars, 287–290
hierarchical menus
adding to applications, 285–286
defined, 272
displaying, 285
icon types displayable in, 238
menu IDs for selected menus, 285
MenuHook and MBarHook procedures defined, 272
installing, 298–300
notifying users by marking application menu, 331
number of menu items allowed, 281
retrieving files’ Finder label menu color and label string, 174–175
RGB values for dimmed menu items, 19–20
See also dialog boxes
modal dialog boxes. See dialog boxes
monitors. See graphics devices
mouse events, 128–133
detecting
double-clicks, 129–130
double-clicks in cells of lists, 245
mouse movement, 131–133
mouseDown and mouseUp events, 129
determining position of mouse click, 133
processing mouse clicks in TextEdit fields, 566
retrieving in all applications with traps, 543–544
retrieving current mouse coordinates, 130–131
See also events
 Notification Manager, 327–337
accessing global variables from notification procedures, 336–337
blinking color icons with, 331
computer hangs when notification procedure is activated a second time, 336
crashes caused by notifications, 334
deleting notifications from notification queue, 331–334
determining if notification was posted, 331
executing response procedures from notifications, 335–336
problems with marking application menu or blinking icons, 331
sending notifications from applications in background, 330
sending notifications from interrupt tasks, 334
See also Process Manager; Time Manager; Vertical Retrace Manager
numbers, converting strings and, 519–520
off-screen memory areas. See graphics devices
palettes. See color palettes
Pascal
versus C language, 58–59
converting C strings and Pascal strings, 516–517
finding functions for Pascal strings, 518
using "pascal" keyword, 60
See also Assembly code; C language
patching traps. See traps
path names. See File Manager
picture comments. See printing
pictures. See graphics devices; QuickDraw
pixmaps. See QuickDraw
pointers. See memory
ports. See serial ports
Power Macintosh programming, 305–314
A5 world and accessing applications’ global variables, 311
code fragments defined, 307
fat binaries
creating fat binary applications, 309–310
defined, 306
differences between native PowerPC, 68K, and fat applications, 310
Power Macintosh programming (continued)
forcing fat binary applications to run 68K code under emulation, 310
porting 68K code to native PowerPC code, 308
universal header files defined, 306
universal procedure pointers, 311–314
in 68K compiled code, 312
calling functions with, 311, 312
converting procptrs to, 308–309
defined, 307
disposing of pointers returned by New***Proc() macros, 308
explicitly calling routines associated with, 313
printers
determining if printer port is present, 70
determining if printer supports color, 323
outputting text or graphics to, 317–318
retrieving name and type of selected printer, 322–323
setting resolution of, 321–322
See also graphics devices
printing, 315–326
multiple page graphics or text, 319–320
Page Setup and Print dialog boxes
creating, 317
saving options set in, 320–321
with picture comments, 316–317, 324–326
dashed lines, 326
defined, 316–317
lines thinner than one QuickDraw point, 326
rotated text, 324–326
retrieving page size, 321
setting name of print jobs created by PrOpen(), 324
when user has selected to spool document to disk, 318–319
Process Manager, 339–347
background-only applications defined, 340–341
current processes versus front processes, 340
determining if an application is the front process, 341
determining if processes are running and if they support high-level events, 342–343
interrupts and interrupt time defined, 340
launching applications, but not as front application, 347
launching applications from applications, 345–347
listing currently running processes, 342
notifying applications when applications they launch terminate, 345
retrieving name and location of applications, 343–344
why applications terminate after calling LaunchApplication(), 345
why applications won't run in background, 344
See also Notification Manager; Time Manager; Vertical Retrace Manager
processing events, 134–139
 applications receiving suspend or resume events, 138
delaying applications for a specified time, 139
determining if applications go to background and return to foreground, 137
determining if windows are activated or deactivated, 136–137, 588, 593
determining time elapsed between events, 134
determining type of OS event, 135–136
event priority in Event Manager, 134
posting events to head of event queue, 138
sending events to applications, 138
sleep parameter in WaitNextEvent(), 135
stopping continuous update events, 139
See also events
programming with events, 3
See also Power Macintosh programming

