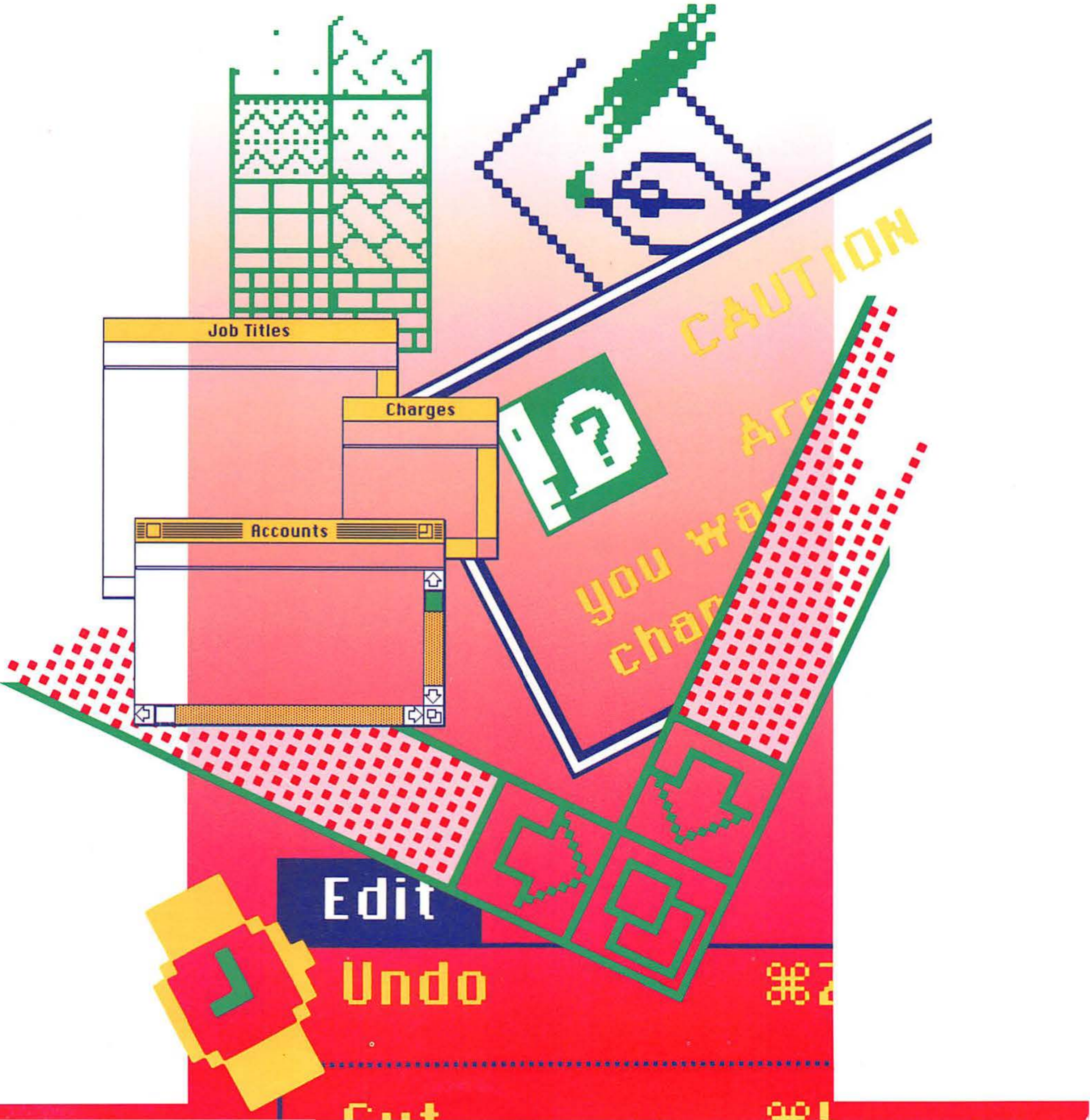
Apple® Inside Macintosh Volume III



Inside Macintosh®
Volume III

Addison-Wesley Publishing Company, Inc.
Reading, Massachusetts Menlo Park, California New York
Don Mills, Ontario Wokingham, England Amsterdam Bonn
Sydney Singapore Tokyo Madrid San Juan

© Apple Computer, Inc., 1985
20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010

Simultaneously published in the United States and Canada.

Written by Caroline Rose with Bradley Hacker, Robert Anders, Katie Withey, Mark Metzler, Steve Chernicoff, Chris Espinosa, Andy Averill, Brent Davis, and Brian Howard, assisted by Sandy Tompkins-Leffler and Louella Pizzuti. Special thanks to Cary Clark and Scott Knaster.

This book was produced using the Apple Macintosh computer and the LaserWriter printer.

ISBN 0-201-17733-1
KLMNOPQRST-MU-93210
11th printing, June 1990

Inside Macintosh
Volume III

Contents

PREFACE

ABOUT INSIDE MACINTOSH

Inside Macintosh is a three-volume set of manuals that tells you what you need to know to write software for the Apple® Macintosh™ 128K, 512K, or XL (or a Lisa® running MacWorks™ XL). Although directed mainly toward programmers writing standard Macintosh applications, *Inside Macintosh* also contains the information needed to write simple utility programs, desk accessories, device drivers, or any other Macintosh software. It includes:

- the user interface guidelines for applications on the Macintosh
- a complete description of the routines available for your program to call (both those built into the Macintosh and others on disk), along with related concepts and background information
- a description of the Macintosh 128K and 512K hardware

It does *not* include information about:

- Programming in general.
- Getting started as a developer. For this, write to:

  Developer Relations
  Mail Stop 27-S
  Apple Computer, Inc.
  20525 Mariani Avenue
  Cupertino, CA 95014
- Any specific development system, except where indicated. You'll need to have additional documentation for the development system you're using.
- The Standard Apple Numeric Environment (SANE), which your program can access to perform extended-precision floating-point arithmetic and transcendental functions. This environment is described in the *Apple Numerics Manual.*

You should already be familiar with the basic information that's in *Macintosh*, the owner's guide, and have some experience using a standard Macintosh application (such as MacWrite™).

The Language

The routines you'll need to call are written in assembly language, but (with a few exceptions) they're also accessible from high-level languages, such as Pascal on the Lisa Workshop development system. *Inside Macintosh* documents the Lisa Pascal interfaces to the routines and the symbolic names defined for assembly-language programmers using the Lisa Workshop; if you're using a different development system, its documentation should tell you how to apply the information presented here to that system.

Inside Macintosh is intended to serve the needs of both high-level language and assembly-language programmers. Every routine is shown in its Pascal form (if it has one), but assembly-language programmers are told how they can access the routines. Information of interest only to assembly-language programmers is isolated and labeled so that other programmers can conveniently skip it.

Familiarity with Lisa Pascal (or a similar high-level language) is recommended for all readers, since it's used for most examples. Lisa Pascal is described in the documentation for the Lisa Pascal Workshop.

What's in Each Volume

Inside Macintosh consists of three volumes. Volume I begins with the following information of general interest:

- a "road map" to the software and the rest of the documentation
- the user interface guidelines
- an introduction to memory management (the least you need to know, with a complete discussion following in Volume II)
- some general information for assembly-language programmers

It then describes the various parts of the **User Interface Toolbox**, the software in ROM that helps you implement the standard Macintosh user interface in your application. This is followed by descriptions of other, RAM-based software that's similar in function to the User Interface Toolbox. (The software overview in the Road Map chapter gives further details.)

Volume II describes the **Operating System**, the software in ROM that does basic tasks such as input and output, memory management, and interrupt handling. As in Volume I, some functionally similar RAM-based software is then described.

Volume III discusses your program's interface with the Finder and then describes the Macintosh 128K and 512K hardware. A comprehensive summary of all the software is provided, followed by some useful appendices and a glossary of all terms defined in *Inside Macintosh*.

Version Numbers

This edition of *Inside Macintosh* describes the following versions of the software:

- version 105 of the ROM in the Macintosh 128K or 512K
- version 112 of the ROM image installed by MacWorks in the Macintosh XL
- version 1.1 of the Lisa Pascal interfaces and the assembly-language definitions

Some of the RAM-based software is read from the file named System (usually kept in the System Folder). This manual describes the software in the System file whose creation date is May 2, 1984.

A HORSE OF A DIFFERENT COLOR

On an innovative system like the Macintosh, programs don't look quite the way they do on other systems. For example, instead of carrying out a sequence of steps in a predetermined order, your program is driven primarily by user actions (such as clicking and typing) whose order cannot be predicted.

You'll probably find that many of your preconceptions about how to write applications don't apply here. Because of this, and because of the sheer volume of information in *Inside Macintosh*, it's essential that you read the Road Map chapter. It will help you get oriented and figure out where to go next.

THE STRUCTURE OF A TYPICAL CHAPTER

Most chapters of *Inside Macintosh* have the same structure, as described below. Reading through this now will save you a lot of time and effort later on. It contains important hints on how to find what you're looking for within this vast amount of technical documentation.

Every chapter begins with a very brief description of its subject and a list of what you should already know before reading that chapter. Then there's a section called, for example, "About the Window Manager", which gives you more information about the subject, telling you what you can do with it in general, elaborating on related user interface guidelines, and introducing terminology that will be used in the chapter. This is followed by a series of sections describing important related concepts and background information; unless they're noted to be for advanced programmers only, you'll have to read them in order to understand how to use the routines described later.

Before the routine descriptions themselves, there's a section called, for example, "Using the Window Manager". It introduces you to the routines, telling you how they fit into the general flow of an application program and, most important, giving you an idea of which ones you'll need to use. Often you'll need only a few routines out of many to do basic operations; by reading this section, you can save yourself the trouble of learning routines you'll never use.

Then, for the details about the routines, read on to the next section. It gives the calling sequence for each routine and describes all the parameters, effects, side effects, and so on.

Following the routine descriptions, there may be some sections that won't be of interest to all readers. Usually these contain information about advanced techniques, or behind the scenes details for the curious.

For review and quick reference, each chapter ends with a summary of the subject matter, including the entire Pascal interface and a separate section for assembly-language programmers.

CONVENTIONS

The following notations are used in *Inside Macintosh* to draw your attention to particular items of information:

Note: A note that may be interesting or useful

Warning: A point you need to be cautious about

Assembly-language note: A note of interest to assembly-language programmers only

---

[Not in ROM]

Routines marked with this notation are not part of the Macintosh ROM. Depending on how the interfaces have been set up on the development system you're using, these routines may or may not be available. They're available to users of Lisa Pascal; other users should check the documentation for their development system for more information. (For related information of interest to assembly-language programmers, see chapter 4 of Volume I.)

1 THE FINDER INTERFACE

ABOUT THIS CHAPTER

This chapter describes the interface between a Macintosh application program and the Finder. You should already be familiar with the details of the User Interface Toolbox and the Operating System.

SIGNATURES AND FILE TYPES

Every application must have a unique **signature** by which the Finder can identify it. The signature can be any four-character sequence not being used for another application on any currently mounted volume (except that it can't be one of the standard resource types). To ensure uniqueness on all volumes, you must register your application's signature by writing to:

Macintosh Technical Support
Mail Stop 3-T
Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014

Note: There's no need to register your own resource types, since they'll usually exist only in your own applications or documents.

Signatures work together with **file types** to enable the user to open or print a document (any file created by an application) from the Finder. When the application creates a file, it sets the file's creator and file type. Normally it sets the creator to its signature and the file type to a four-character sequence that identifies files of that type. When the user asks the Finder to open or print the file, the Finder starts up the application whose signature is the file's creator and passes the file type to the application along with other identifying information, such as the file name. (More information about this process is given in chapter 2 of Volume II.)

An application may create its own special type or types of files. Like signatures, file types must be registered with Macintosh Technical Support to ensure uniqueness. When the user chooses Open from an application's File menu, the application will display (via the Standard File Package) the names of all files of a given type or types, regardless of which application created the files. Having a unique file type for your application's special files ensures that only the names of those files will be displayed for opening.

Note: Signatures and file types may be strange, unreadable combinations of characters; they're never seen by end users of Macintosh.

Applications may also create existing types of files. There might, for example, be an application that merges two MacWrite documents into a single document. In such cases, the application should use the same file type as the original application uses for those files. It should also specify the original application's signature as the file's creator; that way, when the user asks the Finder to open or print the file, the Finder will call on the original application to perform the operation. To learn the signature and file types used by an existing application, check with the application's manufacturer.

Files that consist only of text—a stream of characters, with Return characters at the ends of paragraphs or short lines—should be given the standard file type 'TEXT'. This is the type that MacWrite gives to text only files it creates, for example. If your application uses this file type, its files will be accepted by MacWrite and it in turn will accept MacWrite text-only files (likewise for any other application that deals with 'TEXT' files, such as MacTerminal). Your application can give its own signature as the file's creator if it wants to be called to open or print the file when the user requests this from the Finder.

For files that aren't to be opened or printed from the Finder, as may be the case for certain data files created by the application, the creator should be set to '????' (and the file type to whatever is appropriate).

FINDER-RELATED RESOURCES

To establish the proper interface with the Finder, every application's resource file must specify the signature of the application along with data that provides version information. In addition, there may be resources that provide information about icons and files related to the application. All of these Finder-related resources are described below, followed by a comprehensive example and (for interested programmers) the exact formats of the resources.

Version Data

Your application's resource file must contain a special resource that has the signature of the application as its resource type. This resource is called the **version data** of the application. The version data is typically a string that gives the name, version number, and date of the application, but it can in fact be any data at all. The resource ID of the version data is 0 by convention.

Part of the process of installing an application on the Macintosh is to set the creator of the file that contains the application. You set the creator to the application's signature, and the Finder copies the corresponding version data into a resource file named **Desktop**. (The Finder doesn't display this file on the Macintosh desktop, to ensure that the user won't tamper with it.)

> **Note:** Additional, related resources may be copied into the Desktop file; see "Bundles" below for more information.

Icons and File References

For each application, the Finder needs to know:

- the icon to be displayed for the application on the desktop, if different from the Finder's default icon for applications (see Figure 1)
- if the application creates any files, the icon to be displayed for each type of file it creates, if different from the Finder's default icon for documents

The Finder learns this information from resources called **file references** in the application's resource file. Each file reference contains a file type and an ID number, called a **local ID**, that identifies the icon to be displayed for that type of file. (The local ID is mapped to an actual resource ID as described under "Bundles" below.)

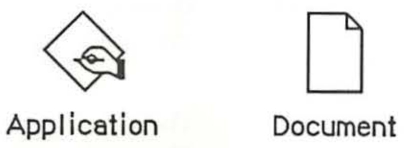


Figure 1. The Finder's Default Icons

The file type for the application itself is 'APPL'. This is the file type in the file reference that designates the application's icon. You also specify it as the application's file type at the same time that you specify its creator—when you install the application on the Macintosh.

The ID number in a file reference corresponds not to a single icon but to an **icon list** in the application's resource file. The icon list consists of two icons: the actual icon to be displayed on the desktop, and a mask consisting of that icon's outline filled with black (see Figure 2).

Figure 2. Icon and Mask

Bundles

A bundle in the application's resource file groups together all the Finder-related resources. It specifies the following:

- the application's signature and the resource ID of its version data
- a mapping between the local IDs for icon lists (as specified in file references) and the actual resource IDs of the icon lists in the resource file
- local IDs for the file references themselves and a mapping to their actual resource IDs

When you install the application on the Macintosh, you set its "bundle bit"; the first time the Finder sees this, it copies the version data, bundle, icon lists, and file references from the application's resource file into the Desktop file. If there are any resource ID conflicts between the icon lists and file references in the application's resource file and those in Desktop, the Finder will change those resource IDs in Desktop. The Finder does this same resource copying and ID conflict resolution when you transfer an application to another volume.

Note: The local IDs are needed only for use by the Finder.

An Example

Suppose you've written an application named SampWriter. The user can create a unique type of document from it, and you want a distinctive icon for both the application and its documents. The application's signature, as recorded with Macintosh Technical Support, is 'SAMP'; the file type assigned for its documents is 'SAMF'. You would include the following resources in the application's resource file:

Resource	**Resource ID**	**Description**
Version data with resource type 'SAMP'	0	The string 'SampWriter Version 1--2/1/85'
Icon list	128	The icon for the application The icon's mask
Icon list	129	The icon for documents The icon's mask
File reference	130	File type 'APPL' Local ID 0 for the icon list
File reference	131	File type 'SAMF' Local ID 1 for the icon list
Bundle	132	Signature 'SAMP' Resource ID 0 for the version data For icon lists, the mapping: local ID 0 → resource ID 128 local ID 1 → resource ID 129 For file references, the mapping: local ID 2 → resource ID 130 local ID 3 → resource ID 131

Note: See the documentation for the development system you're using for information about how to include these resources in a resource file.

Formats of Finder-Related Resources

The resource type for an application's version data is the signature of the application, and the resource ID is 0 by convention. The resource data can be anything at all; typically it's a string giving the name, version number, and date of the application.

The resource type for an icon list is 'ICN#'. The resource data simply consists of the icons, 128 bytes each.

The resource type for a file reference is 'FREF'. The resource data has the format shown below.

Number of bytes	**Contents**
4 bytes	File type
2 bytes	Local ID for icon list

The resource type for a bundle is 'BNDL'. The resource data has the format shown below. The format is more general than needed for Finder-related purposes because bundles will be used in other ways in the future.

Number of bytes	Contents
4 bytes	Signature of the application
2 bytes	Resource ID of version data
2 bytes	Number of resource types in bundle minus 1
For each resource type:	
4 bytes	Resource type
2 bytes	Number of resources of this type minus 1
For each resource:	
2 bytes	Local ID
2 bytes	Actual resource ID

A bundle used for establishing the Finder interface contains the two resource types 'ICN#' and 'FREF'.

2 THE MACINTOSH HARDWARE

ABOUT THIS CHAPTER

This chapter provides a basic description of the hardware of the Macintosh 128K and 512K computers. It gives you information that you'll need to connect other devices to the Macintosh and to write device drivers or other low-level programs. It will help you figure out which technical documents you'll need to design peripherals; in some cases, you'll have to obtain detailed specifications from the manufacturers of the various interface chips.

This chapter is oriented toward assembly-language programmers. It assumes you're familiar with the basic operation of microprocessor-based devices. Knowledge of the Macintosh Operating System will also be helpful.

Warning: Only the Macintosh 128K and 512K are covered in this chapter. In particular, note that the memory addresses and screen size are different on the Macintosh XL (and may be different in future versions of the Macintosh). To maintain software compatibility across the Macintosh line, and to allow for future changes to the hardware, you're *strongly advised* to use the Toolbox and Operating System routines wherever possible.

To learn how your program can determine which hardware environment it's operating in, see the description of the Environs procedure in chapter 13 of Volume II.

OVERVIEW OF THE HARDWARE

The Macintosh computer contains a Motorola MC68000 microprocessor clocked at 7.8336 megahertz, random access memory (RAM), read-only memory (ROM), and several chips that enable it to communicate with external devices. There are five I/O devices: the video display; the sound generator; a Synertek SY6522 Versatile Interface Adapter (VIA) for the mouse and keyboard; a Zilog Z8530 Serial Communications Controller (SCC) for serial communication; and an Apple custom chip, called the **IWM** ("Integrated Woz Machine") for disk control.

The Macintosh uses memory-mapped I/O, which means that each device in the system is accessed by reading or writing to specific locations in the address space of the computer. Each device contains logic that recognizes when it's being accessed and responds in the appropriate manner.

The MC68000 can directly access 16 megabytes (Mb) of address space. In the Macintosh, this is divided into four equal sections. The first four Mb are for RAM, the second four Mb are for ROM, the third are for the SCC, and the last four are for the IWM and the VIA. Since each of the devices within the blocks has far fewer than four Mb of individually addressable locations or registers, the addresses within each block "wrap around" and are repeated several times within the block.

RAM is the "working memory" of the system. Its base address is address 0. The first 256 bytes of RAM (addresses 0 through $FF) are used by the MC68000 as **exception vectors**; these are the addresses of the routines that gain control whenever an exception such as an interrupt or a trap occurs. (The summary at the end of this chapter includes a list of all the exception vectors.) RAM also contains the system and application heaps, the stack, and other information used by applications. In addition, the following hardware devices share the use of RAM with the MC68000:

- the video display, which reads the information for the display from one of two **screen buffers**
- the sound generator, which reads its information from one of two **sound buffers**
- the disk speed controller, which shares its data space with the sound buffers

The MC68000 accesses to RAM are interleaved (alternated) with the video display's accesses during the active portion of a screen scan line (video scanning is described in the next section). The sound generator and disk speed controller are given the first access after each scan line. At all other times, the MC68000 has uninterrupted access to RAM, increasing the average RAM access rate to about 6 megahertz (MHz).

ROM is the system's permanent read-only memory. Its base address, $400000, is available as the constant romStart and is also stored in the global variable ROMBase. ROM contains the routines for the Toolbox and Operating System, and the various system traps. Since the ROM is used exclusively by the MC68000, it's always accessed at the full processor rate of 7.83 MHz.