QuickDraw, 377–457
bitmaps and pixmaps, 392–400
copying overlapping rectangles in the same window with CopyBits(), 400
creating pictures from pixmaps, 429
determining if structures are bitmaps or pixmaps, 399
distortion of images copied with CopyBits(), 399
flipping pixmaps, 392–396
negative values in pixmap rowBytes fields, 398–399
passing pixmaps of color windows to CopyBits(), 400
retrieving most frequently used colors in pixmaps, 398
retrieving pixel addresses of GWorld pixmaps, 404–405
rotating pixmaps, 397–398
size of rowBytes field in GWorld pixmaps, 403–404
speeding up CopyBits(), 396–397
“bottleneck” functions
calling at interrupt time, 424
defined, 378–379
replacing standard functions, 419–423
creating pictures, 426–434
combining two pictures, 428–429
CopyMask() and other functions not recorded in pictures, 432
destroying PicHandle with KillPicture(), ReleaseResource(), and DisposeHandle(), 434
determining if picture creation was successful, 430–431
interrupting picture creation, 431–432
overview of, 426–428
pictures draw as empty space, 429–430
from pixmaps, 429
resolution settings, 433–434
of window contents, 432–433
drawing, 380–391, 413–424
accessing global variables, 381
in color patterns, 389–390
dashed lines, 385–386
dimmed text or objects, 416–417
highlighting objects, 383–384
highlighting objects versus inverting objects, 382–383
implementing ZoomRects, 414–416
initializing applications, 380–381
lines and shape outlines with different widths, 384–385
outside windows on the Desktop, 417–419
in patterns, 387–388
in response to update window events, 391
retrieving and setting RGB values for single pixels, 390
retrieving standard patterns, 388–389
setting pen pattern with universal headers, 386–387
shapes filled with solid colors, 386
drawing pictures, 435–439
centering pictures in windows, 437–438
how applications process picture comments while
drawing pictures, 438–439
in modes other than srcCopy mode, 437
rotating pictures, 439
with transparent backgrounds, 435–437
gray regions defined, 378
GWorlds, 378, 401–407
allocating memory for GWorld buffers, 403–404
assigning custom colors to, 406–407
color distortion using CopyBits(), 406
defined, 378
drawing in windows with, 401–403
problems using CopyBits() from offscreen bitmaps or,
retrieving pixel addresses of GWorld pixmaps, 404–405
size of rowBytes field in GWorld pixmaps, 403–404
marquees and selection lines, 407–413
displaying selection rectangles in color, 412–413
drawing anchored lines and selection rectangles
(“rubber banding” lines), 408–410
drawing “marching ants” (dashed) marquees,
410–412
picture files, 439–441
PICT file format, 439
reading pictures from PICT files, 441
writing pictures to PICT files, 440
pictures, 426, 442–443
defined, 426
retrieving palettes or color tables of pictures, 442–443
retrieving size (bounding rectangle) of pictures, 443
RGB values used in Basic QuickDraw, 22
random number generation, 522–523
resource files, 450–457
“1” in Resource Manager routines, 451
accessing resources from other files, 453
copying resources between, 452–453
determining which resource file resource handles come from,
454–455
how applications access resources in their own resource
fork or System file, 457
how applications retrieve file reference numbers of their own
resource file, 453
limiting resource searches to a single file, 456
maximum number of resources in, 456–457
opening resource fork of files created with FSpCreate(), 451
retrieving number of resources of a particular type in,
456
saving resources to, 451–452
search order in multiple open resource files, 455–456
setting current resource file, 453
resources, 445–450
case sensitivity of resource types, 446
code resources
A4 register in, 269
adding to applications, 446–447
sending events from, 8
detaching, 447
determining if purgeable resources have been purged, 450
determining if Resource Manager functions are
successful, 449
determining the order that resources are returned by
GetlndResource(), 449–450
loading into System heap, 447–449
resource maps, 446
retrieving machine name and user name, 449
updating, 450
valid resource IDs, 449
RGB color model. See color

— S —
Scrap Manager, 459–466
clearing data from the scrap, 464
copying scrap to files, 465
data copied to the scrap does not paste, 464
data types allowed in the scrap, 463
determining if data is in the scrap, 460
determining if data in the scrap has been changed by
another application, 462
determining if a particular data type is in the scrap, 462
determining if the scrap is in memory or has been written to
a file, 466
determining size of handle to pass GetScrap(), 462
does new data replace existing data in the scrap,
463–464
effect on the scrap of GetScrap(), PutScrap(), or
DialogCopy(), 466
identifying data types in the scrap, 462
reading all data from the scrap, 464–465
reading data from and writing data to the scrap, 463
scrap defined, 460
screens. See graphics devices
scroll bars, 94–99, 568–570, 594–597
creating in windows, 94–99
determining window title bar height in pixels, 595
drawing grow boxes, 594, 596
preventing scroll bars from getting drawn on, 596–597
refreshing when window comes to front, 593
in TextEdit fields, 568–570
See also windows
selection lines. See QuickDraw

— R —
random number generation, 522–523
resource files, 450–457
“1” in Resource Manager routines, 451
accessing resources from other files, 453
copying resources between, 452–453
determining which resource file resource handles come from,
454–455
how applications access resources in their own resource
fork or System file, 457
how applications retrieve file reference numbers of their own
resource file, 453
limiting resource searches to a single file, 456
maximum number of resources in, 456–457
opening resource fork of files created with FSpCreate(), 451
retrieving number of resources of a particular type in,
456
saving resources to, 451–452
search order in multiple open resource files, 455–456
setting current resource file, 453
resources, 445–450
case sensitivity of resource types, 446
code resources
A4 register in, 269
adding to applications, 446–447
sending events from, 8
detaching, 447
determining if purgeable resources have been purged, 450
sound, 115–116, 467–507
double buffering defined, 468
playing, 115–116, 470–485
AIFF files, 472–473
asynchronously, 478–479
causes of static, 483
changing playback rate, 477–478
custom sounds for alerts, 115–116
determining when asynchronous sound has completed
playing, 479–483

— C —
color distortion using CopyBits(), 406
defined, 378
drawing in windows with, 401–403
problems using CopyBits() from offscreen bitmaps or,
sound (continued)
from disk, 472-473
installing sampled sound and playing notes with it, 478
multiple sounds, 474-475
number of sound channels that can play sounds concurrently, 475
from a point other than the beginning, 474
setting callback and completion routines for asynchronous sounds, 479
'snd' resources, 470-473
sounds that won't fit in memory, 472
stopping playback, 475-477
synchronizing sound played by sound channels, 483-485
recording, 491-504
from applications, 491-492
asynchronously, 501-504
creating AIFF files from sampled data, 497-500
creating 'sod' resources from sampled data, 495-497
determining elapsed time and remaining time of asynchronous sounds, 500-501
without standard recording dialog box, 492-495
volume, 485-491
balance control (panning), 488
multiple sound channels and, 491
retrieving and setting computer's default volume, 485-487
setting sound channel volume, 488-491
Sound Manager, 504-507
calculating sample rates with getRateCmd command, 507
disposing of sound channels when playing sound asynchronously, 506
disposing of sound handles and sound channels in completion routines, 506
executing sound commands, 504
retrieving 'snd' resources, 507
sound commands listed, 468-470
storing more than 128 commands in a sound channel, 505-506
unlocking and releasing sound resources, 506
when sound channel callback routines are executed, 507
Speech Manager, 509-513
determining when speech channel has completed speaking, 510
retrieving list of available voices, 511
selecting voices, 511-512
speaking a buffer of text, 509-510
stopping speech on channels, 510
synchronizing speech from speech channels, 512-513
Standard File Package, 153-154, 183-193
current directory and volume in File Manager calls versus, 153-154
customizing, 192-193
determining if Cancel button in standard file dialog box is pressed, 185
dialog hook function versus filter function in CustomGetFile() and CustomPutFile(), 184
displaying all file types in StandardGetFile() dialog box, 188
displaying only visible files and folders in standard file dialog boxes, 188
opening files selected by user with StandardGetFile(), 184-185
recognizing sfHookFirstCall and sfHookLastCall pseudo-item numbers, 193
rescanning directories and displaying new contents, 187-188
retrieving current folder and volume displayed by, 191-192
selecting files displayed in Standard Get File routine file lists, 186
selecting folders with, 193
setting current volume and folder for Standard File Package calls, 189-191
See also File Manager
strings, 516-520
converting C strings and Pascal strings, 516-517
converting numbers and strings, 519-520
converting SStr255 and OSType strings, 517
converting to uppercase or lowercase, 518
finding functions for Pascal strings, 518
retrieving handles of strings, 517
See also fonts; text
System extensions (INITs), 528-533
communicating and sharing data with control panels, 532-533
controlling execution of, 532
creating, 529-531
defined, 528
displaying icon for at startup, 531-532
how System extensions tell when INIT time is over, 531
PicHandle problems at INIT time, 533
system folders. See File Manager
System heap. See memory