The address space reserved for the device I/O contains blocks devoted to each of the devices within the computer. This region begins at address $800000 and continues to the highest address at $FFFFFF.

Note: Since the VIA is involved in some way in almost every operation of the Macintosh, the following sections frequently refer to the VIA and VIA-related constants. The VIA itself is described later, and all the constants are listed in the summary at the end of this chapter.

THE VIDEO INTERFACE

The video display is created by a moving electron beam that scans across the screen, turning on and off as it scans in order to create black and white pixels. Each pixel is a square, approximately 1/74 inch on a side.

To create a screen image, the electron beam starts at the top left corner of the screen (see Figure 1). The beam scans horizontally across the screen from left to right, creating the top line of graphics. When it reaches the last pixel on the right end of the top line it turns off, and continues past the last pixel to the physical right edge of the screen. Then it flicks invisibly back to the left edge and moves down one scan line. After tracing across the black border, it begins displaying the data in the second scan line. The time between the display of the rightmost pixel on one line and the leftmost pixel on the next is called the **horizontal blanking interval.** When the electron beam reaches the last pixel of the last (342nd) line on the screen, it traces out to the right edge and then flicks up to the top left corner, where it traces the left border and then begins once again to display the top line. The time between the last pixel on the bottom line and the first one on the top line is called the **vertical blanking interval.** At the beginning of the vertical blanking interval, the VIA generates a **vertical blanking interrupt.**

The pixel clock rate (the frequency at which pixels are displayed) is 15.6672 MHz, or about .064 microseconds (μsec) per pixel. For each scan line, 512 pixels are drawn on the screen, requiring 32.68 μsec. The horizontal blanking interval takes the time of an additional 192 pixels, or 12.25 μsec. Thus, each full scan line takes 44.93 μsec, which means the horizontal scan rate is 22.25 kilohertz.

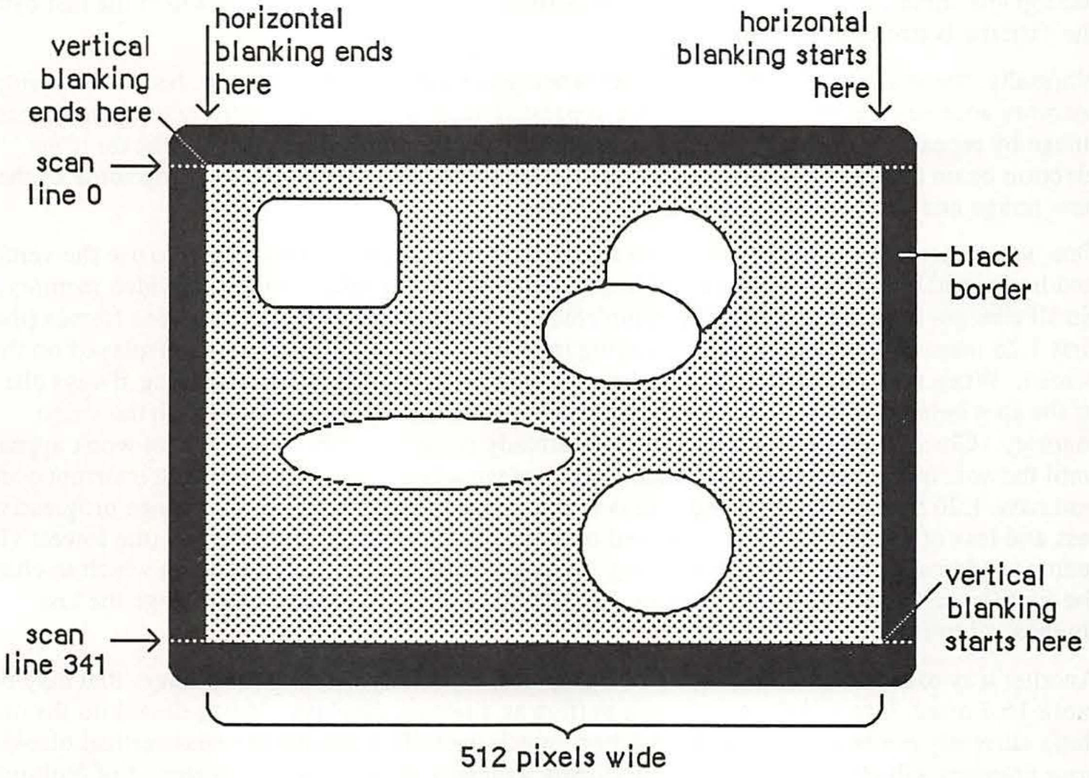

vertical blanking ends here
horizontal blanking ends here
horizontal blanking starts here
scan line 0
black border
vertical blanking starts here
scan line 341
512 pixels wide

Figure 1. Video Scanning Pattern

A full screen display consists of 342 horizontal scan lines, occupying 15367.65 µsec, or about 15.37 milliseconds (msec). The vertical blanking interval takes the time of an additional 28 scan lines—1258.17 µsec, or about 1.26 msec. This means the full screen is redisplayed once every 16625.8 µsec. That's about 16.6 msec per frame, which means the vertical scan rate (the full screen display frequency) is 60.15 hertz.

The video generator uses 21,888 bytes of RAM to compose a bit-mapped video image 512 pixels wide by 342 pixels tall. Each bit in this range controls a single pixel in the image: A 0 bit is white, and a 1 bit is black.

There are two screen buffers (areas of memory from which the video circuitry can read information to create a screen display): the main buffer and the alternate buffer. The starting addresses of the screen buffers depend on how much memory you have in your Macintosh. In a Macintosh 128K, the main screen buffer starts at $1A700 and the alternate buffer starts at $12700; for a 512K Macintosh, add $60000 to these numbers.

Warning: To be sure you don't use the wrong area of memory and to maintain compatibility with future Macintosh systems, you should get the video base address and bit map dimensions from screenBits (see chapter 6 of Volume I).

Each scan line of the screen displays the contents of 32 consecutive words of memory, each word controlling 16 horizontally adjacent pixels. In each word, the high-order bit (bit 15) controls the leftmost pixel and the low-order bit (bit 0) controls the rightmost pixel. The first word in each scan line follows the last word on the line above it. The starting address of the screen is thus in the top left corner, and the addresses progress from there to the right and down, to the last byte in the extreme bottom right corner.

Normally, the video display doesn't flicker when you read from or write to it, because the video memory accesses are interleaved with the processor accesses. But if you're creating an animated image by repeatedly drawing the graphics in quick succession, it may appear to flicker if the electron beam displays it when your program hasn't finished updating it, showing some of the new image and some of the old in the same frame.

One way to prevent flickering when you're updating the screen continuously is to use the vertical and horizontal blanking signals to synchronize your updates to the scanning of video memory. Small changes to your screen can be completed entirely during the interval between frames (the first 1.26 msec following a vertical blanking interrupt), when nothing is being displayed on the screen. When making larger changes, the trick is to keep your changes happening always ahead of the spot being displayed by the electron beam, as it scans byte by byte through the video memory. Changes you make in the memory already passed over by the scan spot won't appear until the *next* frame. If you start changing your image when the vertical blanking interrupt occurs, you have 1.26 msec of unrestricted access to the image. After that, you can change progressively less and less of your image as it's scanned onto the screen, starting from the top (the lowest video memory address). From vertical blanking interrupt, you have only 1.26 msec in which to change the first (lowest address) screen location, but you have almost 16.6 msec to change the last (highest address) screen location.

Another way to create smooth, flicker-free graphics, especially useful with changes that may take more 16.6 msec, is to use the two screen buffers as alternate displays. If you draw into the one that's currently *not* being displayed, and then switch the buffers during the next vertical blanking, your graphics will change all at once, producing a clean animation. (See chapter 11 of Volume II to find out how to specify tasks to be performed during vertical blanking.)

If you want to use the alternate screen buffer, you'll have to specify this to the Segment Loader (see chapter 2 of Volume II for details). To switch to the alternate screen buffer, clear the following bit of VIA data register A (vBase+vBufA):

```
vPage2      .EQU        6      ;0 = alternate screen buffer
```

For example:

```
            BCLR        #vPage2,vBase+vBufA
```

To switch back to the main buffer, set the same bit.

Warning: Whenever you change a bit in a VIA data register, be sure to leave the other bits in the register unchanged.

Warning: The alternate screen buffer may not be supported in future versions of the Macintosh.

THE SOUND GENERATOR

The Macintosh sound circuitry uses a series of values taken from an area of RAM to create a changing waveform in the output signal. This signal drives a small speaker inside the Macintosh and is connected to the external sound jack on the back of the computer. If a plug is inserted into the external sound jack, the internal speaker is disabled. The external sound line can drive a load of 600 or more ohms, such as the input of almost any audio amplifier, but not a directly connected external speaker.

The sound generator may be turned on or off by writing 1 (off) or 0 (on) to the following bit of VIA data register B (vBase+vBufB):

```
vSndEnb      .EQU         7       ;0 = sound enabled, 1 = disabled
```

For example:

```
            BSET          #vSndEnb,vBase+vBufB     ;turn off sound
```

By storing a range of values in the sound buffer, you can create the corresponding waveform in the sound channel. The sound generator uses a form of pulse-width encoding to create sounds. The sound circuitry reads one word in the sound buffer during each horizontal blanking interval (including the "virtual" intervals during vertical blanking) and uses the high-order byte of the word to generate a pulse of electricity whose duration (width) is proportional to the value in the byte. Another circuit converts this pulse into a voltage that's attenuated (reduced) by a three-bit value from the VIA. This reduction corresponds to the current setting of the volume level. To set the volume directly, store a three-bit number in the low-order bits of VIA data register A (vBase+vBufA). You can use the following constant to isolate the bits involved:

```
vSound       .EQU         7        ;sound volume bits
```

Here's an example of how to set the sound level:

```
            MOVE.B      vBase+vBufA,D0      ;get current value of register A
            ANDI.B      #255-vSound,D0      ;clear the sound bits
            ORI.B       #3,D0               ;set medium sound level
            MOVE.B      D0,vBase+vBufA      ;put the data back
```

After attenuation, the sound signal is passed to the audio output line.

The sound circuitry scans the sound buffer at a fixed rate of 370 words per video frame, repeating the full cycle 60.15 times per second. To create sounds with frequencies other than multiples of the basic scan rate, you must store phase-shifted patterns into the sound buffer between each scan. You can use the vertical and horizontal blanking signals (available in the VIA) to synchronize your sound buffer updates to the buffer scan. You may find that it's much easier to use the routines in the Sound Driver to do these functions.

Warning: The low-order byte of each word in the sound buffer is used to control the speed of the motor in the disk drive. Don't store any information there, or you'll interfere with the disk I/O.

There are two sound buffers, just as there are two screen buffers. The address of the main sound buffer is stored in the global variable SoundBase and is also available as the constant soundLow. The main sound buffer is at $1FD00 in a 128K Macintosh, and the alternate buffer is at $1A100; for a 512K Macintosh, add $60000 to these values. Each sound buffer contains 370 words of data. As when you want to use the alternate screen buffer, you'll have to specify to the Segment Loader that you want the alternate buffer (see chapter 2 of Volume II for details). To select the alternate sound buffer for output, clear the following bit of VIA data register A (vBase+vBufA):

```
vSndPg2      .EQU       3      ;0 = alternate sound buffer
```

To return to the main buffer, set the same bit.

> **Warning:** Be sure to switch back to the main sound buffer before doing a disk access, or the disk won't work properly.

> **Warning:** The alternate sound buffer may not be supported in future versions of the Macintosh.

There's another way to generate a simple, square-wave tone of any frequency, using almost no processor intervention. To do this, first load a constant value into all 370 sound buffer locations (use $00's for minumum volume, $FF's for maximum volume). Next, load a value into the VIA's timer 1 latches, and set the high-order two bits of the VIA's auxiliary control register (vBase+vACR) for "square wave output" from timer 1. The timer will then count down from the latched value at 1.2766 µsec/count, over and over, inverting the vSndEnb bit of VIA register B (vBase+vBufB) after each count down. This takes the constant voltage being generated from the sound buffer and turns it on and off, creating a square-wave sound whose period is

2 * 1.2766 µsec * timer 1's latched value

> **Note:** You may want to disable timer 1 interrupts during this process (bit 6 in the VIA's interrupt enable register, which is at vBase+vIER).

To stop the square-wave sound, reset the high-order two bits of the auxiliary control register.

> **Note:** See the SY6522 technical specifications for details of the VIA registers. See also "Sound Driver Hardware" in chapter 8 of Volume II.

Diagram

Figure 2 shows a block diagram for the sound port.

THE SCC

The two serial ports are controlled by a Zilog Z8530 **Serial Communications Controller** (SCC). The port known as SCC port A is the one with the modem icon on the back of the Macintosh. SCC port B is the one with the printer icon.

Macintosh serial ports conform to the EIA standard RS422, which differs from the more common RS232C standard. While RS232C modulates a signal with respect to a common ground ("single-ended" transmission), RS422 modulates two signals against each other ("differential" transmission). The RS232C receiver senses whether the received signal is sufficiently negative with respect to ground to be a logic "1", whereas the RS422 receiver simply senses which line is more negative than the other. This makes RS422 more immune to noise and interference, and more versatile over longer distances. If you ground the positive side of each RS422 receiver and leave unconnected the positive side of each transmitter, you've converted to EIA standard RS423, which can be used to communicate with most RS232C devices over distances up to fifty feet or so.

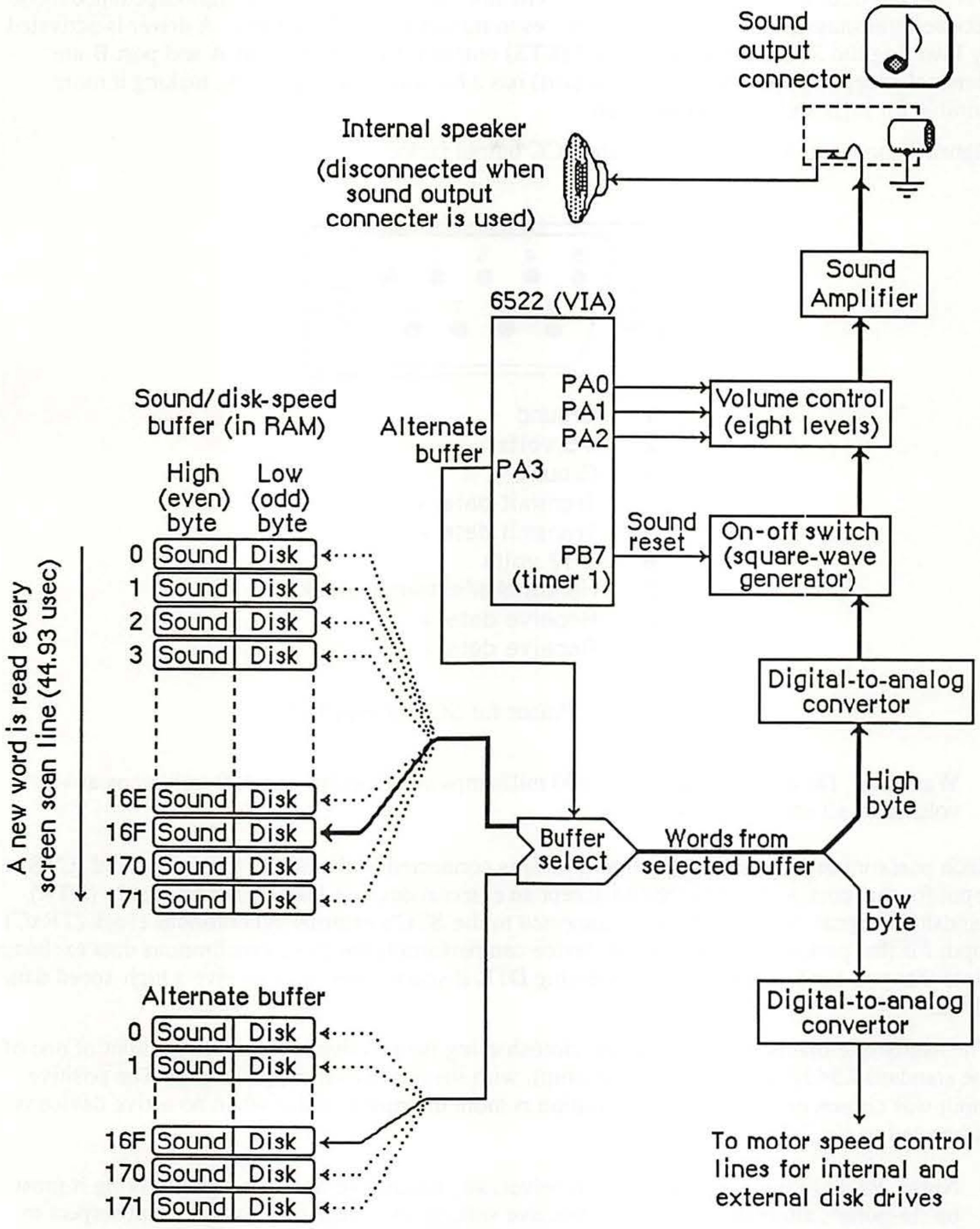
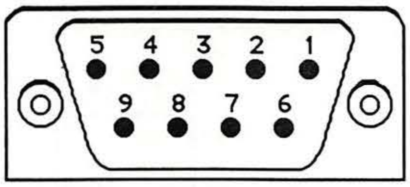

Sound output connector
Internal speaker (disconnected when sound output connecter is used)
Sound Amplifier
6522 (VIA)
PA0
PA1
PA2
PA3
Alternate buffer
Volume control (eight levels)
Sound reset
PB7 (timer 1)
On-off switch (square-wave generator)
Digital-to-analog convertor
High byte
Low byte
Sound/disk-speed buffer (in RAM)
High (even) byte
Low (odd) byte
A new word is read every screen scan line (44.93 usec)
0 Sound Disk
1 Sound Disk
2 Sound Disk
3 Sound Disk
16E Sound Disk
16F Sound Disk
170 Sound Disk
171 Sound Disk
Buffer select
Words from selected buffer
Alternate buffer
0 Sound Disk
1 Sound Disk
16F Sound Disk
170 Sound Disk
171 Sound Disk
Digital-to-analog convertor
To motor speed control lines for internal and external disk drives

Figure 2. Diagram of Sound Port

The serial inputs and outputs of the SCC are connected to the ports through differential line drivers (26LS30) and receivers (26LS32). The line drivers can be put in high-impedance mode between transmissions, to allow other devices to transmit over those lines. A driver is activated by lowering the SCC's Request To Send (RTS) output for that port. Port A and port B are identical except that port A (the modem port) has a higher interrupt priority, making it more suitable for high-speed communication.

Figure 3 shows the DB-9 pinout for the SCC output jacks.



1	Ground
2	+5 volts
3	Ground
4	Transmit data +
5	Transmit data -
6	+12 volts
7	Handshake/external clock
8	Receive data +
9	Receive data -

Figure 3. Pinout for SCC Output Jacks

> **Warning:** Do not draw more than 100 milliamps at +12 volts, and 200 milliamps at +5 volts from all connectors combined.

Each port's input-only handshake line (pin 7) is connected to the SCC's Clear To Send (CTS) input for that port, and is designed to accept an external device's Data Terminal Ready (DTR) handshake signal. This line is also connected to the SCC's external synchronous clock (TRxC) input for that port, so that an external device can perform high-speed synchronous data exchange. Note that you can't use the line for receiving DTR if you're using it to receive a high-speed data clock.

The handshake line is sensed by the Macintosh using the positive (noninverting) input of one of the standard RS422 receivers (26LS32 chip), with the negative input grounded. The positive input was chosen because this configuration is more immune to noise when no active device is connected to pin 7.

> **Note:** Because this is a differential receiver, any handshake or clock signal driving it must be "bi-polar", alternating between a positive voltage and a negative voltage, with respect to the internally grounded negative input. If a device tries to use ground (0 volts) as one of its handshake logic levels, the Macintosh will receive that level as an indeterminate state, with unpredictable results.

The SCC itself (at its PCLK pin) is clocked at 3.672 megahertz. The internal synchronous clock (RTxC) pins for both ports are also connected to this 3.672 MHz clock. This is the clock that, after dividing by 16, is normally fed to the SCC's internal baud-rate generator.

The SCC chip generates level-1 processor interrupts during I/O over the serial lines. For more information about SCC interrupts, see chapter 6 of Volume II.

The locations of the SCC control and data lines are given in the following table as offsets from the constant sccWBase for writes, or sccRBase for reads. These base addresses are also available in the global variables SCCWr and SCCRd. The SCC is on the upper byte of the data bus, so you must use only even-addressed byte reads (a byte read of an odd SCC read address tries to reset the entire SCC). When writing, however, you must use only *odd*-addressed byte writes (the MC68000 puts your data on both bytes of the bus, so it works correctly). A word access to any SCC address will shift the phase of the computer's high-frequency timing by 128 nanoseconds (system software adjusts it correctly during the system startup process).

Location	**Contents**
sccWBase+aData	Write data register A
sccRBase+aData	Read data register A
sccWBase+bData	Write data register B
sccRBase+bData	Read data register B
sccWBase+aCtl	Write control register A
sccRBase+aCtl	Read control register A
sccWBase+bCtl	Write control register B
sccRBase+bCtl	Read control register B

Warning: Don't access the SCC chip more often than once every 2.2 µsec. The SCC requires that much time to let its internal lines stabilize.

Refer to the technical specifications of the Zilog Z8530 for the detailed bit maps and control methods (baud rates, protocols, and so on) of the SCC.

Diagram

Figure 4 shows a circuit diagram for the serial ports.

THE MOUSE

The DB-9 connector labeled with the mouse icon connects to the Apple mouse (Apple II, Apple III, Lisa, and Macintosh mice are electrically identical). The mouse generates four square-wave signals that describe the amount and direction of the mouse's travel. Interrupt-driven routines in the Macintosh ROM convert this information into the corresponding motion of the pointer on the screen. By turning an option called **mouse scaling** on or off in the Control Panel desk accessory, the user can change the amount of screen pointer motion that corresponds to a

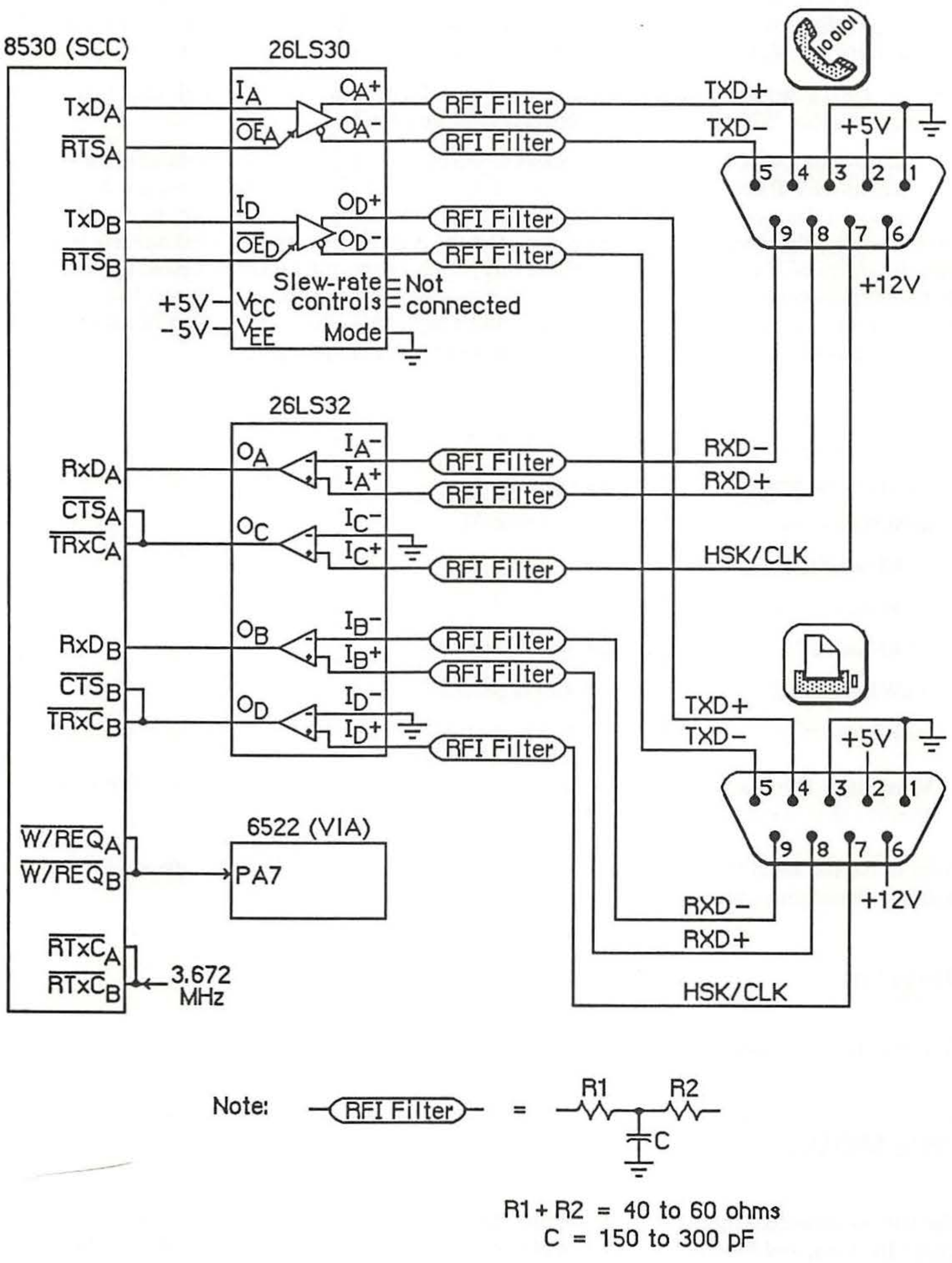


Figure 4. Diagram of Serial Ports

given mouse motion, depending on how fast the mouse is moved; for more information about mouse scaling, see the discussion of parameter RAM in chapter 13 of Volume II.

> **Note:** The mouse is a relative-motion device; that is, it doesn't report where it is, only how far and in which direction it's moving. So if you want to connect graphics tablets, touch screens, light pens, or other absolute-position devices to the mouse port, you must either convert their coordinates into motion information or install your own device-handling routines.

The mouse operates by sending square-wave trains of information to the Macintosh that change as the velocity and direction of motion change. The rubber-coated steel ball in the mouse contacts two capstans, each connected to an interrupter wheel: Motion along the mouse's X axis rotates one of the wheels and motion along the Y axis rotates the other wheel.

The Macintosh uses a scheme known as quadrature to detect which direction the mouse is moving along each axis. There's a row of slots on an interrupter wheel, and two beams of infrared light shine through the slots, each one aimed at a phototransistor detector. The detectors are offset just enough so that, as the wheel turns, they produce two square-wave signals (called the interrupt signal and the quadrature signal) 90 degrees out of phase. The quadrature signal precedes the interrupt signal by 90 degrees when the wheel turns one way, and trails it when the wheel turns the other way.

The interrupt signals, X1 and Y1, are connected to the SCC's DCDA and DCDB inputs, respectively, while the quadrature signals, X2 and Y2, go to inputs of the VIA's data register B. When the Macintosh is interrupted (from the SCC) by the rising edge of a mouse interrupt signal, it checks the VIA for the state of the quadrature signal for that axis: If it's low, the mouse is moving to the left (or down), and if it's high, the mouse is moving to the right (or up). When the SCC interrupts on the falling edge, a high quadrature level indicates motion to the left (or down) and a low quadrature level indicates motion to the right (or up):

SCC	**VIA**	**Mouse**
Mouse interrupt X1 (or Y1)	Mouse quadrature X2 (or Y2)	Motion direction in X (or Y) axis
Positive edge	Low	Left (or down)
	High	Right (or up)
Negative edge	Low	Right (or up)
	High	Left (or down)

Figure 5 shows the interrupt (Y1) and quadrature (Y2) signals when the mouse is moved downwards.

The switch on the mouse is a pushbutton that grounds pin 7 on the mouse connector when pressed. The state of the button is checked by software during each vertical blanking interrupt. The small delay between each check is sufficient to debounce the button. You can look directly at the mouse button's state by examining the following bit of VIA data register B (vBase+vBufB):

```
vSW         .EQU        3      ;0 = mouse button is down
```

If the bit is clear, the mouse button is down. However, it's recommended that you let the Operating System handle this for you through the event mechanism.

Figure 6 shows the DB-9 pinout for the mouse jack at the back of the Macintosh.

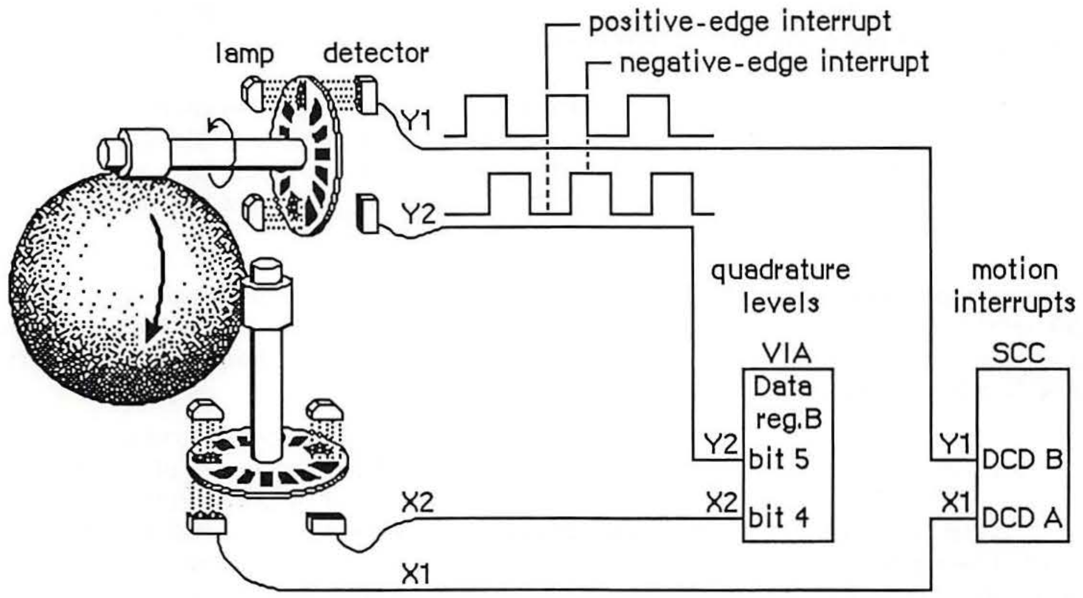
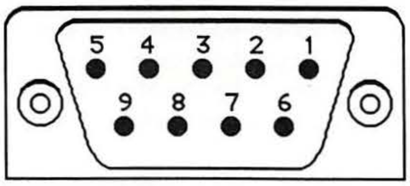


Figure 5. Mouse Mechanism



1	Ground
2	+5 volts
3	Ground
4	Mouse X2 (VIA quadrature signal)
5	Mouse X1 (SCC interrupt signal)
6	(not connected)
7	Mouse switch
8	Mouse Y2 (VIA quadrature signal)
9	Mouse Y1 (SCC interrupt signal)

Figure 6. Pinout for Mouse Jack

Warning: Do not draw more than 200 milliamps at +5 volts from all connectors combined.

Diagram

Figure 7 shows a circuit diagram for the mouse port.

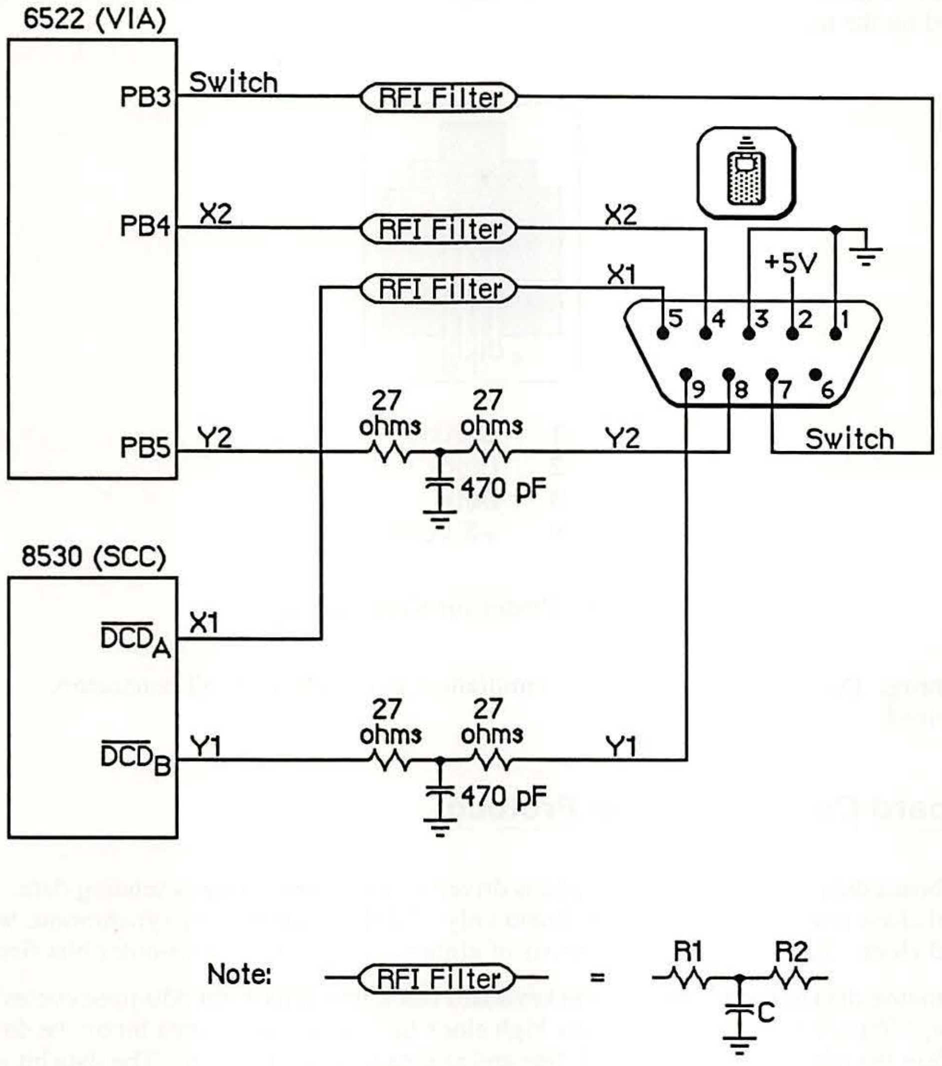


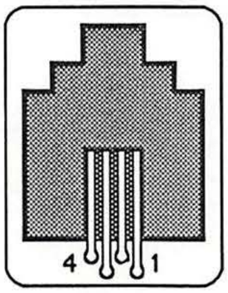
Figure 7. Diagram of Mouse Port

THE KEYBOARD AND KEYPAD

The Macintosh keyboard and numeric keypad each contain an Intel 8021 microprocessor that scans the keys. The 8021 contains ROM and RAM, and is programmed to conform to the interface protocol described below.

The keyboard plugs into the Macintosh through a four-wire RJ-11 telephone-style jack. If a numeric keypad is installed in the system, the keyboard plugs into it and it in turn plugs into the

Macintosh. Figure 8 shows the pinout for the keyboard jack on the Macintosh, on the keyboard itself, and on the numeric keypad.



1	Ground
2	Clock
3	Data
4	+5 volts

Figure 8. Pinout for Keyboard Jack

Warning: Do not draw more than 200 milliamps at +5 volts from all connectors combined.

Keyboard Communication Protocol

The keyboard data line is bidirectional and is driven by whatever device is sending data. The keyboard clock line is driven by the keyboard only. All data transfers are synchronous with the keyboard clock. Each transmission consists of eight bits, with the highest-order bits first.

When sending data to the Macintosh, the keyboard clock transmits eight 330-µsec cycles (160 µsec low, 170 µsec high) on the normally high clock line. It places the data bit on the data line 40 µsec before the falling edge of the clock line and maintains it for 330 µsec. The data bit is clocked into the Macintosh's VIA shift register on the rising edge of the keyboard clock cycle.

When the Macintosh sends data to the keyboard, the keyboard clock transmits eight 400-µsec cycles (180 µsec low, 220 µsec high) on the clock line. On the falling edge of the keyboard clock cycle, the Macintosh places the data bit on the data line and holds it there for 400 µsec. The keyboard reads the data bit 80 µsec after the rising edge of the keyboard clock cycle.

Only the Macintosh can initiate communication over the keyboard lines. On power-up of either the Macintosh or the keyboard, the Macintosh is in charge, and the external device is passive. The Macintosh signals that it's ready to begin communication by pulling the keyboard data line low. Upon detecting this, the keyboard starts clocking and the Macintosh sends a command. The last bit of the command leaves the keyboard data line low; the Macintosh then indicates it's ready to receive the keyboard's response by setting the data line high.

The first command the Macintosh sends out is the Model Number command. The keyboard's response to this command is to reset itself and send back its model number to the Macintosh. If no response is received for 1/2 second, the Macintosh tries the Model Number command again. Once the Macintosh has successfully received a model number from the keyboard, normal operation can begin. The Macintosh sends the Inquiry command; the keyboard sends back a Key Transition response if a key has been pressed or released. If no key transition has occurred after 1/4 second, the keyboard sends back a Null response to let the Macintosh know it's still there. The Macintosh then sends the Inquiry command again. In normal operation, the Macintosh sends out an Inquiry command every 1/4 second. If it receives no response within 1/2 second, it assumes the keyboard is missing or needs resetting, so it begins again with the Model Number command.

There are two other commands the Macintosh can send: the Instant command, which gets an instant keyboard status without the 1/4-second timeout, and the Test command, to perform a keyboard self-test. Here's a list of the commands that can be sent from the Macintosh to the keyboard:

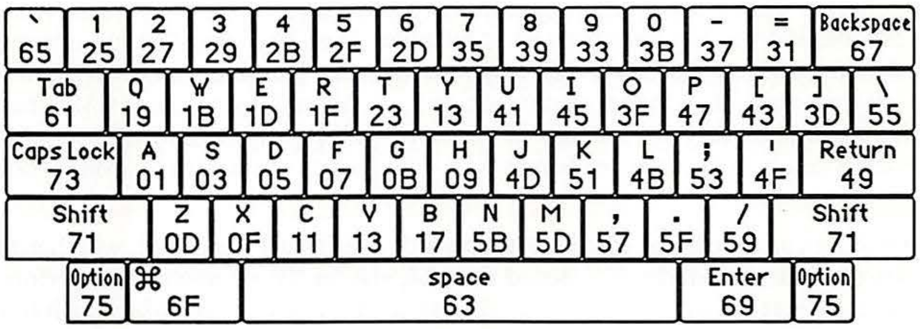
Command name	Value	Keyboard response
Inquiry	$10	Key Transition code or Null ($7B)
Instant	$14	Key Transition code or Null ($7B)
Model Number	$16	Bit 0: 1 Bits 1-3: keyboard model number, 1-8 Bits 4-6: next device number, 1-8 Bit 7: 1 if another device connected
Test	$36	ACK ($7D) or NAK ($77)

The Key Transition responses are sent out by the keyboard as a single byte: Bit 7 high means a key-up transition, and bit 7 low means a key-down. Bit 0 is always high. The Key Transition responses for key-down transitions on the keyboard are shown (in hexadecimal) in Figure 9. Note that these response codes are different from the key codes returned by the keyboard driver software. The keyboard driver strips off bit 7 of the response and shifts the result one bit to the right, removing bit 0. For example, response code $33 becomes $19, and $2B becomes $15.

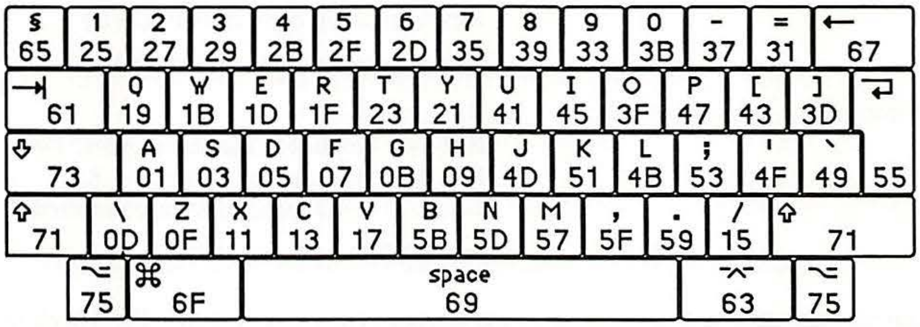
Keypad Communication Protocol

When a numeric keypad is used, it must be inserted *between* the keyboard and the Macintosh; that is, the keypad cable plugs into the jack on the front of the Macintosh, and the keyboard cable plugs into a jack on the numeric keypad. In this configuration, the timings and protocol for the clock and data lines work a little differently: The keypad acts like a keyboard when communicating with the Macintosh, and acts like a Macintosh when communicating over the separate clock and data lines going to the keyboard. All commands from the Macintosh are now received by the keypad instead of the keyboard, and only the keypad can communicate directly with the keyboard.

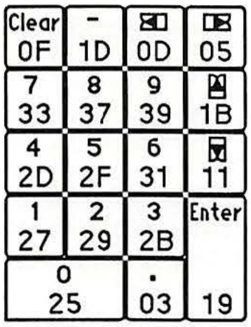
When the Macintosh sends out an Inquiry command, one of two things may happen, depending on the state of the keypad. If no key transitions have occurred on the keypad since the last Inquiry, the keypad sends an Inquiry command to the keyboard and, later, retransmits the keyboard's response back to the Macintosh. But if a key transition has occurred on the keypad, the keypad responds to an Inquiry by sending back the Keypad response ($79) to the Macintosh. In that case, the Macintosh immediately sends an Instant command, and this time the keypad sends back its own Key Transition response. As with the keyboard, bit 7 high means key-up and bit 7 low means key-down.