— T —
text, drawing, 549-559
changing font, style, and size of drawn text, 550-551
determining text characteristics used by graphics ports to draw text, 551-552
determining width of text from font attributes, 552
dimmed (grayed-out) text, 553-554
PenMode settings and, 552
retrieving and changing current textMode, 553
rotated text, 554-558
with transfer modes, 552
why text drawn with font IDs looks different on different computers, 548
in windows, 549-550
See also fonts; strings
TextEdit fields, 561-571
changing caret blinking rate, 571
creating scroll bars in, 568-569
determining if fields are style-aware, 565
determining line height of, 565-566
displaying more than 32K of text, 568
displaying static text of more than 255 characters, 562
drawing blocks of text in specified rectangular areas, 562
drawing text over pictures with TEXTBox(), 567-568
making TextEdit caret blink, 571
processing mouse clicks in, 566
retrieving start of each line in, 567
retrieving text of, 563
retrieving text and styles from multistyle fields, 563–564
setting cursor position in, 571
setting selection range in, 570–571
setting text styles in, 566
TESetText() and changes to text, 567
updating scroll bars in response to dragging outside the window, 569–570
word wrapping in, 567
Time Manager, 349–362
accessing global variables from 68K tasks, 360–361
accessing global variables from native PowerPC tasks, 361–362
determining time required for routines to run, 352–353
executing tasks, 354–360
in 68K applications, 355–357
crashes caused when exiting applications after installing tasks, 354
in native PowerPC applications, 357–358
periodically, 359
running tasks after applications quit, 359
stopping periodic tasks, 359–360
when installing application is switched out or no longer controls CPU, 355
InstTime() versus InstXTime(), 354
PrimeTime and installing tasks, 350
See also Notification Manager; Process Manager; Vertical Retrace Manager
traps, 526–528
defined, 526–527
determining if System routines are available, 538–539
effect on applications of patching traps, 541–542
finding trap numbers for routines, 537
head patches versus tail patches, 528
identifying Operating System versus Toolbox trap routines, 540–541
moving memory with patch routines, 543
patching traps from two applications simultaneously, 542–543
retrieving keydown and mouse events in all applications, 543–544
retrieving trap addresses, 537–538
Unimplemented() routine and, 527

- U -

universal procedure pointers. See Power Macintosh programming

- V -

Vertical Retrace Manager, 363–376
accessing application global variables from 68K VBL tasks, 372–373
accessing application global variables from native PowerPC VBL tasks, 373
determining slot number of screens, 374
disabling VBL tasks, 375
executing VBL tasks
68K VBL tasks, 365–367
after applications quit, 369
how often VBL tasks are executed, 365
native PowerPC VBL tasks, 367–368
periodically, 371–372
slot-based VBL tasks, 368–369
when installing application is switched out or no longer controls CPU, 369–371
retrieving slot number of main monitor, 374–375
synchronizing drawings to VBL tasks for smooth animation, 375–376
system-based VBL tasks versus slot-based VBL tasks, 364
VBL tasks versus Time Manager tasks, 350
Vertical Blinking (VBL) interrupts defined, 364
See also Notification Manager; Process Manager; Time Manager
time delay resolution settings, 351
Vertical Retrace Manager tasks versus VBL tasks, 350
Vertical Blinking (VBL) interrupts defined, 364
See also Notification Manager; Process Manager; Time Manager
volume controls. See sound volumes. See File Manager