` 65, 1 25, 2 27, 3 29, 4 2B, 5 2F, 6 2D, 7 35, 8 39, 9 33, 0 3B, - 37, = 31, Backspace 67
Tab 61, Q 19, W 1B, E 1D, R 1F, T 23, Y 13, U 41, I 45, O 3F, P 47, [43,] 3D, \ 55
Caps Lock 73, A 01, S 03, D 05, F 07, G 0B, H 09, J 4D, K 51, L 4B, ; 53, ' 4F, Return 49
Shift 71, Z 0D, X 0F, C 11, V 13, B 17, N 5B, M 5D, , 57, . 5F, / 59, Shift 71
Option 75, ⌘ 6F, space 63, Enter 69, Option 75

U.S. keyboard


§ 65, 1 25, 2 27, 3 29, 4 2B, 5 2F, 6 2D, 7 35, 8 39, 9 33, 0 3B, - 37, = 31, ← 67
→| 61, Q 19, W 1B, E 1D, R 1F, T 23, Y 21, U 41, I 45, O 3F, P 47, [43,] 3D, ↵
⇩ 73, A 01, S 03, D 05, F 07, G 0B, H 09, J 4D, K 51, L 4B, ; 53, ' 4F, ` 49, 55
⇧ 71, \ 0D, Z 0F, X 11, C 13, V 17, B 5B, N 5D, M 57, , 5F, . 59, / 15, ⇧ 71
⌥ 75, ⌘ 6F, space 69, 63, ⌥ 75

International keyboard (Great Britain key caps shown)



Keypad (U.S. key caps shown)

Figure 9. Key-Down Transitions

The Key Transition responses for key-down transitions on the keypad are shown in Figure 9. Again, note that these response codes are different from the key codes returned by the keyboard driver software. The keyboard driver strips off bit 7 of the response and shifts the result one bit to the right, removing bit 0.

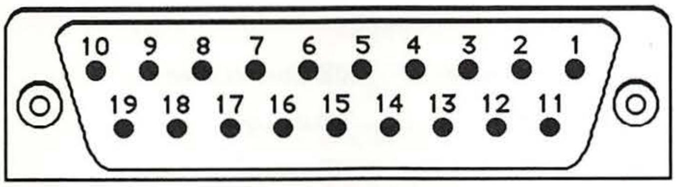
THE DISK INTERFACE

The Macintosh disk interface uses a design similar to that used on the Apple II and Apple III computers, employing the Apple custom IWM chip. Another custom chip called the Analog Signal Generator (ASG) reads the disk speed buffer in RAM and generates voltages that control the disk speed. Together with the VIA, the IWM and the ASG generate all the signals necessary to read, write, format, and eject the 3 1/2-inch disks used by the Macintosh.

The IWM controls four of the disk state-control lines (called CA0, CA1, CA2, and LSTRB), chooses which drive (internal or external) to enable, and processes the disk's read-data and write-data signals. The VIA provides another disk state-control line called SEL.

A buffer in RAM (actually the low-order bytes of words in the sound buffer) is read by the ASG to generate a pulse-width modulated signal that's used to control the speed of the disk motor. The Macintosh Operating System uses this speed control to allow it to store more sectors of information in the tracks closer to the edge of the disk by running the disk motor at slower speeds.

Figure 10 shows the DB-19 pinout for the external disk jack at the back of the Macintosh.



1	Ground	11	CA0
2	Ground	12	CA1
3	Ground	13	CA2
4	Ground	14	LSTRB
5	-12 volts	15	Write request
6	+5 volts	16	SEL
7	+12 volts	17	External drive enable
8	+12 volts	18	Read data
9	(not connected)	19	Write data
10	Motor speed control		

Figure 10. Pinout for Disk Jack

Warning: This connector was designed for a Macintosh 3 1/2-inch disk drive, which represents a load of 500 milliamps at +12 volts, 500 milliamps at +5 volts, and 0 milliamps at –12 volts. If any other device uses this connector, it must not exceed these loads by more than 100 milliamps at +12 volts, 200 milliamps at +5 volts, and 10 milliamps at –12 volts, including loads from all other connectors combined.

Controlling the Disk State-Control Lines

The IWM contains registers that can be used by the software to control the state-control lines leading out to the disk. By reading or writing certain memory locations, you can turn these state-control lines on or off. Other locations set various IWM internal states. The locations are given in the following table as offsets from the constant dBase, the base address of the IWM; this base address is also available in a global variable named IWM. The IWM is on the lower byte of the data bus, so use odd-addressed byte accesses only.

IWM line	**Location to turn line on**	**Location to turn line off**
Disk state-control lines:		
CA0	dBase+ph0H	dBase+ph0L
CA1	dBase+ph1H	dBase+ph1L
CA2	dBase+ph2H	dBase+ph2L
LSTRB	dBase+ph3H	dBase+ph3L
Disk enable line:		
ENABLE	dBase+motorOn	dBase+motorOff
IWM internal states:		
SELECT	dBase+extDrive	dBase+intDrive
Q6	dBase+q6H	dBase+q6L
Q7	dBase+q7H	dBase+q7L

To turn one of the lines on or off, do any kind of memory byte access (read or write) to the respective location.

The CA0, CA1, and CA2 lines are used along with the SEL line from the VIA to select from among the registers and data signals in the disk drive. The LSTRB line is used when writing control information to the disk registers (as described below), and the ENABLE line enables the selected disk drive. SELECT is an IWM internal line that chooses which disk drive can be enabled: On selects the external drive, and off selects the internal drive. The Q6 and Q7 lines are used to set up the internal state of the IWM for reading disk register information, as well as for reading or writing actual disk-storage data.

You can read information from several registers in the disk drive to find out whether the disk is locked, whether a disk is in the drive, whether the head is at track 0, how many heads the drive has, and whether there's a drive connected at all. In turn, you can write to some of these registers to step the head, turn the motor on or off, and eject the disk.

Reading from the Disk Registers

Before you can read from any of the disk registers, you must set up the state of the IWM so that it can pass the data through to the MC68000's memory space where you'll be able to read it. To do that, you must first turn off Q7 by reading or writing dBase+q7L. Then turn on Q6 by accessing dBase+q6H. After that, the IWM will be able to pass data from the disk's RD/SENSE line through to you.

Once you've set up the IWM for disk register access, you must next select which register you want to read. To read one of the disk registers, first enable the drive you want to use (by accessing dBase+intDrive or dBase+extDrive and then dBase+motorOn) and make sure LSTRB is low. Then set CA0, CA1, CA2, and SEL to address the register you want. Once this is done, you can read the disk register data bit in the high-order bit of dBase+q7L. After you've read the data, you may read another disk register by again setting the proper values in CA0, CA1, CA2, and SEL, and then reading dBase+q7L.

> **Warning:** When you're finished reading data from the disk registers, it's important to leave the IWM in a state that the Disk Driver will recognize. To be sure it's in a valid logic state, always turn Q6 back off (by accessing dBase+q6L) after you've finished reading the disk registers.

The following table shows how you must set the disk state-control lines to read from the various disk registers and data signals:

State-control lines				Register	
CA2	**CA1**	**CA0**	**SEL**	**addressed**	**Information in register**
0	0	0	0	DIRTN	Head step direction
0	0	0	1	CSTIN	Disk in place
0	0	1	0	STEP	Disk head stepping
0	0	1	1	WRTPRT	Disk locked
0	1	0	0	MOTORON	Disk motor running
0	1	0	1	TKO	Head at track 0
0	1	1	1	TACH	Tachometer
1	0	0	0	RDDATA0	Read data, lower head
1	0	0	1	RDDATA1	Read data, upper head
1	1	0	0	SIDES	Single- or double-sided drive
1	1	1	1	DRVIN	Drive installed

Writing to the Disk Registers

To write to a disk register, first be sure that LSTRB is off, then turn on CA0 and CA1. Next, set SEL to 0. Set CA0 and CA1 to the proper values from the table below, then set CA2 to the value you want to write to the disk register. Hold LSTRB high for at least one µsec but not more than one msec (unless you're ejecting a disk) and bring it low again. Be sure that you don't change CA0-CA2 or SEL while LSTRB is high, and that CA0 and CA1 are set high before changing SEL.

The following table shows how you must set the disk state-control lines to write to the various disk registers:

Control lines			Register	
CA1	**CA0**	**SEL**	**addressed**	**Register function**
0	0	0	DIRTN	Set stepping direction
0	1	0	STEP	Step disk head one track
1	0	0	MOTORON	Turn on/off disk motor
1	1	0	EJECT	Eject the disk

Explanations of the Disk Registers

The information written to or read from the various disk registers can be interpreted as follows:

- The DIRTN signal sets the direction of subsequent head stepping: 0 causes steps to go toward the inside track (track 79), 1 causes them to go toward the outside track (track 0).
- CSTIN is 0 only when a disk is in the drive.
- Setting STEP to 0 steps the head one full track in the direction last set by DIRTN. When the step is complete (about 12 msec), the disk drive sets STEP back to 1, and then you can step again.
- WRTPRT is 0 whenever the disk is locked. Do not write to a disk unless WRTPRT is 1.
- MOTORON controls the state of the disk motor: 0 turns on the motor, and 1 turns it off. The motor will run only if the drive is enabled and a disk is in place; otherwise, writing to this line will have no effect.
- TKO goes to 0 only if the head is at track 0. This is valid beginning 12 msec after the step that puts it at track 0.
- Writing 1 to EJECT ejects the disk from the drive. To eject a disk, you must hold LSTRB high for at least 1/2 second.
- The current disk speed is available as a pulse train on TACH. The TACH line produces 60 pulses for each rotation of the drive motor. The disk motor speed is controlled by the ASG as it reads the disk speed RAM buffer.
- RDDATA0 and RDDATA1 carry the instantaneous data from the disk head.
- SIDES is always 0 on single-sided drives and 1 on double-sided drives.
- DRVIN is always 0 if the selected disk drive is physically connected to the Macintosh, otherwise it floats to 1.

THE REAL-TIME CLOCK

The Macintosh real-time clock is a custom chip whose interface lines are available through the VIA. The clock contains a four-byte counter that's incremented once each second, as well as a line that can be used by the VIA to generate an interrupt once each second. It also contains 20 bytes of RAM that are powered by a battery when the Macintosh is turned off. These RAM bytes, called **parameter RAM**, contain important data that needs to be preserved even when the system power is not available. The Operating System maintains a copy of parameter RAM that you can access in low memory. To find out how to use the values in parameter RAM, see chapter 13 of Volume II.

Accessing the Clock Chip

The clock is accessed through the following bits of VIA data register B (vBase+vBufB):

```
rTCData     .EQU    0      ;real-time clock serial data line
rTCClk      .EQU    1      ;real-time clock data-clock line
rTCEnb      .EQU    2      ;real-time clock serial enable
```

These three bits constitute a simple serial interface. The rTCData bit is a bidirectional serial data line used to send command and data bytes back and forth. The rTCClk bit is a data-clock line, always driven by the processor (you set it high or low yourself) that regulates the transmission of the data and command bits. The rTCEnb bit is the serial enable line, which signals the real-time clock that the processor is about to send it serial commands and data.

To access the clock chip, you must first enable its serial function. To do this, set the serial enable line (rTCEnb) to 0. Keep the serial enable line 'ow during the entire transaction; if you set it to 1, you'll abort the transfer.

> **Warning:** Be sure you don't alter any of bits 3-7 of VIA data register B during clock serial access.

A command can be either a write request or a read request. After the eight bits of a write request, the clock will expect the next eight bits across the serial data line to be your data for storage into one of the internal registers of the clock. After receiving the eight bits of a read request, the clock will respond by putting eight bits of its data on the serial data line. Commands and data are transferred serially in eight-bit groups over the serial data line, with the high-order bit first and the low-order bit last.

To send a command to the clock, first set the rTCData bit of VIA data direction register B (vBase+vDirB) so that the real-time clock's serial data line will be used for output to the clock. Next, set the rTCClk bit of vBase+vBufB to 0, then set the rTCData bit to the value of the first (high-order) bit of your data byte. Then raise (set to 1) the data-clock bit (rTCClk). Then lower the data-clock, set the serial data line to the next bit, and raise the data-clock line again. After the last bit of your command has been sent in this way, you can either continue by sending your data byte in the same way (if your command was a write request) or switch to receiving a data byte from the clock (if your command was a read request).

To receive a byte of data from the clock, you must first send a command that's a read request. After you've clocked out the last bit of the command, clear the rTCData bit of the data direction register so that the real-time clock's serial data line can be used for input from the clock; then lower the data-clock bit (rTCClk) and read the first (high-order) bit of the clock's data byte on the serial data line. Then raise the data-clock, lower it again, and read the next bit of data. Continue this until all eight bits are read, then raise the serial enable line (rTCEnb), disabling the data transfer.

The following table lists the commands you can send to the clock. A 1 in the high-order bit makes your command a read request; a 0 in the high-order bit makes your command a write request. (In this table, "z" is the bit that determines read or write status, and bits marked "a" are bits whose values depend on what parameter RAM byte you want to address.)

Command byte	**Register addressed by the command**
z0000001	Seconds register 0 (lowest-order byte)
z0000101	Seconds register 1
z0001001	Seconds register 2
z0001101	Seconds register 3 (highest-order byte)
00110001	Test register (write only)
00110101	Write-protect register (write only)
z010aa01	RAM address 100aa ($10-$13)
z1aaaa01	RAM address 0aaaa ($00-$0F)

Note that the last two bits of a command byte must always be 01.

If the high-order bit (bit 7) of the write-protect register is set, this prevents writing into any other register on the clock chip (including parameter RAM). Clearing the bit allows you to change any values in any registers on the chip. Don't try to read from this register; it's a write-only register.

The two highest-order bits (bits 7 and 6) of the test register are used as device control bits during testing, and should always be set to 0 during normal operation. Setting them to anything else will interfere with normal clock counting. Like the write-protect register, this is a write-only register; don't try to read from it.

All clock data must be sent as full eight-bit bytes, even if only one or two bits are of interest. The rest of the bits may not matter, but you must send them to the clock or the write will be aborted when you raise the serial enable line.

It's important to use the proper sequence if you're writing to the clock's seconds registers. If you write to a given seconds register, there's a chance that the clock may increment the data in the next higher-order register during the write, causing unpredictable results. To avoid this possibility, always write to the registers in low-to-high order. Similarly, the clock data may increment during a read of all four time bytes, which could cause invalid data to be read. To avoid this, always read the time twice (or until you get the same value twice).

Warning: When you've finished reading from the clock registers, always end by doing a final write such as setting the write-protect bit. Failure to do this may leave the clock in a state that will run down the battery more quickly than necessary.

The One-Second Interrupt

The clock also generates a VIA interrupt once each second (if this interrupt is enabled). The enable status for this interrupt can be read from or written to bit 0 of the VIA's interrupt enable register (vBase+vIER). When reading the enable register, a 1 bit indicates the interrupt is enabled, and 0 means it's disabled. Writing $01 to the enable register disables the clock's one-second interrupt (without affecting any other interrupts), while writing $81 enables it again. See chapter 6 of Volume II for more information about writing your own interrupt handlers.

Warning: Be sure when you write to bit 0 of the VIA's interrupt enable register that you don't change any of the other bits.

THE VIA

The Synertek SY6522 **Versatile Interface Adapter** (VIA) controls the keyboard, internal real-time clock, parts of the disk, sound, and mouse interfaces, and various internal Macintosh signals. Its base address is available as the constant vBase and is also stored in a global variable named VIA. The VIA is on the upper byte of the data bus, so use even-addressed byte accesses only.

There are two parallel data registers within the VIA, called A and B, each with a data direction register. There are also several event timers, a clocked shift register, and an interrupt flag register with an interrupt enable register.

Normally you won't have to touch the direction registers, since the Operating System sets them up for you at system startup. A 1 bit in a data direction register means the corresponding bit of the respective data register will be used for output, while a 0 bit means it will be used for input.

Note: For more information on the registers and control structure of the VIA, consult the technical specifications for the SY6522 chip.

VIA Register A

VIA data register A is at vBase+vBufA. The corresponding data direction register is at vBase+vDirA.

Bit(s)	Name	Description
7	vSCCWReq	SCC wait/request
6	vPage2	Alternate screen buffer
5	vHeadSel	Disk SEL line
4	vOverlay	ROM low-memory overlay
3	vSndPg2	Alternate sound buffer
0-2	vSound (mask)	Sound volume

The vSCCWReq bit can signal that the SCC has received a character (used to maintain serial communications during disk accesses, when the CPU's interrupts from the SCC are disabled). The vPage2 bit controls which screen buffer is being displayed, and the vHeadSel bit is the SEL control line used by the disk interface. The vOverlay bit (used only during system startup) can be used to place another image of ROM at the bottom of memory, where RAM usually is (RAM moves to $600000). The sound buffer is selected by the vSndPg2 bit. Finally, the vSound bits control the sound volume.

VIA Register B

VIA data register B is at vBase+vBufB. The corresponding data direction register is at vBase+vDirB.

Bit	Name	Description
7	vSndEnb	Sound enable/disable
6	vH4	Horizontal blanking
5	vY2	Mouse Y2
4	vX2	Mouse X2
3	vSW	Mouse switch
2	rTCEnb	Real-time clock serial enable
1	rTCClk	Real-time clock data-clock line
0	rTCData	Real-time clock serial data

The vSndEnb bit turns the sound generator on or off, and the vH4 bit is set when the video beam is in its horizontal blanking period. The vY2 and vX2 bits read the quadrature signals from the Y (vertical) and X (horizontal) directions, respectively, of the mouse's motion lines. The vSW bit reads the mouse switch. The rTCEnb, rTCClk, and rTCData bits control and read the real-time clock.

The VIA Peripheral Control Register

The VIA's peripheral control register, at vBase+vPCR, allows you to set some very low-level parameters (such as positive-edge or negative-edge triggering) dealing with the keyboard data and clock interrupts, the one-second real-time clock interrupt line, and the vertical blanking interrupt.

Bit(s)	Description
5-7	Keyboard data interrupt control
4	Keyboard clock interrupt control
1-3	One-second interrupt control
0	Vertical blanking interrupt control

The VIA Timers

The timers controlled by the VIA are called timer 1 and timer 2. Timer 1 is used to time various events having to do with the Macintosh sound generator. Timer 2 is used by the Disk Driver to time disk I/O events. If either timer isn't being used by the Operating System, you're free to use it for your own purposes. When a timer counts down to 0, an interrupt will be generated if the proper interrupt enable has been set. See chapter 6 of Volume II for information about writing your own interrupt handlers.

To start one of the timers, store the appropriate values in the high- and low-order bytes of the timer counter (or the timer 1 latches, for multiple use of the value). The counters and latches are at the following locations:

Location	Contents
vBase+vT1C	Timer 1 counter (low-order byte)
vBase+vT1CH	Timer 1 counter (high-order byte)
vBase+vT1L	Timer 1 latch (low-order byte)
vBase+vT1LH	Timer 1 latch (high-order byte)
vBase+vT2C	Timer 2 counter (low-order byte)
vBase+vT2CH	Timer 2 counter (high-order byte)

Note: When setting a timer, it's not enough to simply store a full word to the high-order address, because the high- and low-order bytes of the counters are not adjacent. You must explicitly do *two* stores, one for the high-order byte and one for the low-order byte.

VIA Interrupts

The VIA (through its IRQ line) can cause a level-0 processor interrupt whenever one of the following occurs: Timer 1 or timer 2 times out; the keyboard is clocking a bit in through its serial port; the shift register for the keyboard serial interface has finished shifting in or out; the vertical blanking interval is beginning; or the one-second clock has ticked. For more information on how to use these interrupts, see chapter 6 of Volume II.

The interrupt flag register at vBase+vIFR contains flag bits that are set whenever the interrupt corresponding to that bit has occurred. The Operating System uses these flags to determine which device has caused an interrupt. Bit 7 of the interrupt flag register is not really a flag: It remains set (and the IRQ line to the processor is held low) as long as any enabled VIA interrupt is occurring.

Bit	Interrupting device
7	IRQ (all enabled VIA interrupts)
6	Timer 1
5	Timer 2
4	Keyboard clock
3	Keyboard data bit
2	Keyboard data ready
1	Vertical blanking interrupt
0	One-second interrupt

The interrupt enable register, at vBase+vIER, lets you enable or disable any of these interrupts. If an interrupt is disabled, its bit in the interrupt flag register will continue to be set whenever that interrupt occurs, but it won't affect the IRQ flag, nor will it interrupt the processor.

The bits in the interrupt enable register are arranged just like those in the interrupt flag register, except for bit 7. When you write to the interrupt enable register, bit 7 is "enable/disable": If bit 7 is a 1, each 1 in bits 0-6 enables the corresponding interrupt; if bit 7 is a 0, each 1 in bits 0-6 disables that interrupt. In either case, 0's in bits 0-6 do not change the status of those interrupts. Bit 7 is always read as a 1.

Other VIA Registers

The shift register, at vBase+vSR, contains the eight bits of data that have been shifted in or that will be shifted out over the keyboard data line.

The auxiliary control register, at vBase+vACR, is described in the SY6522 documentation. It controls various parameters having to do with the timers and the shift register.

SYSTEM STARTUP

When power is first supplied to the Macintosh, a carefully orchestrated sequence of events takes place.

First, the processor is held in a wait state while a series of circuits gets the system ready for operation. The VIA and IWM are initialized, and the mapping of ROM and RAM are altered temporarily by setting the overlay bit in VIA data register A. This places the ROM starting at the normal ROM location $400000, and a duplicate image of the same ROM starting at address 0 (where RAM normally is), while RAM is placed starting at $600000. Under this mapping, the Macintosh software executes out of the normal ROM locations above $400000, but the MC68000 can obtain some critical low-memory vectors from the ROM image it finds at address 0.

Next, a memory test and several other system tests take place. After the system is fully tested and initialized, the software clears the VIA's overlay bit, mapping the system RAM back where it belongs, starting at address 0. Then the disk startup process begins.

First the internal disk is checked: If there's a disk inserted, the system attempts to read it. If no disk is in the internal drive and there's an external drive with an inserted disk, the system will try to read that one. Otherwise, the question-mark disk icon is displayed until a disk is inserted. If the disk startup fails for some reason, the "sad Macintosh" icon is displayed and the Macintosh goes into an endless loop until it's turned off again.

Once a readable disk has been inserted, the first two sectors (containing the system startup blocks) are read in and the normal disk load begins.

SUMMARY

Warning: This information applies only to the Macintosh 128K and 512K, not to the Macintosh XL.

Constants

```
; VIA base addresses

vBase      .EQU    $EFE1FE     ;main base for VIA chip (in variable VIA)
aVBufB     .EQU    vBase       ;register B base
aVBufA     .EQU    $EFFFFE     ;register A base
aVBufM     .EQU    aVBufB      ;register containing mouse signals
aVIFR      .EQU    $EFFBFE     ;interrupt flag register
aVIER      .EQU    $EFFDFE     ;interrupt enable register

; Offsets from vBase

vBufB      .EQU    512*0       ;register B (zero offset)
vDirB      .EQU    512*2       ;register B direction register
vDirA      .EQU    512*3       ;register A direction register
vT1C       .EQU    512*4       ;timer 1 counter (low-order byte)
vT1CH      .EQU    512*5       ;timer 1 counter (high-order byte)
vT1L       .EQU    512*6       ;timer 1 latch (low-order byte)
vT1LH      .EQU    512*7       ;timer 1 latch (high-order byte)
vT2C       .EQU    512*8       ;timer 2 counter (low-order byte)
vT2CH      .EQU    512*9       ;timer 2 counter (high-order byte)
vSR        .EQU    512*10      ;shift register (keyboard)
vACR       .EQU    512*11      ;auxiliary control register
vPCR       .EQU    512*12      ;peripheral control register
vIFR       .EQU    512*13      ;interrupt flag register
vIER       .EQU    512*14      ;interrupt enable register
vBufA      .EQU    512*15      ;register A

; VIA register A constants

vAOut      .EQU    $7F         ;direction register A:  1 bits = outputs
vAInit     .EQU    $7B         ;initial value for vBufA (medium volume)
vSound     .EQU    7           ;sound volume bits

; VIA register A bit numbers

vSndPg2    .EQU    3           ;0 = alternate sound buffer
vOverlay   .EQU    4           ;1 = ROM overlay (system startup only)
vHeadSel   .EQU    5           ;disk SEL control line
vPage2     .EQU    6           ;0 = alternate screen buffer
vSCCWReq   .EQU    7           ;SCC wait/request line
```

```
; VIA register B constants

vBOut       .EQU    $87         ;direction register B:  1 bits = outputs
vBInit      .EQU    $07         ;initial value for vBufB

; VIA register B bit numbers

rTCData     .EQU    0           ;real-time clock serial data line
rTCClk      .EQU    1           ;real-time clock data-clock line
rTCEnb      .EQU    2           ;real-time clock serial enable
vSW         .EQU    3           ;0 = mouse button is down
vX2         .EQU    4           ;mouse X quadrature level
vY2         .EQU    5           ;mouse Y quadrature level
vH4         .EQU    6           ;1 = horizontal blanking
vSndEnb     .EQU    7           ;0 = sound enabled, 1 = disabled

; SCC base addresses

sccRBase    .EQU    $9FFFF8     ;SCC base read address (in variable SCCRd)
sccWBase    .EQU    $BFFFF9     ;SCC base write address (in variable SCCWr)

; Offsets from SCC base addresses

aData       .EQU    6           ;channel A data in or out
aCtl        .EQU    2           ;channel A control
bData       .EQU    4           ;channel B data in or out
bCtl        .EQU    0           ;channel B control

; Bit numbers for control register RR0

rxBF        .EQU    0           ;1 = SCC receive buffer full
txBE        .EQU    2           ;1 = SCC send buffer empty

; IWM base address

dBase       .EQU    $DFE1FF     ;IWM base address (in variable IWM)

; Offsets from dBase

ph0L        .EQU    512*0       ;CA0 off (0)
ph0H        .EQU    512*1       ;CA0 on (1)
ph1L        .EQU    512*2       ;CA1 off (0)
ph1H        .EQU    512*3       ;CA1 on (1)
ph2L        .EQU    512*4       ;CA2 off (0)
ph2H        .EQU    512*5       ;CA2 on (1)
ph3L        .EQU    512*6       ;LSTRB off (low)
ph3H        .EQU    512*7       ;LSTRB on (high)
mtrOff      .EQU    512*8       ;disk enable off
mtrOn       .EQU    512*9       ;disk enable on
intDrive    .EQU    512*10      ;select internal drive
extDrive    .EQU    512*11      ;select external drive
q6L         .EQU    512*12      ;Q6 off
```

```
q6H         .EQU    512*13      ;Q6 on
q7L         .EQU    512*14      ;Q7 off
q7H         .EQU    512*15      ;Q7 on

; Screen and sound addresses for 512K Macintosh (will also work for
; 128K, since addresses wrap)

screenLow   .EQU    $7A700      ;top left corner of main screen buffer
soundLow    .EQU    $7FD00      ;main sound buffer (in variable SoundBase)
pwmBuffer   .EQU    $7FD01      ;main disk speed buffer
ovlyRAM     .EQU    $600000     ;RAM start address when overlay is set
ovlyScreen  .EQU    $67A700     ;screen start with overlay set
romStart    .EQU    $400000     ;ROM start address (in variable ROMBase)
```

Variables

ROMBase	Base address of ROM
SoundBase	Address of main sound buffer
SCCRd	SCC read base address
SCCWr	SCC write base address
IWM	IWM base address
VIA	VIA base address

Exception Vectors

Location	Purpose
$00	Reset: initial stack pointer (not a vector)
$04	Reset: initial vector
$08	Bus error
$0C	Address error
$10	Illegal instruction
$14	Divide by zero
$18	CHK instruction
$1C	TRAPV instruction
$20	Privilege violation
$24	Trace interrupt
$28	Line 1010 emulator
$2C	Line 1111 emulator
$30-$3B	Unassigned (reserved)
$3C	Uninitialized interrupt
$40-$5F	Unassigned (reserved)

Location	**Purpose**
$60	Spurious interrupt
$64	VIA interrupt
$68	SCC interrupt
$6C	VIA+SCC vector (temporary)
$70	Interrupt switch
$74	Interrupt switch + VIA
$78	Interrupt switch + SCC
$7C	Interrupt switch + VIA + SCC
$80-$BF	TRAP instructions
$C0-$FF	Unassigned (reserved)

3 SUMMARY

ABOUT THIS CHAPTER

This chapter includes all the summaries that appear at the end of other chapters of *Inside Macintosh*. The summaries are arranged in alphabetical order of the part of the Toolbox or Operating System being summarized.

> **Note:** The summaries of the Event Managers are listed under "Event Manager, Operating System" and "Event Manager, Toolbox". The Toolbox and Operating System Utilities are listed similarly.

The last section of this chapter, "Assembly Language", contains information for assembly-language programmers only. It lists some miscellaneous global variables along with hardware-related definitions for the Macintosh 128K and 512K.

APPLETALK MANAGER

Constants

```
CONST lapSize = 20;    {ABusRecord size for ALAP}
      ddpSize = 26;    {ABusRecord size for DDP}
      nbpSize = 26;    {ABusRecord size for NBP}
      atpSize = 56;    {ABusRecord size for ATP}
```

Data Types

```
TYPE ABProtoType  = (lapProto,ddpProto,nbpProto,atpProto);

     ABRecHandle  = ^ABRecPtr;
     ABRecPtr     = ^ABusRecord;
     ABusRecord   =
       RECORD
          abOpcode:         ABCallType;    {type of call}
          abResult:         INTEGER;       {result code}
          abUserReference:  LONGINT;       {for your use}
          CASE ABProtoType  OF
            lapProto:
             (lapAddress:  LAPAdrBlock;  {destination or source node ID}
              lapReqCount: INTEGER;      {length of frame data or buffer }
                                         { size in bytes}
              lapActCount  INTEGER;      {number of frame data bytes }
                                         { actually received}
              lapDataPtr:  Ptr);         {pointer to frame data or pointer }
                                         { to buffer}
            ddpProto:
             (ddpType:     Byte;       {DDP protocol type}
              ddpSocket:   Byte;       {source or listening socket number}
              ddpAddress:  AddrBlock;  {destination or source socket address}
              ddpReqCount: INTEGER;    {length of datagram data or buffer }
                                       { size in bytes}
              ddpActCount: INTEGER;    {number of bytes actually received}
              ddpDataPtr:  Ptr;        {pointer to buffer}
              ddpNodeID:   Byte);      {original destination node ID}
            nbpProto:
             (nbpEntityPtr:        EntityPtr;     {pointer to entity name}
              nbpBufPtr:           Ptr;           {pointer to buffer}
              nbpBufSize:          INTEGER;       {buffer size in bytes}
              nbpDataField:        INTEGER;       {number of addresses or }
                                                  { socket number}
              nbpAddress:          AddrBlock;     {socket address}
              nbpRetransmitInfo:   RetransType);  {retransmission information}
```

```
      atpProto:
         (atpSocket:     Byte;        {listening or responding socket }
                                      { number}
          atpAddress:    AddrBlock;   {destination or source socket }
                                      { address}
          atpReqCount:   INTEGER;     {request size or buffer size}
          atpDataPtr     Ptr;         {pointer to buffer}
          atpRspBDSPtr:  BDSPtr;      {pointer to response BDS}
          atpBitMap:     BitMapType;  {transaction bit map}
          atpTransID:    INTEGER;     {transaction ID}
          atpActCount:   INTEGER;     {number of bytes actually received}
          atpUserData:   LONGINT;     {user bytes}
          atpXO:         BOOLEAN;     {exactly-once flag}
          atpEOM:        BOOLFAN;     {end-of-message flag}
          atpTimeOut:    Byte;        {retry timeout interval in seconds}
          atpRetries:    Byte;        {maximum number of retries}
          atpNumBufs:    Byte;        {number of elements in response }
                                      { BDS or number of response }
                                      { packets sent}
          atpNumRsp:     Byte;        {number of response packets }
                                      { received or sequence number}
          atpBDSSize:    Byte;        {number of elements in response BDS}
          atpRspUData:   LONGINT;     {user bytes sent or received in }
                                      { transaction response}
          atpRspBuf:     Ptr;         {pointer to response message buffer}
          atpRspSize:    INTEGER);    {size of response message buffer}
    END;

ABCallType =  (tLAPRead,tLAPWrite,tDDPRead,tDDPWrite,tNBPLookup,
               tNBPConfirm,tNBPRegister,tATPSndRequest,
               tATPGetRequest,tATPSdRsp,tATPAddRsp,tATPRequest,
               tATPResponse);

LAPAdrBlock =  PACKED RECORD
                 dstNodeID:    Byte;   {destination node ID}
                 srcNodeID:    Byte;   {source node ID}
                 lapProtType:  ABByte  {ALAP protocol type}
               END;

ABByte = 1..127; {ALAP protocol type}

AddrBlock =  PACKED RECORD
               aNet:     INTEGER;  {network number}
               aNode:    Byte;     {node ID}
               aSocket:  Byte      {socket number}
             END;

BDSPtr     = ^BDSType;
BDSType    = ARRAY[0..7] OF BDSElement; {response BDS}
```

```
BDSElement = RECORD
                buffSize:   INTEGER;  {buffer size in bytes}
                buffPtr:    Ptr;      {pointer to buffer}
                dataSize:   INTEGER;  {number of bytes actually received}
                userBytes:  LONGINT   {user bytes}
             END;

BitMapType = PACKED ARRAY[0..7] OF BOOLEAN;

EntityPtr  = ^EntityName;
EntityName = RECORD
               objStr:    Str32;   {object}
               typeStr:   Str32;   {type}
               zoneStr:   Str32    {zone}
             END;

Str32 = STRING[32];

RetransType =
     PACKED RECORD
       retransInterval:  Byte;  {retransmit interval in 8-tick units}
       retransCount:     Byte   {total number of attempts}
     END;
```

Routines [Not in ROM]

Opening and Closing AppleTalk

```
FUNCTION MPPOpen :  OSErr;
FUNCTION MPPClose : OSErr;
```

AppleTalk Link Access Protocol

```
FUNCTION LAPOpenProtocol  (theLAPType: ABByte; protoPtr: Ptr) : OSErr;
FUNCTION LAPCloseProtocol (theLAPType: ABByte) : OSErr;
```

```
FUNCTION LAPWrite (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;
```

←	abOpcode	{always tLAPWrite}
←	abResult	{result code}
→	abUserReference	{for your use}
→	lapAddress.dstNodeID	{destination node ID}
→	lapAddress.lapProtType	{ALAP protocol type}
→	lapReqCount	{length of frame data}
→	lapDataPtr	{pointer to frame data}

```
FUNCTION LAPRead (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;
    ←    abOpcode                    {always tLAPRead}
    ←    abResult                    {result code}
    →    abUserReference             {for your use}
    ←    lapAddress.dstNodeID        {destination node ID}
    ←    lapAddress.srcNodeID        {source node ID}
    →    lapAddress.lapProtType      {ALAP protocol type}
    →    lapReqCount                 {buffer size in bytes}
    ←    lapActCount                 {number of frame data bytes actually received}
    →    lapDataPtr                  {pointer to buffer}

FUNCTION LAPRdCancel (abRecord: ABRecHandle) : OSErr;
```

Datagram Delivery Protocol

```
FUNCTION DDPOpenSocket  (VAR theSocket: Byte; sktListener: Ptr) : OSErr;
FUNCTION DDPCloseSocket (theSocket: Byte) : OSErr;

FUNCTION DDPWrite  (abRecord: ABRecHandle; doChecksum: BOOLEAN; async:
                    BOOLEAN) : OSErr;
    ←    abOpcode                    {always tDDPWrite}
    ←    abResult                    {result code}
    →    abUserReference             {for your use}
    →    ddpType                     {DDP protocol type}
    →    ddpSocket                   {source socket number}
    →    ddpAddress                  {destination socket address}
    →    ddpReqCount                 {length of datagram data}
    →    ddpDataPtr                  {pointer to buffer}

FUNCTION DDPRead  (abRecord: ABRecHandle; retCksumErrs: BOOLEAN; async:
                   BOOLEAN) : OSErr;
    ←    abOpcode                    {always tDDPRead}
    ←    abResult                    {result code}
    →    abUserReference             {for your use}
    ←    ddpType                     {DDP protocol type}
    →    ddpSocket                   {listening socket number}
    ←    ddpAddress                  {source socket address}
    →    ddpReqCount                 {buffer size in bytes}
    ←    ddpActCount                 {number of bytes actually received}
    →    ddpDataPtr                  {pointer to buffer}
    ←    ddpNodeID                   {original destination node ID}

FUNCTION DDPRdCancel (abRecord: ABRecHandle) : OSErr;
```

AppleTalk Transaction Protocol

```
FUNCTION ATPLoad :          OSErr;
FUNCTION ATPUnload :        OSErr;
FUNCTION ATPOpenSocket     (addrRcvd: AddrBlock; VAR atpSocket: Byte) : OSErr;
FUNCTION ATPCloseSocket    (atpSocket: Byte) : OSErr;
```

`FUNCTION ATPSndRequest (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;`

←	abOpcode	{always tATPSndRequest}
←	abResult	{result code}
→	abUserReference	{for your use}
→	atpAddress	{destination socket address}
→	atpReqCount	{request size in bytes}
→	atpDataPtr	{pointer to buffer}
→	atpRspBDSPtr	{pointer to response BDS}
→	atpUserData	{user bytes}
→	atpXO	{exactly-once flag}
←	atpEOM	{end-of-message flag}
→	atpTimeOut	{retry timeout interval in seconds}
→	atpRetries	{maximum number of retries}
→	atpNumBufs	{number of elements in response BDS}
←	atpNumRsp	{number of response packets actually received}

`FUNCTION ATPRequest (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;`

←	abOpcode	{always tATPRequest}
←	abResult	{result code}
→	abUserReference	{for your use}
→	atpAddress	{destination socket address}
→	atpReqCount	{request size in bytes}
→	atpDataPtr	{pointer to buffer}
←	atpActCount	{number of bytes actually received}
→	atpUserData	{user bytes}
→	atpXO	{exactly-once flag}
←	atpEOM	{end-of-message flag}
→	atpTimeOut	{retry timeout interval in seconds}
→	atpRetries	{maximum number of retries}
←	atpRspUData	{user bytes received in transaction response}
→	atpRspBuf	{pointer to response message buffer}
→	atpRspSize	{size of response message buffer}

`FUNCTION ATPReqCancel (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;`

`FUNCTION ATPGetRequest (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;`

←	abOpcode	{always tATPGetRequest}
←	abResult	{result code}
→	abUserReference	{for your use}
→	atpSocket	{listening socket number}
←	atpAddress	{source socket address}
→	atpReqCount	{buffer size in bytes}
→	atpDataPtr	{pointer to buffer}
←	atpBitMap	{transaction bit map}
←	atpTransID	{transaction ID}
←	atpActCount	{number of bytes actually received}
←	atpUserData	{user bytes}
←	atpXO	{exactly-once flag}

```
FUNCTION ATPSndRsp (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;
      ←     abOpcode              {always tATPSdRsp}
      ←     abResult              {result code}
      →     abUserReference       {for your use}
      →     atpSocket             {responding socket number}
      →     atpAddress            {destination socket address}
      →     atpRspBDSPtr          {pointer to response BDS}
      →     atpTransID            {transaction ID}
      →     atpEOM                {end-of-message flag}
      →     atpNumBufs            {number of response packets being sent}
      →     atpBDSSize            {number of elements in response BDS}

FUNCTION ATPAddRsp (abRecord: ABRecHandle) : OSErr;
      ←     abOpcode              {always tATPAddRsp}
      ←     abResult              {result code}
      →     abUserReference       {for your use}
      →     atpSocket             {responding socket number}
      →     atpAddress            {destination socket address}
      →     atpReqCount           {buffer size in bytes}
      →     atpDataPtr            {pointer to buffer}
      →     atpTransID            {transaction ID}
      →     atpUserData           {user bytes}
      →     atpEOM                {end-of-message flag}
      →     atpNumRsp             {sequence number}

FUNCTION ATPResponse (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;
      ←     abOpcode              {always tATPResponse}
      ←     abResult              {result code}
      →     abUserReference       {for your use}
      →     atpSocket             {responding socket number}
      →     atpAddress            {destination socket address}
      →     atpTransID            {transaction ID)
      →     atpRspUData           {user bytes sent in transaction response}
      →     atpRspBuf             {pointer to response message buffer}
      →     atpRspSize            {size of response message buffer}

FUNCTION ATPRspCancel (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;
```