- W -

windows, 573–600
changing cursor shape over areas of, 42–43
code structure, 597–600
determining if System routines are available, 538–539
effect on applications of patching traps, 541–542
finding trap numbers for routines, 537
generating OS versus Toolbox trap routines, 540–541
moving memory with patch routines, 543
patching traps from two applications simultaneously, 542–543
retrieving keydown and mouse events in all applications, 543–544
retrieving trap addresses, 537–538
Unimplemented() routine and, 527
drawing in windows with GWorlds, 401–403
passing pixmaps of color windows to CopyBits(), 400
windows (continued)
refreshing controls when windows come to front, 92–93
RGB values for dimmed window titles, 19–20
updating and drawing on windows, 589–594
activating with SelectWindow(), 593
copying contents between windows, 592
creating color backgrounds, 593–594
determining what needs to be redrawn in response to
update events, 593
drawing in color in windows, 592
forcing windows to redraw, 589
origin point of windows, 589
retrieving GrafPort of the Desktop, 591–592
selecting which window to draw to, 593
setting custom colors, 590–591
window definition procedure (WDEF) defined, 575
See also dialog boxes; scroll bars
CD-ROM Disc
Instructions

The CD-ROM contains a database of the information that's presented in the book. Along with this data, we've included a special FAQ DB Viewer program that enables you to search the FAQs by keyword(s) and to view the contents of the database.

Starting the FAQ DB Viewer Program

To run the FAQ DB Viewer program, follow these steps:

1. Place the CD-ROM in the CD drive. The CD-ROM icon appears on the desktop.
2. Double-click the CD-ROM icon. The Mac Programming FAQs folder appears.
3. Double-click the folder
4. Double-click the FAQ Database folder
5. Double-click the FAQ DB Viewer icon. The Main FAQ screen appears, displaying FAQ 1-1.

Refer to Appendix A for complete instructions on using the database and searching for FAQs by keyword(s) or by topic.

Using the Sample Code and Resource Files

We’ve also included two folders that contain sample code and resource files:

- **FAQ Code Projects**: The FAQ Code Projects folder contains CodeWarrior and Symantec projects that enable you to build applications which are based on the longer — and more interesting — sample code in the book. The contents of the FAQ Code Projects folder include subfolders that represent different chapters. The contents of each chapter subfolder include other subfolders that contain the individual projects. Each project includes the complete source, resource, and project files that are necessary to build executables, as well as a pre-built executable.

- **FAQ Code Snippets**: The contents of the FAQ Code Snippets folder include 27 SimpleText files that contain sample code from the book. Each file represents a different chapter; the code is organized by its associated FAQ.
IDG BOOKS WORLDWIDE, INC.
END-USER LICENSE AGREEMENT

Read This. You should carefully read these terms and conditions before opening the software packet(s) included with this book ("Book"). This is a license agreement ("Agreement") between you and IDG Books Worldwide, Inc. ("IDGB"). By opening the accompanying software packet(s), you acknowledge that you have read and accept the following terms and conditions. If you do not agree and do not want to be bound by such terms and conditions, promptly return the Book and the unopened software packet(s) to the place you obtained them for a full refund.

1. License Grant. IDGB grants to you (either an individual or entity) a nonexclusive license to use one copy of the enclosed software program(s) (collectively, the "Software") solely for your own personal or business purposes on a single computer (whether a standard computer or a workstation component of a multi-user network). The Software is in use on a computer when it is loaded into temporary memory (i.e., RAM) or installed into permanent memory (e.g., hard disk, CD-ROM or other storage device). IDGB reserves all rights not expressly granted herein.

2. Ownership. IDGB is the owner of all right, title, and interest, including copyright, in and to the compilation of the Software recorded on the disk(s)/CD-ROM. Copyright to the individual programs on the disk(s)/CD-ROM is owned by the author or other authorized copyright owner of each program. Ownership of the Software and all proprietary rights relating thereto remain with IDGB and its licensors.

3. Restrictions On Use and Transfer.

(a) You may only (i) make one copy of the Software for backup or archival purposes, or (ii) transfer the Software to a single hard disk, provided that you keep the original for backup or archival purposes. You may not (i) rent or lease the Software, (ii) copy or reproduce the Software through a LAN or other network system or through any computer subscriber system or bulletin-board system, or (iii) modify, adapt, or create derivative works based on the Software.

(b) You may not reverse engineer, decompile, or disassemble the Software. You may transfer the Software and user documentation on a permanent basis, provided that the transferee agrees to accept the terms and conditions of this Agreement and
you retain no copies. If the Software is an update or has been updated, any transfer must include the most recent update and all prior versions.

4. **Restrictions on Use of Individual Programs.** You must follow the individual requirements and restrictions detailed for each individual program in Appendix A of this Book. These limitations are contained in the individual license agreements recorded on the disk(s)/CD-ROM. These restrictions include a requirement that after using the program for the period of time specified in its text, the user must pay a registration fee or discontinue use. By opening the Software packet(s), you will be agreeing to abide by the licenses and restrictions of these individual programs. None of the material on this disk(s) or listed in this Book may ever be distributed, in original or modified form, for commercial purposes.

5. **Limited Warranty.**

   (a) IDGB warrants that the Software and disk(s)/CD-ROM are free from defects in materials and workmanship under normal use for a period of sixty (60) days from the date of purchase of this Book. If IDGB receives notification within the warranty period of defects in materials or workmanship, IDGB will replace the defective disk(s)/CD-ROM.

   (b) **IDGB AND THE AUTHOR OF THE BOOK DISCLAIM ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE PROGRAMS, THE SOURCE CODE CONTAINED THEREIN, AND/OR THE TECHNIQUES DESCRIBED IN THIS BOOK. IDGB DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR FREE.**

   (c) This limited warranty gives you specific legal rights, and you may have other rights which vary from jurisdiction to jurisdiction.

6. **Remedies.**

   (a) IDGB’s entire liability and your exclusive remedy for defects in materials and workmanship shall be limited to replacement of the Software, which is returned to IDGB at the address set forth below with a copy of your receipt. This Limited
Warranty is void if failure of the Software has resulted from accident, abuse, or misapplication. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

(b) In no event shall IDGB or the author be liable for any damages whatsoever (including without limitation damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use the Book or the Software, even if IDGB has been advised of the possibility of such damages.

(c) Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation or exclusion may not apply to you.

7. U.S. Government Restricted Rights. Use, duplication, or disclosure of the Software by the U.S. Government is subject to restrictions stated in paragraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013, and in subparagraphs (a) through (d) of the Commercial Computer—Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR supplement, when applicable.

8. General. This Agreement constitutes the entire understanding of the parties, and revokes and supersedes all prior agreements, oral or written, between them and may not be modified or amended except in a writing signed by both parties hereto which specifically refers to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict herewith. If any one or more provisions contained in this Agreement are held by any court or tribunal to be invalid, illegal, or otherwise unenforceable, each and every other provision shall remain in full force and effect.

Replacement CD-ROM Disc Available. If a replacement CD-ROM disc is needed, please write to the following address: Disk Fulfillment Department, Attn: Macworld Mac Programming FAQs, IDG Books Worldwide, Inc., 7260 Shadeland Station, Indianapolis, IN 46256, or call 1-800-762-2974. Please specify the size of disk you need and allow 3 to 4 weeks for delivery.
It's one-stop shopping for Mac programmers looking for answers to the most common questions! Got a problem? Thinking of going on a wild goose chase from book to book, or searching online for answers? Forget it! Mac experts Stephen Baker and Dave Mark have put all the information you need into one convenient place. In the pages of this book and in its accompanying CD, you’ll find the answers to 650 of the most common questions Mac programmers like you encounter. And everything's arranged by Toolbox Manager—organized just the way Mac programmers think! Loaded with specific solutions to real problems and written without fluff, this is your best source for the hard-to-find tips, techniques, and secrets that can make your programming life a lot easier.

Quick and easy access to the answers Mac programmers need most

- Loaded with specific solutions—full of practical, not theoretical, information
- Solutions that work for any of the top programming tools—Metrowerks CodeWarrior, Symantec C, and others
- Values for the Gestalt calls—a key element of any Mac program—are listed in an appendix for easy referencing of key information
- Written in straightforward terms that clearly lay out the solutions—so answers are easy to find, easy to read
- Cross-references to related information in the book lead you through a chain of information within the text
- Scores of hard-to-find tips, techniques, and secrets used by Mac experts Stephen Baker and Dave Mark!

About the Authors

Stephen H. Baker is an experienced Mac programmer and program designer. He is currently working as Director of Release Management at Visix Software, Inc., a cross-platform development company.

Dave Mark is the author of bestselling Mac programming titles and writes a popular column in MacTech magazine. He co-hosts OnLine Today, a New York-based, nationally syndicated radio show, and he's been featured at the Macworld Expo on panels since 1988. He is currently working in publisher relations at Metrowerks.