Name-Binding Protocol

```
FUNCTION NBPRegister (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;
      ←     abOpcode              {always tNBPRegister}
      ←     abResult              {result code}
      →     abUserReference       {for your use}
      →     nbpEntityPtr          {pointer to entity name}
      →     nbpBufPtr             {pointer to buffer}
      →     nbpBufSize            {buffer size in bytes}
      →     nbpAddress.aSocket    {socket address}
      →     nbpRetransmitInfo     {retransmission information}
```

```
FUNCTION NBPLookup (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;
    ←    abOpcode              {always tNBPLookup}
    ←    abResult              {result code}
    →    abUserReference       {for your use}
    →    nbpEntityPtr          {pointer to entity name}
    →    nbpBufPtr             {pointer to buffer}
    →    nbpBufSize            {buffer size in bytes}
    ↔    nbpDataField          {number of addresses received}
    →    nbpRetransmitInfo     {retransmission information}

FUNCTION NBPExtract  (theBuffer: Ptr; numInBuf: INTEGER; whichOne:
                      INTEGER; VAR abEntity: EntityName; VAR address:
                      AddrBlock) : OSErr;

FUNCTION NBPConfirm (abRecord: ABRecHandle; async: BOOLEAN) : OSErr;
    ←    abOpcode              {always tNBPConfirm}
    ←    abResult              {result code}
    →    abUserReference       {for your use}
    →    nbpEntityPtr          {pointer to entity name}
    ←    nbpDataField          {socket number}
    →    nbpAddress            {socket address}
    →    nbpRetransmitInfo     {retransmission information}

FUNCTION NBPRemove  (abEntity: EntityPtr) : OSErr;
FUNCTION NBPLoad :   OSErr;
FUNCTION NBPUnload : OSErr;
```

Miscellaneous Routines

```
FUNCTION GetNodeAddress (VAR myNode,myNet: INTEGER) : OSErr;
FUNCTION IsMPPOpen :     BOOLEAN;
FUNCTION IsATPOpen :     BOOLEAN;
```

Result Codes

Name	Value	Meaning
atpBadRsp	–3107	Bad response from ATPRequest
atpLenErr	–3106	ATP response message too large
badATPSkt	–1099	ATP bad responding socket
badBuffNum	–1100	ATP bad sequence number
buf2SmallErr	–3101	ALAP frame too large for buffer DDP datagram too large for buffer
cbNotFound	–1102	ATP control block not found
cksumErr	–3103	DDP bad checksum
ddpLenErr	–92	DDP datagram or ALAP data length too big

Name	Value	Meaning
ddpSktErr	–91	DDP socket error: socket already active; not a well-known socket; socket table full; all dynamic socket numbers in use
excessCollsns	–95	ALAP no CTS received after 32 RTS's, or line sensed in use 32 times (not necessarily caused by collisions)
extractErr	–3104	NBP can't find tuple in buffer
lapProtErr	–94	ALAP error attaching/detaching ALAP protocol type: attach error when ALAP protocol type is negative, not in range, already in table, or when table is full; detach error when ALAP protocol type isn't in table
nbpBuffOvr	–1024	NBP buffer overflow
nbpConfDiff	–1026	NBP name confirmed for different socket
nbpDuplicate	–1027	NBP duplicate name already exists
nbpNISErr	–1029	NBP names information socket error
nbpNoConfirm	–1025	NBP name not confirmed
nbpNotFound	–1028	NBP name not found
noBridgeErr	–93	No bridge found
noDataArea	–1104	Too many outstanding ATP calls
noErr	0	No error
noMPPError	–3102	MPP driver not installed
noRelErr	–1101	ATP no release received
noSendResp	–1103	ATPAddRsp issued before ATPSndRsp
portInUse	–97	Driver Open error, port already in use
portNotCf	–98	Driver Open error, port not configured for this connection
readQErr	–3105	Socket or protocol type invalid or not found in table
recNotFnd	–3108	ABRecord not found
reqAborted	–1105	Request aborted
reqFailed	–1096	ATPSndRequest failed: retry count exceeded
sktClosedErr	–3109	Asynchronous call aborted because socket was closed before call was completed
tooManyReqs	–1097	ATP too many concurrent requests
tooManySkts	–1098	ATP too many responding sockets

Assembly-Language Information

Constants

```
; Serial port use types

useFree         .EQU    0     ;unconfigured
useATalk        .EQU    1     ;configured for AppleTalk
useASync        .EQU    2     ;configured for the Serial Driver

; Bit in PortBUse for .ATP driver status

atpLoadedBit    .EQU    4     ;set if .ATP driver is opened

; Unit numbers for AppleTalk drivers

mppUnitNum      .EQU    9     ;.MPP driver
atpUnitNum      .EQU    10    ;.ATP driver

; csCode values for Control calls (MPP)

writeLAP        .EQU    243
detachPH        .EQU    244
attachPH        .EQU    245
writeDDP        .EQU    246
closeSkt        .EQU    247
openSkt         .EQU    248
loadNBP         .EQU    249
confirmName     .EQU    250
lookupName      .EQU    251
removeName      .EQU    252
registerName    .EQU    253
killNBP         .EQU    254
unloadNBP       .EQU    255

; csCode values for Control calls (ATP)

relRspCB        .EQU    249
closeATPSkt     .EQU    250
addResponse     .EQU    251
sendResponse    .EQU    252
getRequest      .EQU    253
openATPSkt      .EQU    254
sendRequest     .EQU    255
relTCB          .EQU    256

; ALAP header

lapDstAdr       .EQU    0     ;destination node ID
lapSrcAdr       .EQU    1     ;source node ID
lapType         .EQU    2     ;ALAP protocol type
```

```
; ALAP header size

lapHdSz         .EQU    3

; ALAP protocol type values

shortDDP        .EQU    1     ;short DDP header
longDDP         .EQU    2     ;long DDP header

; Long DDP header

ddpHopCnt       .EQU    0     ;count of bridges passed (4 bits)
ddpLength       .EQU    0     ;datagram length (10 bits)
ddpChecksum     .EQU    2     ;checksum
ddpDstNet       .EQU    4     ;destination network number
ddpSrcNet       .EQU    6     ;source network number
ddpDstNode      .EQU    8     ;destination node ID
ddpSrcNode      .EQU    9     ;source node ID
ddpDstSkt       .EQU    10    ;destination socket number
ddpSrcSkt       .EQU    11    ;source socket number
ddpType         .EQU    12    ;DDP protocol type

; DDP long header size

ddpHSzLong      .EQU    ddpType+1

; Short DDP header

ddpLength       .EQU    0               ;datagram length
sDDPDstSkt      .EQU    ddpChecksum     ;destination socket number
sDDPSrcSkt      .EQU    sDDPDstSkt+1    ;source socket number
sDDPType        .EQU    sDDPSrcSkt+1    ;DDP protocol type

; DDP short header size

ddpHSzShort     .EQU    sDDPType+1

; Mask for datagram length

ddpLenMask      .EQU    $03FF

; Maximum size of DDP data

ddpMaxData      .EQU    586

; ATP header

atpControl      .EQU    0     ;control information
atpBitMap       .EQU    1     ;bit map
atpRespNo       .EQU    1     ;sequence number
atpTransID      .EQU    2     ;transaction ID
atpUserData     .EQU    4     ;user bytes
```

```
; ATP header size

atpHdSz         .EQU    8

; DDP protocol type for ATP packets

atp             .EQU    3

; ATP function code

atpReqCode      .EQU    $40     ;TReq packet
atpRspCode      .EQU    $80     ;TResp packet
atpRelCode      .EQU    $C0     ;TRel packet

; ATPFlags control information bits

sendChk         .EQU    0       ;send-checksum bit
tidValid        .EQU    1       ;transaction ID validity bit
atpSTSBit       .EQU    3       ;send-transmission-status bit
atpEOMBit       .EQU    4       ;end-of-message bit
atpXOBit        .EQU    5       ;exactly-once bit

; Maximum number of ATP request packets

atpMaxNum       .EQU    8

; ATP buffer data structure

bdsBuffSz       .EQU    0       ;size of data to send or buffer size
bdsBuffAddr     .EQU    2       ;pointer to data or buffer
bdsDataSz       .EQU    6       ;number of bytes actually received
bdsUserData     .EQU    8       ;user bytes

; BDS element size

bdsEntrySz      .EQU    12

; NBP packet

nbpControl      .EQU    0       ;packet type
nbpTCount       .EQU    0       ;tuple count
nbpID           .EQU    1       ;packet identifier
nbpTuple        .EQU    2       ;start of first tuple

; DDP protocol type for NBP packets

nbp             .EQU    2
```

```
; NBP packet types

brRq            .EQU    1       ;broadcast request
lkUp            .EQU    2       ;lookup request
lkUpReply       .EQU    3       ;lookup reply

; NBP tuple

tupleNet        .EQU    0       ;network number
tupleNode       .EQU    2       ;node ID
tupleSkt        .EQU    3       ;socket number
tupleEnum       .EQU    4       ;used internally
tupleName       .EQU    5       ;entity name

; Maximum number of tuples in NBP packet

tupleMax        .EQU    15

; NBP meta-characters

equals          .EQU    '='     ;"wild-card" meta-character
star            .EQU    '*'     ;"this zone" meta-character

; NBP names table entry

ntLink          .EQU    0       ;pointer to next entry
ntTuple         .EQU    4       ;tuple
ntSocket        .EQU    7       ;socket number
ntEntity        .EQU    9       ;entity name

; NBP names information socket number

nis             .EQU    2
```

Routines

Link Access Protocol

WriteLAP function

→	26	csCode	word	;always writeLAP
→	30	wdsPointer	pointer	;write data structure

AttachPH function

→	26	csCode	word	;always attachPH
→	28	protType	byte	;ALAP protocol type
→	30	handler	pointer	;protocol handler

DetachPH function

→	26	csCode	word	;always detachPH
→	28	protType	byte	;ALAP protocol type

Datagram Delivery Protocol

OpenSkt function

→	26	csCode	word	;always openSkt
↔	28	socket	byte	;socket number
→	30	listener	pointer	;socket listener

CloseSkt function

→	26	csCode	word	;always closeSkt
→	28	socket	byte	;socket number

WriteDDP function

→	26	csCode	word	;always writeDDP
→	28	socket	byte	;socket number
→	29	checksumFlag	byte	;checksum flag
→	30	wdsPointer	pointer	;write data structure

AppleTalk Transaction Protocol

OpenATPSkt function

→	26	csCode	word	;always openATPSkt
↔	28	atpSocket	byte	;socket number
→	30	addrBlock	long word	;socket request specification

CloseATPSkt function

→	26	csCode	word	;always closeATPSkt
→	28	atpSocket	byte	;socket number

SendRequest function

→	18	userData	long word	;user bytes
←	22	reqTID	word	;transaction ID used in request
→	26	csCode	word	;always sendRequest
←	28	currBitMap	byte	;bit map
↔	29	atpFlags	byte	;control information
→	30	addrBlock	long word	;destination socket address
→	34	reqLength	word	;request size in bytes
→	36	reqPointer	pointer	;pointer to request data
→	40	bdsPointer	pointer	;pointer to response BDS
→	44	numOfBuffs	byte	;number of responses expected
→	45	timeOutVal	byte	;timeout interval
←	46	numOfResps	byte	;number of responses received
↔	47	retryCount	byte	;number of retries

GetRequest function

←	18	userData	long word	;user bytes
→	26	csCode	word	;always getRequest
→	28	atpSocket	byte	;socket number

GetRequest function

←	18	userData	long word	;user bytes
→	26	csCode	word	;always getRequest
→	28	atpSocket	byte	;socket number
←	29	atpFlags	byte	;control information
←	30	addrBlock	long word	;source of request
↔	34	reqLength	word	;request buffer size
→	36	reqPointer	pointer	;pointer to request buffer
←	44	bitMap	byte	;bit map
←	46	transID	word	;transaction ID

SendResponse function

←	18	userData	long word	;user bytes from TRel
→	26	csCode	word	;always sendResponse
→	28	atpSocket	byte	;socket number
→	29	atpFlags	byte	;control information
→	30	addrBlock	long word	;response destination
→	40	bdsPointer	pointer	;pointer to response BDS
→	44	numOfBuffs	byte	;number of response packets being sent
→	45	bdsSize	byte	;BDS size in elements
→	46	transID	word	;transaction ID

AddResponse function

→	18	userData	long word	;user bytes
→	26	csCode	word	;always addResponse
→	28	atpSocket	byte	;socket number
→	29	atpFlags	byte	;control information
→	30	addrBlock	long word	;response destination
→	34	reqLength	word	;response size
→	36	reqPointer	pointer	;pointer to response
→	44	rspNum	byte	;sequence number
→	46	transID	word	;transaction ID

RelTCB function

→	26	csCode	word	;always relTCB
→	30	addrBlock	long word	;destination of request
→	46	transID	word	;transaction ID of request

RelRspCB function

→	26	csCode	word	;always relRspCB
→	28	atpSocket	byte	;socket number that request was received on
→	30	addrBlock	long word	;source of request
→	46	transID	word	;transaction ID of request

Name-Binding Protocol

RegisterName function

→	26	csCode	word	;always registerName
→	28	interval	byte	;retry interval
↔	29	count	byte	;retry count
→	30	ntQElPtr	pointer	;names table element pointer
→	34	verifyFlag	byte	;set if verify needed

LookupName function

→	26	csCode	word	;always lookupName
→	28	interval	byte	;retry interval
↔	29	count	byte	;retry count
→	30	entityPtr	pointer	;pointer to entity name
→	34	retBuffPtr	pointer	;pointer to buffer
→	38	retBuffSize	word	;buffer size in bytes
→	40	maxToGet	word	;matches to get
←	42	numGotten	word	;matches found

ConfirmName function

→	26	csCode	word	;always confirmName
→	28	interval	byte	;retry interval
↔	29	count	byte	;retry count
→	30	entityPtr	pointer	;pointer to entity name
→	34	confirmAddr	pointer	;entity address
←	38	newSocket	byte	;socket number

RemoveName function

→	26	csCode	word	;always removeName
→	30	entityPtr	pointer	;pointer to entity name

LoadNBP function

→	26	csCode	word	;always loadNBP

UnloadNBP function

→	26	csCode	word	;always unloadNBP

Variables

SPConfig	Use types for serial ports (byte) (bits 0-3: current configuration of serial port B bits 4-6: current configuration of serial port A)
PortBUse	Current availability of serial port B (byte) (bit 7: 1 = not in use, 0 = in use bits 0-3: current use of port bits bits 4-6: driver-specific)
ABusVars	Pointer to AppleTalk variables

BINARY-DECIMAL CONVERSION PACKAGE

Routines

```
PROCEDURE NumToString (theNum: LONGINT; VAR theString: Str255);
PROCEDURE StringToNum (theString: Str255; VAR theNum: LONGINT);
```

Assembly-Language Information

Constants

```
; Routine selectors

numToString     .EQU    0
stringToNum     .EQU    1
```

Routines

Name	On entry	On exit
NumToString	A0: ptr to theString (preceded by length byte) D0: theNum (long)	A0: ptr to theString
StringToNum	A0: ptr to theString (preceded by length byte)	D0: theNum (long)

Trap Macro Name

_Pack7

CONTROL MANAGER

Constants

```
CONST { Control definition IDs }

      pushButProc   = 0;   {simple button}
      checkBoxProc  = 1;   {check box}
      radioButProc  = 2;   {radio button}
      useWFont      = 8;   {add to above to use window's font}
      scrollBarProc = 16;  {scroll bar}

      { Part codes }

      inButton     = 10;  {simple button}
      inCheckBox   = 11;  {check box or radio button}
      inUpButton   = 20;  {up arrow of a scroll bar}
      inDownButton = 21;  {down arrow of a scroll bar}
      inPageUp     = 22;  {"page up" region of a scroll bar}
      inPageDown   = 23;  {"page down" region of a scroll bar}
      inThumb      = 129; {thumb of a scroll bar}

      { Axis constraints for DragControl }

      noConstraint = 0;  {no constraint}
      hAxisOnly    = 1;  {horizontal axis only}
      vAxisOnly    = 2;  {vertical axis only}

      { Messages to control definition function }

      drawCntl  = 0; {draw the control (or control part)}
      testCntl  = 1; {test where mouse button was pressed}
      calcCRgns = 2; {calculate control's region (or indicator's)}
      initCntl  = 3; {do any additional control initialization}
      dispCntl  = 4; {take any additional disposal actions}
      posCntl   = 5; {reposition control's indicator and update it}
      thumbCntl = 6; {calculate parameters for dragging indicator}
      dragCntl  = 7; {drag control (or its indicator)}
      autoTrack = 8; {execute control's action procedure}
```

Data Types

```
TYPE ControlHandle  = ^ControlPtr;
     ControlPtr     = ^ControlRecord;
```

```
ControlRecord =
     PACKED RECORD
       nextControl:   ControlHandle;    {next control}
       contrlOwner:   WindowPtr;        {control's window}
       contrlRect:    Rect;             {enclosing rectangle}
       contrlVis:     Byte;             {255 if visible}
       contrlHilite:  Byte;             {highlight state}
       contrlValue:   INTEGER;          {control's current setting}
       contrlMin:     INTEGER;          {control's minimum setting}
       contrlMax:     INTEGER;          {control's maximum setting}
       contrlDefProc: Handle;           {control definition function}
       contrlData:    Handle;           {data used by contrlDefProc}
       contrlAction:  ProcPtr;          {default action procedure}
       contrlRfCon:   LONGINT;          {control's reference value}
       contrlTitle:   Str255            {control's title}
     END;
```

Routines

Initialization and Allocation

```
FUNCTION  NewControl     (theWindow: WindowPtr; boundsRect: Rect; title:
                          Str255; visible: BOOLEAN; value: INTEGER;
                          min,max: INTEGER; procID: INTEGER; refCon:
                          LONGINT) : ControlHandle;
FUNCTION  GetNewControl  (controlID: INTEGER; theWindow: WindowPtr) :
                          ControlHandle;
PROCEDURE DisposeControl (theControl: ControlHandle);
PROCEDURE KillControls   (theWindow: WindowPtr);
```

Control Display

```
PROCEDURE SetCTitle      (theControl: ControlHandle; title: Str255);
PROCEDURE GetCTitle      (theControl: ControlHandle; VAR title: Str255);
PROCEDURE HideControl    (theControl: ControlHandle);
PROCEDURE ShowControl    (theControl: ControlHandle);
PROCEDURE DrawControls   (theWindow: WindowPtr);
PROCEDURE HiliteControl  (theControl: ControlHandle; hiliteState:
                          INTEGER);
```

Mouse Location

```
FUNCTION FindControl    (thePoint: Point; theWindow: WindowPtr; VAR
                         whichControl: ControlHandle) : INTEGER;
FUNCTION TrackControl   (theControl: ControlHandle; startPt: Point;
                         actionProc: ProcPtr) : INTEGER;
FUNCTION TestControl    (theControl: ControlHandle; thePoint: Point) :
                         INTEGER;
```

Control Movement and Sizing

```
PROCEDURE MoveControl  (theControl: ControlHandle; h,v: INTEGER);
PROCEDURE DragControl  (theControl: ControlHandle; startPt: Point;
                        limitRect,slopRect: Rect; axis: INTEGER);
PROCEDURE SizeControl  (theControl: ControlHandle; w,h: INTEGER);
```

Control Setting and Range

```
PROCEDURE SetCtlValue  (theControl: ControlHandle; theValue: INTEGER);
FUNCTION  GetCtlValue  (theControl: ControlHandle) : INTEGER;
PROCEDURE SetCtlMin    (theControl: ControlHandle; minValue: INTEGER);
FUNCTION  GetCtlMin    (theControl: ControlHandle) : INTEGER;
PROCEDURE SetCtlMax    (theControl: ControlHandle; maxValue INTEGER);
FUNCTION  GetCtlMax    (theControl: ControlHandle) : INTEGER;
```

Miscellaneous Routines

```
PROCEDURE SetCRefCon    (theControl: ControlHandle; data: LONGINT);
FUNCTION  GetCRefCon    (theControl: ControlHandle) : LONGINT;
PROCEDURE SetCtlAction  (theControl: ControlHandle; actionProc ProcPtr);
FUNCTION  GetCtlAction  (theControl: ControlHandle) : ProcPtr;
```

Action Procedure for TrackControl

```
If an indicator:      PROCEDURE MyAction;
If not an indicator:  PROCEDURE MyAction (theControl: ControlHandle;
                                          partCode: INTEGER);
```

Control Definition Function

```
FUNCTION MyControl  (varCode: INTEGER; theControl: ControlHandle;
                     message: INTEGER; param: LONGINT) : LONGINT;
```

Assembly-Language Information

Constants

```
; Control definition IDs

pushButProc     .EQU  0   ;simple button
checkBoxProc    .EQU  1   ;check box
radioButProc    .EQU  2   ;radio button
useWFont        .EQU  8   ;add to above to use window's font
scrollBarProc   .EQU  16  ;scroll bar
```

```
; Part codes

inButton        .EQU  10   ;simple button
inCheckBox      .EQU  11   ;check box or radio button
inUpButton      .EQU  20   ;up arrow of a scroll bar
inDownButton    .EQU  21   ;down arrow of a scroll bar
inPageUp        .EQU  22   ;"page up" region of a scroll bar
inPageDown      .EQU  23   ;"page down" region of a scroll bar
inThumb         .EQU  129  ;thumb of a scroll bar

; Axis constraints for DragControl

noConstraint    .EQU  0   ;no constraint
hAxisOnly       .EQU  1   ;horizontal axis only
vAxisOnly       .EQU  2   ;vertical axis only

; Messages to control definition function

drawCtlMsg      .EQU  0   ;draw the control (or control part)
hitCtlMsg       .EQU  1   ;test where mouse button was pressed
calcCtlMsg      .EQU  2   ;calculate control's region (or indicator's)
newCtlMsg       .EQU  3   ;do any additional control initialization
dispCtlMsg      .EQU  4   ;take any additional disposal actions
posCtlMsg       .EQU  5   ;reposition control's indicator and update it
thumbCtlMsg     .EQU  6   ;calculate parameters for dragging indicator
dragCtlMsg      .EQU  7   ;drag control (or its indicator)
trackCtlMsg     .EQU  8   ;execute control's action procedure
```

Control Record Data Structure

nextControl	Handle to next control in control list
contrlOwner	Pointer to this control's window
contrlRect	Control's enclosing rectangle (8 bytes)
contrlVis	255 if control is visible (byte)
contrlHilite	Highlight state (byte)
contrlValue	Control's current setting (word)
contrlMin	Control's minimum setting (word)
contrlMax	Control's maximum setting (word)
contrlDefHandle	Handle to control definition function
contrlData	Data used by control definition function (long)
contrlAction	Address of default action procedure
contrlRfCon	Control's reference value (long)
contrlTitle	Handle to control's title (preceded by length byte)
contrlSize	Size in bytes of control record except contrlTitle field

Special Macro Names

Pascal name	**Macro name**
DisposeControl	_DisposControl
GetCtlMax	_GetMaxCtl
GetCtlMin	_GetMinCtl
SetCtlMax	_SetMaxCtl
SetCtlMin	_SetMinCtl

Variables

DragHook	Address of procedure to execute during TrackControl and DragControl
DragPattern	Pattern of dragged region's outline (8 bytes)

DESK MANAGER

Routines

Opening and Closing Desk Accessories

```
FUNCTION  OpenDeskAcc  (theAcc: Str255) : INTEGER;
PROCEDURE CloseDeskAcc (refNum: INTEGER);
```

Handling Events in Desk Accessories

```
PROCEDURE SystemClick (theEvent: EventRecord; theWindow: WindowPtr);
FUNCTION  SystemEdit  (editCmd: INTEGER) : BOOLEAN;
```

Performing Periodic Actions

```
PROCEDURE SystemTask;
```

Advanced Routines

```
FUNCTION  SystemEvent (theEvent: EventRecord) : BOOLEAN;
PROCEDURE SystemMenu  (menuResult: LONGINT);
```

Assembly-Language Information

Constants

```
; Desk accessory flag

dNeedTime     .EQU     5      ;set if driver needs time for performing a
                              ; periodic action

; Control routine messages

accEvent      .EQU     64     ;handle a given event
accRun        .EQU     65     ;take the periodic action, if any, for
                              ; this desk accessory
accCursor     .EQU     66     ;change cursor shape if appropriate;
                              ; generate null event if window was
                              ; created by Dialog Manager
accMenu       .EQU     67     ;handle a given menu item
accUndo       .EQU     68     ;handle the Undo command
```

```
accCut        .EQU     70    ;handle the Cut command
accCopy       .EQU     71    ;handle the Copy command
accPaste      .EQU     72    ;handle the Paste command
accClear      .EQU     73    ;handle the Clear command
```

Special Macro Names

Pascal name	**Macro name**
SystemEdit	_SysEdit

Variables

MBarEnable	Unique menu ID for active desk accessory, when menu bar belongs to the accessory (word)
SEvtEnb	0 if SystemEvent should return FALSE (byte)

DEVICE MANAGER

Constants

```
CONST { Values for requesting read/write access }

      fsCurPerm   = 0;    {whatever is currently allowed}
      fsRdPerm    = 1;    {request to read only}
      fsWrPerm    = 2;    {request to write only}
      fsRdWrPerm  = 3;    {request to read and write}

      { Positioning modes }

      fsAtMark     = 0;   {at current position}
      fsFromStart  = 1;   {offset relative to beginning of medium}
      fsFromMark   = 3;   {offset relative to current position}
      rdVerify     = 64;  {add to above for read-verify}
```

Data Types

```
TYPE ParamBlkType    = (ioParam,fileParam,volumeParam,cntrlParam);

     ParmBlkPtr      = ^ParamBlockRec;
     ParamBlockRec   = RECORD
         qLink:          QElemPtr;      {next queue entry}
         qType:          INTEGER;       {queue type}
         ioTrap:         INTEGER;       {routine trap}
         ioCmdAddr:      Ptr;           {routine address}
         ioCompletion:   ProcPtr;       {completion routine}
         ioResult:       OSErr;         {result code}
         ioNamePtr:      StringPtr;     {driver name}
         ioVRefNum:      INTEGER;       {volume reference or drive number}
       CASE ParamBlkType OF
         ioParam:
          (ioRefNum:     INTEGER;       {driver reference number}
           ioVersNum:    SignedByte;    {not used}
           ioPermssn:    SignedByte;    {read/write permission}
           ioMisc:       Ptr;           {not used}
           ioBuffer:     Ptr;           {pointer to data buffer}
           ioReqCount:   LONGINT;       {requested number of bytes}
           ioActCount:   LONGINT;       {actual number of bytes}
           ioPosMode:    INTEGER;       {positioning mode}
           ioPosOffset:  LONGINT);      {positioning offset}
         fileParam:
           . . . {used by File Manager}
         volumeParam:
           . . . {used by File Manager}
```

```
  cntrlParam:
   (ioCRefNum: INTEGER;   {driver reference number}
    csCode:    INTEGER;   {type of Control or Status call}
    csParam:   ARRAY[0..10] OF INTEGER) {control or status information}
END;

DCtlHandle  = ^DCtlPtr;
DCtlPtr     = ^DCtlEntry;
DCtlEntry   =
  RECORD
    dCtlDriver:    Ptr;         {pointer to ROM driver or handle to }
                                { RAM driver}
    dCtlFlags:     INTEGER;     {flags}
    dCtlQHdr:      QHdr;        {driver I/O queue header}
    dCtlPosition:  LONGINT;     {byte position used by Read and }
                                { Write calls}
    dCtlStorage:   Handle;      {handle to RAM driver's private }
                                { storage}
    dCtlRefNum:    INTEGER;     {driver reference number}
    dCtlCurTicks:  LONGINT;     {used internally}
    dCtlWindow:    WindowPtr;   {pointer to driver's window}
    dCtlDelay:     INTEGER;     {number of ticks between periodic }
                                { actions}
    dCtlEMask:     INTEGER;     {desk accessory event mask}
    dCtlMenu:      INTEGER      {menu ID of menu associated with }
                                { driver}
  END;
```

High-Level Routines [Not in ROM]

```
FUNCTION OpenDriver   (name: Str255; VAR refNum: INTEGER) : OSErr;
FUNCTION CloseDriver  (refNum: INTEGER) : OSErr;
FUNCTION FSRead       (refNum: INTEGER; VAR count: LONGINT; buffPtr: Ptr)
                       : OSErr;
FUNCTION FSWrite      (refNum: INTEGER; VAR count: LONGINT; buffPtr: Ptr)
                       : OSErr;
FUNCTION Control      (refNum: INTEGER; csCode: INTEGER; csParamPtr: Ptr)
                       : OSErr;
FUNCTION Status       (refNum: INTEGER; csCode: INTEGER; csParamPtr: Ptr)
                       : OSErr;
FUNCTION KillIO       (refNum: INTEGER) : OSErr;
```

Low-Level Routines

```
FUNCTION PBOpen (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
←	24	ioRefNum	word
→	27	ioPermssn	byte

```
FUNCTION PBClose (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
    →  12      ioCompletion      pointer
    ←  16      ioResult          word
    →  24      ioRefNum          word

FUNCTION PBRead (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
    →  12      ioCompletion      pointer
    ←  16      ioResult          word
    →  22      ioVRefNum         word
    →  24      ioRefNum          word
    →  32      ioBuffer          pointer
    →  36      ioReqCount        long word
    ←  40      ioActCount        long word
    →  44      ioPosMode         word
    ↔  46      ioPosOffset       long word

FUNCTION PBWrite (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
    →  12      ioCompletion      pointer
    ←  16      ioResult          word
    →  22      ioVRefNum         word
    →  24      ioRefNum          word
    →  32      ioBuffer          pointer
    →  36      ioReqCount        long word
    ←  40      ioActCount        long word
    →  44      ioPosMode         word
    ↔  46      ioPosOffset       long word

FUNCTION PBControl (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
    →  12      ioCompletion      pointer
    ←  16      ioResult          word
    →  22      ioVRefNum         word
    →  24      ioRefNum          word
    →  26      csCode            word
    →  28      csParam           record

FUNCTION PBStatus (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
    →  12      ioCompletion      pointer
    ←  16      ioResult          word
    →  22      ioVRefNum         word
    →  24      ioRefNum          word
    →  26      csCode            word
    ←  28      csParam           record

FUNCTION PBKillIO (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
    →  12      ioCompletion      pointer
    ←  16      ioResult          word
    →  24      ioRefNum          word
```

Accessing a Driver's Device Control Entry

```
FUNCTION GetDCtlEntry (refNum: INTEGER) : DCtlHandle;   [Not in ROM]
```

Result Codes

Name	Value	Meaning
abortErr	–27	I/O request aborted by KillIO
badUnitErr	–21	Driver reference number doesn't match unit table
controlErr	–17	Driver can't respond to this Control call
dInstErr	–26	Couldn't find driver in resource file
dRemovErr	–25	Attempt to remove an open driver
noErr	0	No error
notOpenErr	–28	Driver isn't open
openErr	–23	Requested read/write permission doesn't match driver's open permission
readErr	–19	Driver can't respond to Read calls
statusErr	–18	Driver can't respond to this Status call
unitEmptyErr	–22	Driver reference number specifies NIL handle in unit table
writErr	–20	Driver can't respond to Write calls

Assembly-Language Information

Constants

```
 ; Flags in trap words

asnycTrpBit     .EQU    10     ;set for an asynchronous call
noQueueBit      .EQU     9     ;set for immediate execution

; Values for requesting read/write access

fsCurPerm       .EQU    0      ;whatever is currently allowed
fsRdPerm        .EQU    1      ;request to read only
fsWrPerm        .EQU    2      ;request to write only
fsRdWrPerm      .EQU    3      ;request to read and write

; Positioning modes

fsAtMark        .EQU     0     ;at current position
fsFromStart     .EQU     1     ;offset relative to beginning of medium
fsFromMark      .EQU     3     ;offset relative to current position
rdVerify        .EQU    64     ;add to above for read-verify
```

```
; Driver flags

dReadEnable      .EQU     0  ;set if driver can respond to Read calls
dWriteEnable     .EQU     1  ;set if driver can respond to Write calls
dCtlEnable       .EQU     2  ;set if driver can respond to Control calls
dStatEnable      .EQU     3  ;set if driver can respond to Status calls
dNeedGoodBye     .EQU     4  ;set if driver needs to be called before the
                             ; application heap is reinitialized
dNeedTime        .EQU     5  ;set if driver needs time for performing a
                             ; periodic action
dNeedLock        .EQU     6  ;set if driver will be locked in memory as
                             ; soon as it's opened (always set for ROM
                             ; drivers)

; Device control entry flags

dOpened          .EQU     5    ;set if driver is open
dRAMBased        .EQU     6    ;set if driver is RAM-based
drvrActive       .EQU     7    ;set if driver is currently executing

; csCode values for driver control routine

accRun      .EQU    65   ;take the periodic action, if any, for this
                         ; driver
goodBye     .EQU    -1   ;heap will be reinitialized, clean up if
                         ; necessary
killCode    .EQU     1   ;handle the KillIO call

; Low-order byte of Device Manager traps

aRdCmd      .EQU    2    ;Read call (trap $A002)
aWrCmd      .EQU    3    ;Write call (trap $A003)

; Offsets from SCC base addresses

aData       .EQU    6    ;channel A data in or out
aCtl        .EQU    2    ;channel A control
bData       .EQU    4    ;channel B data in or out
bCtl        .EQU    0    ;channel B control
```

Standard Parameter Block Data Structure

qLink	Pointer to next queue entry
qType	Queue type (word)
ioTrap	Routine trap (word)
ioCmdAddr	Routine address
ioCompletion	Address of completion routine
ioResult	Result code (word)
ioVNPtr	Pointer to driver name (preceded by length byte)
ioVRefNum	Volume reference number (word)
ioDrvNum	Drive number (word)

Control and Status Parameter Block Data Structure

ioRefNum	Driver reference number (word)
csCode	Type of Control or Status call (word)
csParam	Parameters for Control or Status call (22 bytes)

I/O Parameter Block Data Structure

ioRefNum	Driver reference number (word)
ioPermssn	Open permission (byte)
ioBuffer	Pointer to data buffer
ioReqCount	Requested number of bytes (long)
ioActCount	Actual number of bytes (long)
ioPosMode	Positioning mode (word)
ioPosOffset	Positioning offset (long)

Device Driver Data Structure

drvrFlags	Flags (word)
drvrDelay	Number of ticks between periodic actions (word)
drvrEMask	Desk accessory event mask (word)
drvrMenu	Menu ID of menu associated with driver (word)
drvrOpen	Offset to open routine (word)
drvrPrime	Offset to prime routine (word)
drvrCtl	Offset to control routine (word)
drvrStatus	Offset to status routine (word)
drvrClose	Offset to close routine (word)
drvrName	Driver name (preceded by length byte)

Device Control Entry Data Structure

dCtlDriver	Pointer to ROM driver or handle to RAM driver
dCtlFlags	Flags (word)
dCtlQueue	Queue flags: low-order byte is driver's version number (word)
dCtlQHead	Pointer to first entry in driver's I/O queue
dCtlQTail	Pointer to last entry in driver's I/O queue
dCtlPosition	Byte position used by Read and Write calls (long)
dCtlStorage	Handle to RAM driver's private storage
dCtlRefNum	Driver's reference number (word)
dCtlWindow	Pointer to driver's window
dCtlDelay	Number of ticks between periodic actions (word)
dCtlEMask	Desk accessory event mask (word)
dCtlMenu	Menu ID of menu associated with driver (word)

Structure of Primary Interrupt Vector Table

autoInt1	Vector to level-1 interrupt handler
autoInt2	Vector to level-2 interrupt handler

autoInt3	Vector to level-3 interrupt handler
autoInt4	Vector to level-4 interrupt handler
autoInt5	Vector to level-5 interrupt handler
autoInt6	Vector to level-6 interrupt handler
autoInt7	Vector to level-7 interrupt handler

Macro Names

Pascal name	Macro name
PBRead	_Read
PBWrite	_Write
PBControl	_Control
PBStatus	_Status
PBKillIO	_KillIO

Routines for Writing Drivers

Routine	Jump vector	On entry	On exit
Fetch	JFetch	A1: ptr to device control entry	D0: character fetched; bit 15=1 if last character in buffer
Stash	JStash	A1: ptr to device control entry D0: character to stash	D0: bit 15=1 if last character requested
IODone	JIODone	A1: ptr to device control entry D0: result code (word)	

Variables

UTableBase	Base address of unit table
JFetch	Jump vector for Fetch function
JStash	Jump vector for Stash function
JIODone	Jump vector for IODone function
Lvl1DT	Level-1 secondary interrupt vector table (32 bytes)
Lvl2DT	Level-2 secondary interrupt vector table (32 bytes)
VIA	VIA base address
ExtStsDT	External/status interrupt vector table (16 bytes)
SCCWr	SCC write base address
SCCRd	SCC read base address

DIALOG MANAGER

Constants

```
CONST { Item types }

      ctrlItem     = 4;    {add to following four constants}
      btnCtrl      = 0;    {standard button control}
      chkCtrl      = 1;    {standard check box control}
      radCtrl      = 2;    {standard radio button control}
      resCtrl      = 3;    {control defined in control template}
      statText     = 8;    {static text}
      editText     = 16;   {editable text (dialog only)}
      iconItem     = 32;   {icon}
      picItem      = 64;   {QuickDraw picture}
      userItem     = 0;    {application-defined item (dialog only)}
      itemDisable  = 128;  {add to any of above to disable}

      { Item numbers of OK and Cancel buttons }

      ok      = 1;
      cancel  = 2;

      { Resource IDs of alert icons }

      stopIcon     = 0;
      noteIcon     = 1;
      cautionIcon  = 2;
```

Data Types

```
TYPE DialogPtr   = WindowPtr;
     DialogPeek  = ^DialogRecord;

     DialogRecord =
              RECORD
                window:     WindowRecord;  {dialog window}
                items:      Handle;        {item list}
                textH:      TEHandle;      {current editText item}
                editField:  INTEGER;       {editText item number minus 1}
                editOpen:   INTEGER;       {used internally}
                aDefItem:   INTEGER        {default button item number}
              END;

      DialogTHndl  = ^DialogTPtr;
      DialogTPtr   = ^DialogTemplate;
```

```
DialogTemplate  =
        RECORD
          boundsRect: Rect;     {becomes window's portRect}
          procID:     INTEGER;  {window definition ID}
          visible:    BOOLEAN;  {TRUE if visible}
          filler1:    BOOLEAN;  {not used}
          goAwayFlag: BOOLEAN;  {TRUE if has go-away region}
          filler2:    BOOLEAN;  {not used}
          refCon:     LONGINT;  {window's reference value}
          itemsID:    INTEGER;  {resource ID of item list}
          title:      Str255    {window's title}
        END;

AlertTHndl     = ^AlertTPtr;
AlertTPtr      = ^AlertTemplate;
AlertTemplate  = RECORD
                   boundsRect: Rect;      {becomes window's portRect}
                   itemsID:    INTEGER;   {resource ID of item list}
                   stages:     StageList  {alert stage information}
                 END:

StageList =  PACKED RECORD
               boldItm4: 0..1;  {default button item number minus 1}
               boxDrwn4: BOOLEAN;  {TRUE if alert box to be drawn}
               sound4:   0..3      {sound number}
               boldItm3: 0..1;
               boxDrwn3: BOOLEAN;
               sound3:   0..3
               boldItm2: 0..1;
               boxDrwn2: BOOLEAN;
               sound2:   0..3
               boldItm1: 0..1;
               boxDrwn1: BOOLEAN;
               sound1:   0..3
             END;
```

Routines

Initialization

```
PROCEDURE InitDialogs  (resumeProc: ProcPtr);
PROCEDURE ErrorSound   (soundProc: ProcPtr);
PROCEDURE SetDAFont    (fontNum: INTEGER);  [Not in ROM]
```

Creating and Disposing of Dialogs

```
FUNCTION  NewDialog    (dStorage: Ptr; boundsRect: Rect; title: Str255;
                        visible: BOOLEAN; procID: INTEGER; behind:
                        WindowPtr; goAwayFlag: BOOLEAN; refCon: LONGINT;
                        items: Handle) : DialogPtr;
```

```
FUNCTION  GetNewDialog  (dialogID: INTEGER; dStorage: Ptr; behind:
                         WindowPtr) : DialogPtr;
PROCEDURE CloseDialog   (theDialog: DialogPtr);
PROCEDURE DisposDialog  (theDialog: DialogPtr);
PROCEDURE CouldDialog   (dialogID: INTEGER);
PROCEDURE FreeDialog    (dialogID: INTEGER);
```

Handling Dialog Events

```
PROCEDURE ModalDialog    (filterProc: ProcPtr; VAR itemHit: INTEGER);
FUNCTION  IsDialogEvent  (theEvent: EventRecord) : BOOLEAN;
FUNCTION  DialogSelect   (theEvent: EventRecord; VAR theDialog:
                          DialogPtr; VAR itemHit: INTEGER) : BOOLEAN;
PROCEDURE DlgCut         (theDialog: DialogPtr);  [Not in ROM]
PROCEDURE DlgCopy        (theDialog: DialogPtr);  [Not in ROM]
PROCEDURE DlgPaste       (theDialog: DialogPtr);  [Not in ROM]
PROCEDURE DlgDelete      (theDialog: DialogPtr);  [Not in ROM]
PROCEDURE DrawDialog     (theDialog: DialogPtr);
```

Invoking Alerts

```
FUNCTION  Alert         (alertID: INTEGER; filterProc: ProcPtr) : INTEGER;
FUNCTION  StopAlert     (alertID: INTEGER; filterProc: ProcPtr) : INTEGER;
FUNCTION  NoteAlert     (alertID: INTEGER; filterProc: ProcPtr) : INTEGER;
FUNCTION  CautionAlert  (alertID: INTEGER; filterProc: ProcPtr) : INTEGER;
PROCEDURE CouldAlert    (alertID: INTEGER);
PROCEDURE FreeAlert     (alertID: INTEGER);
```

Manipulating Items in Dialogs and Alerts

```
PROCEDURE ParamText       (param0,param1,param2,param3: Str255);
PROCEDURE GetDItem        (theDialog: DialogPtr; itemNo: INTEGER; VAR
                           itemType: INTEGER; VAR item: Handle; VAR box:
                           Rect);
PROCEDURE SetDItem        (theDialog: DialogPtr; itemNo: INTEGER;
                           itemType: INTEGER; item: Handle; box: Rect);
PROCEDURE GetIText        (item: Handle; VAR text: Str255);
PROCEDURE SetIText        (item: Handle; text: Str255);
PROCEDURE SelIText        (theDialog: DialogPtr; itemNo: INTEGER;
                           strtSel,endSel: INTEGER);
FUNCTION  GetAlrtStage :   INTEGER;    [Not in ROM]
PROCEDURE ResetAlrtStage;  [Not in ROM]
```

UserItem Procedure

```
PROCEDURE MyItem (theWindow: WindowPtr; itemNo: INTEGER);
```

Sound Procedure

```
PROCEDURE MySound (soundNo: INTEGER);
```

FilterProc Function for Modal Dialogs and Alerts

```
FUNCTION MyFilter (theDialog: DialogPtr; VAR theEvent: EventRecord;
                   VAR itemHit: INTEGER) : BOOLEAN;
```

Assembly-Language Information

Constants

```
; Item types

ctrlItem      .EQU   4      ;add to following four constants
btnCtrl       .EQU   0      ;standard button control
chkCtrl       .EQU   1      ;standard check box control
radCtrl       .EQU   2      ;standard radio button control
resCtrl       .EQU   3      ;control defined in control template
statText      .EQU   8      ;static text
editText      .EQU   16     ;editable text (dialog only)
iconItem      .EQU   32     ;icon
picItem       .EQU   64     ;QuickDraw picture
userItem      .EQU   0      ;application-defined item (dialog only)
itemDisable   .EQU   128    ;add to any of above to disable

; Item numbers of OK and Cancel buttons

okButton      .EQU   1
cancelButton .EQU    2

; Resource IDs of alert icons

stopIcon      .EQU   0
noteIcon      .EQU   1
cautionIcon   .EQU   2

; Masks for stages word in alert template

volBits       .EQU   3      ;sound number
alBit         .EQU   4      ;whether to draw box
okDismissal   .EQU   8      ;item number of default button minus 1
```

Dialog Record Data Structure

dWindow	Dialog window
items	Handle to dialog's item list
teHandle	Handle to current editText item
editField	Item number of editText item minus 1 (word)
aDefItem	Item number of default button (word)
dWindLen	Size in bytes of dialog record

Dialog Template Data Structure

dBounds	Rectangle that becomes portRect of dialog window's grafPort (8 bytes)
dWindProc	Window definition ID (word)
dVisible	Nonzero if dialog window is visible (word)
dGoAway	Nonzero if dialog window has a go-away region (word)
dRefCon	Dialog window's reference value (long)
dItems	Resource ID of dialog's item list (word)
dTitle	Dialog window's title (preceded by length byte)

Alert Template Data Structure

aBounds	Rectangle that becomes portRect of alert window's grafPort (8 bytes)
aItems	Resource ID of alert's item list (word)
aStages	Stages word; information for alert stages

Item List Data Structure

dlgMaxIndex	Number of items minus 1 (word)
itmHndl	Handle or procedure pointer for this item
itmRect	Display rectangle for this item (8 bytes)
itmType	Item type for this item (byte)
itmData	Length byte followed by data for this item (data must be even number of bytes)

Variables

ResumeProc	Address of resume procedure
DAStrings	Handles to ParamText strings (16 bytes)
DABeeper	Address of current sound procedure
DlgFont	Font number for dialogs and alerts (word)
ACount	Stage number (0 through 3) of last alert (word)
ANumber	Resource ID of last alert (word)

DISK DRIVER

Constants

```
CONST { Positioning modes }

      fsAtMark     = 0;   {at current sector}
      fsFromStart  = 1;   {relative to first sector}
      fsFromMark   = 3;   {relative to current sector}
      rdVerify     = 64;  {add to above for read-verify}
```

Data Types

```
TYPE DrvSts = RECORD
                 track:        INTEGER;     {current track}
                 writeProt:    SignedByte;  {bit 7=1 if volume is locked}
                 diskInPlace:  SignedByte;  {disk in place}
                 installed:    SignedByte;  {drive installed}
                 sides:        SignedByte;  {bit 7=0 if single-sided drive}
                 qLink:        QElemPtr;    {next queue entry}
                 qType:        INTEGER;     {reserved for future use}
                 dQDrive:      INTEGER;     {drive number}
                 dQRefNum:     INTEGER;     {driver reference number}
                 dQFSID:       INTEGER;     {file-system identifier}
                 twoSideFmt:   SignedByte;  {-1 if two-sided disk}
                 needsFlush:   SignedByte;  {reserved for future use}
                 diskErrs:     INTEGER      {error count}
              END;
```

Routines [Not in ROM]

```
FUNCTION DiskEject     (drvNum: INTEGER) : OSErr;
FUNCTION SetTagBuffer (buffPtr: Ptr) : OSErr;
FUNCTION DriveStatus   (drvNum: INTEGER; VAR status: DrvSts) : OSErr;
```

Result Codes

Name	Value	Meaning
noErr	0	No error
nsDrvErr	–56	No such drive
paramErr	–50	Bad positioning information
wPrErr	–44	Volume is locked by a hardware setting

Name	Value	Meaning
firstDskErr	–84	First of the range of low-level disk errors
sectNFErr	–81	Can't find sector
seekErr	–80	Drive error
spdAdjErr	–79	Can't correctly adjust disk speed
twoSideErr	–78	Tried to read side 2 of a disk in a single-sided drive
initIWMErr	–77	Can't initialize disk controller chip
tk0BadErr	–76	Can't find track 0
cantStepErr	–75	Drive error
wrUnderrun	–74	Write underrun occurred
badDBtSlp	–73	Bad data mark
badDCksum	–72	Bad data mark
noDtaMkErr	–71	Can't find data mark
badBtSlpErr	–70	Bad address mark
badCksmErr	–69	Bad address mark
dataVerErr	–68	Read-verify failed
noAdrMkErr	–67	Can't find an address mark
noNybErr	–66	Disk is probably blank
offLinErr	–65	No disk in drive
noDriveErr	–64	Drive isn't connected
lastDskErr	–64	Last of the range of low-level disk errors

Assembly-Language Information

Constants

```
; Positioning modes

fsAtMark      .EQU    0    ;at current sector
fsFromStart   .EQU    1    ;relative to first sector
fsFromMark    .EQU    3    ;relative to current sector
rdVerify      .EQU    64   ;add to above for read-verify

; csCode values for Control/Status calls

ejectCode     .EQU    7    ;Control call, DiskEject
tgBuffCode    .EQU    8    ;Control call, SetTagBuffer
drvStsCode    .EQU    8    ;Status call, DriveStatus
```

Structure of Status Information

dsTrack	Current track (word)
dsWriteProt	Bit 7=1 if volume is locked (byte)
dsDiskInPlace	Disk in place (byte)
dsInstalled	Drive installed (byte)
dsSides	Bit 7=0 if single-sided drive (byte)
dsQLink	Pointer to next queue entry
dsDQDrive	Drive number (word)

dsDQRefNum	Driver reference number (word)
dsDQFSID	File-system identifier (word)
dsTwoSideFmt	–1 if two-sided disk (byte)
dsDiskErrs	Error count (word)

Equivalent Device Manager Calls

Pascal routine	**Call**
DiskEject	Control with csCode=ejectCode
SetTagBuffer	Control with csCode=tgBuffCode
DriveStatus	Status with csCode=drvStsCode, status returned in csParam through csParam+21

Variables

BufTgFNum	File tags buffer: file number (long)
BufTgFFlag	File tags buffer: flags (word: bit 1=1 if resource fork)
BufTgFBkNum	File tags buffer: logical block number (word)
BufTgDate	File tags buffer: date and time of last modification (long)

DISK INITIALIZATION PACKAGE

Routines

```
PROCEDURE DILoad;
PROCEDURE DIUnload;
FUNCTION  DIBadMount  (where: Point; evtMessage: LONGINT) : INTEGER;
FUNCTION  DIFormat    (drvNum: INTEGER) : OSErr;
FUNCTION  DIVerify    (drvNum: INTEGER) : OSErr;
FUNCTION  DIZero      (drvNum: INTEGER; volName: Str255) : OSErr;
```

Result Codes

Name	Value	Meaning
badMDBErr	–60	Bad master directory block
extFSErr	–58	External file system
firstDskErr	–84	First of the range of low-level disk errors
ioErr	–36	I/O error
lastDskErr	–64	Last of the range of low-level disk errors
memFullErr	–108	Not enough room in heap zone
noErr	0	No error
noMacDskErr	–57	Not a Macintosh disk
nsDrvErr	–56	No such drive
paramErr	–50	Bad drive number
volOnLinErr	–55	Volume already on-line

Assembly-Language Information

Constants

```
; Routine selectors

diBadMount      .EQU      0
diLoad          .EQU      2
diUnload        .EQU      4
diFormat        .EQU      6
diVerify        .EQU      8
diZero          .EQU      10
```

Trap Macro Name

_Pack2

EVENT MANAGER, OPERATING SYSTEM

Constants

```
CONST { Event codes }

      nullEvent    = 0;    {null}
      mouseDown    = 1;    {mouse-down}
      mouseUp      = 2;    {mouse-up}
      keyDown      = 3;    {key-down}
      keyUp        = 4;    {key-up}
      autoKey      = 5;    {auto-key}
      updateEvt    = 6;    {update; Toolbox only}
      diskEvt      = 7;    {disk-inserted}
      activateEvt  = 8;    {activate; Toolbox only}
      networkEvt   = 10;   {network}
      driverEvt    = 11;   {device driver}
      app1Evt      = 12;   {application-defined}
      app2Evt      = 13;   {application-defined}
      app3Evt      = 14;   {application-defined}
      app4Evt      = 15;   {application-defined}

      { Masks for keyboard event message }

      charCodeMask  = $000000FF;     {character code}
      keyCodeMask   = $0000FF00;     {key code}

      { Masks for forming event mask }

      mDownMask     =  2;       {mouse-down}
      mUpMask       =  4;       {mouse-up}
      keyDownMask   =  8;       {key-down}
      keyUpMask     =  16;      {key-up}
      autoKeyMask   =  32;      {auto-key}
      updateMask    =  64;      {update}
      diskMask      =  128;     {disk-inserted}
      activMask     =  256;     {activate}
      networkMask   =  1024;    {network}
      driverMask    =  2048;    {device driver}
      app1Mask      =  4096;    {application-defined}
      app2Mask      =  8192;    {application-defined}
      app3Mask      =  16384;   {application-defined}
      app4Mask      = -32768;   {application-defined}
      everyEvent    = -1;       {all event types}
```

```
{ Modifier flags in event record }

activeFlag = 1;      {set if window being activated}
btnState   = 128;    {set if mouse button up}
cmdKey     = 256;    {set if Command key down}
shiftKey   = 512;    {set if Shift key down}
alphaLock  = 1024;   {set if Caps Lock key down}
optionKey  = 2048;   {set if Option key down}

{ Result codes returned by PostEvent }

noErr     =  0;  {no error (event posted)}
evtNotEnb =  1;  {event type not designated in system event mask}
```

Data Types

```
TYPE EventRecord = RECORD
                     what:      INTEGER;   {event code}
                     message:   LONGINT;   {event message}
                     when:      LONGINT;   {ticks since startup}
                     where:     Point;     {mouse location}
                     modifiers: INTEGER    {modifier flags}
                   END;

     EvQEl = RECORD
               qLink:         QElemPtr; {next queue entry}
               qType:         INTEGER;  {queue type}
               evtQWhat:      INTEGER;  {event code}
               evtQMessage:   LONGINT;  {event message}
               evtQWhen:      LONGINT;  {ticks since startup}
               evtQWhere:     Point;    {mouse location}
               evtQModifiers: INTEGER   {modifier flags}
             END;
```

Routines

Posting and Removing Events

```
FUNCTION  PostEvent   (eventCode: INTEGER; eventMsg: LONGINT) : OSErr;
PROCEDURE FlushEvents (eventMask,stopMask: INTEGER);
```

Accessing Events

```
FUNCTION GetOSEvent   (eventMask: INTEGER; VAR theEvent: EventRecord) :
                       BOOLEAN;
FUNCTION OSEventAvail (eventMask: INTEGER; VAR theEvent: EventRecord) :
                       BOOLEAN;
```

Setting the System Event Mask

```
PROCEDURE SetEventMask (theMask: INTEGER);    [Not in ROM]
```

Directly Accessing the Event Queue

```
FUNCTION GetEvQHdr : QHdrPtr;    [Not in ROM]
```

Assembly-Language Information

Constants

```
; Event codes

nullEvt          .EQU      0       ;null
mButDwnEvt       .EQU      1       ;mouse-down
mButUpEvt        .EQU      2       ;mouse-up
keyDwnEvt        .EQU      3       ;key-down
keyUpEvt         .EQU      4       ;key-up
autoKeyEvt       .EQU      5       ;auto-key
updatEvt         .EQU      6       ;update; Toolbox only
diskInsertEvt    .EQU      7       ;disk-inserted
activateEvt      .EQU      8       ;activate; Toolbox only
networkEvt       .EQU      10      ;network
ioDrvrEvt        .EQU      11      ;device driver
app1Evt          .EQU      12      ;application-defined
app2Evt          .EQU      13      ;application-defined
app3Evt          .EQU      14      ;application-defined
app4Evt          .EQU      15      ;application-defined

; Modifier flags in event record

activeFlag       .EQU      0       ;set if window being activated
btnState         .EQU      2       ;set if mouse button up
cmdKey           .EQU      3       ;set if Command key down
shiftKey         .EQU      4       ;set if Shift key down
alphaLock        .EQU      5       ;set if Caps Lock key down
optionKey        .EQU      6       ;set if Option key down

; Result codes returned by PostEvent

noErr            .EQU      0       ;no error (event posted)
evtNotEnb        .EQU      1       ;event type not designated in system
                                   ; event mask
```

Event Record Data Structure

evtNum	Event code (word)
evtMessage	Event message (long)
evtTicks	Ticks since startup (long)
evtMouse	Mouse location (point; long)
evtMeta	State of modifier keys (byte)
evtMBut	State of mouse button (byte)
evtBlkSize	Size in bytes of event record

Event Queue Entry Data Structure

qLink	Pointer to next queue entry
qType	Queue type (word)
evtQWhat	Event code (word)
evtQMessage	Event message (long)
evtQWhen	Ticks since startup (long)
evtQWhere	Mouse location (point; long)
evtQMeta	State of modifier keys (byte)
evtQMBut	State of mouse button (byte)
evtQBlkSize	Size in bytes of event queue entry

Routines

Trap macro	On entry	On exit
_PostEvent	A0: eventCode (word) D0: eventMsg (long)	D0: result code (word)
_FlushEvents	D0: low word: eventMask high word: stopMask	D0: 0 or event code (word)
_GetOSEvent and _OSEventAvail	A0: ptr to event record theEvent D0: eventMask (word)	D0: 0 if non-null event, –1 if null event (byte)

Variables

SysEvtMask	System event mask (word)
EventQueue	Event queue header (10 bytes)

EVENT MANAGER, TOOLBOX

Constants

```
CONST { Event codes }

      nullEvent    = 0;    {null}
      mouseDown    = 1;    {mouse down}
      mouseUp      = 2;    {mouse up}
      keyDown      = 3;    {key-down}
      keyUp        = 4;    {key-up}
      autoKey      = 5;    {auto-key}
      updateEvt    = 6;    {update}
      diskEvt      = 7;    {disk-inserted}
      activateEvt  = 8;    {activate}
      networkEvt   = 10;   {network}
      driverEvt    = 11;   {device driver}
      app1Evt      = 12;   {application-defined}
      app2Evt      = 13;   {application-defined}
      app3Evt      = 14;   {application-defined}
      app4Evt      = 15;   {application-defined}

      { Masks for keyboard event message }

      charCodeMask = $000000FF;  {character code}
      keyCodeMask  = $0000FF00;  {key code}

      { Masks for forming event mask }

      mDownMask    =   2;     {mouse down}
      mUpMask      =   4;     {mouse up}
      keyDownMask  =   8;     {key-down}
      keyUpMask    =   16;    {key-up}
      autoKeyMask  =   32;    {auto-key}
      updateMask   =   64;    {update}
      diskMask     =   128;   {disk-inserted}
      activMask    =   256;   {activate}
      networkMask  =   1024;  {network}
      driverMask   =   2048;  {device driver}
      app1Mask     =   4096;  {application-defined}
      app2Mask     =   8192;  {application-defined}
      app3Mask     =   16384; {application-defined}
      app4Mask     =  -32768; {application-defined}
      everyEvent   =  -1;     {all event types}
```

```
{ Modifier flags in event record }

activeFlag   = 1;    {set if window being activated}
btnState     = 128;  {set if mouse button up}
cmdKey       = 256;  {set if Command key down}
shiftKey     = 512;  {set if Shift key down}
alphaLock    = 1024; {set if Caps Lock key down}
optionKey    = 2048; {set if Option key down}
```

Data Types

```
TYPE EventRecord = RECORD
                     what:       INTEGER;  {event code}
                     message:    LONGINT;  {event message}
                     when:       LONGINT;  {ticks since startup}
                     where:      Point;    {mouse location}
                     modifiers:  INTEGER   {modifier flags}
                   END;

  KeyMap = PACKED ARRAY[0..127] OF BOOLEAN;
```

Routines

Accessing Events

```
FUNCTION GetNextEvent (eventMask: INTEGER; VAR theEvent:
                       EventRecord) : BOOLEAN;
FUNCTION EventAvail   (eventMask: INTEGER; VAR theEvent:
                       EventRecord) : BOOLEAN;
```

Reading the Mouse

```
PROCEDURE  GetMouse     (VAR mouseLoc: Point);
FUNCTION   Button :      BOOLEAN;
FUNCTION   StillDown :   BOOLEAN;
FUNCTION   WaitMouseUp : BOOLEAN;
```

Reading the Keyboard and Keypad

```
PROCEDURE GetKeys (VAR theKeys: KeyMap);
```

Miscellaneous Routines

```
FUNCTION TickCount    : LONGINT;
FUNCTION GetDblTime   : LONGINT;   [Not in ROM]
FUNCTION GetCaretTime : LONGINT;   [Not in ROM]
```

Event Message in Event Record

Event type	Event message
Keyboard	Character code and key code in low-order word
Activate, update	Pointer to window
Disk-inserted	Drive number in low-order word, File Manager result code in high-order word
Mouse-down, mouse-up, null	Undefined
Network	Handle to parameter block
Device driver	See chapter describing driver
Application-defined	Whatever you wish

Assembly-Language Information

Constants

```
; Event codes

nullEvt       .EQU  0    ;null
mButDwnEvt    .EQU  1    ;mouse-down
mButUpEvt     .EQU  2    ;mouse-up
keyDwnEvt     .EQU  3    ;key-down
keyUpEvt      .EQU  4    ;key-up
autoKeyEvt    .EQU  5    ;auto-key
updatEvt      .EQU  6    ;update
diskInsertEvt .EQU  7    ;disk-inserted
activateEvt   .EQU  8    ;activate
networkEvt    .EQU  10   ;network
ioDrvrEvt     .EQU  11   ;device driver
app1Evt       .EQU  12   ;application-defined
app2Evt       .EQU  13   ;application-defined
app3Evt       .EQU  14   ;application-defined
app4Evt       .EQU  15   ;application-defined

; Modifier flags in event record

activeFlag    .EQU  0    ;set if window being activated
btnState      .EQU  2    ;set if mouse button up
cmdKey        .EQU  3    ;set if Command key down
shiftKey      .EQU  4    ;set if Shift key down
alphaLock     .EQU  5    ;set if Caps Lock key down
optionKey     .EQU  6    ;set if Option key down
```

```
; Journaling mechanism Control call

jPlayCtl       .EQU  16   ;journal in playback mode
jRecordCtl     .EQU  17   ;journal in recording mode
jcTickCount    .EQU  0    ;journal code for TickCount
jcGetMouse     .EQU  1    ;journal code for GetMouse
jcButton       .EQU  2    ;journal code for Button
jcGetKeys      .EQU  3    ;journal code for GetKeys
jcEvent        .EQU  4    ;journal code for GetNextEvent and EventAvail
```

Event Record Data Structure

evtNum	Event code (word)
evtMessage	Event message (long)
evtTicks	Ticks since startup (long)
evtMouse	Mouse location (point; long)
evtMeta	State of modifier keys (byte)
evtMBut	State of mouse button (byte)
evtBlkSize	Size in bytes of event record

Variables

KeyThresh	Auto-key threshold (word)
KeyRepThresh	Auto-key rate (word)
WindowList	0 if using events but not windows (long)
ScrDmpEnb	0 if GetNextEvent shouldn't process Command-Shift-number combinations (byte)
Ticks	Current number of ticks since system startup (long)
DoubleTime	Double-click interval in ticks (long)
CaretTime	Caret-blink interval in ticks (long)
JournalRef	Reference number of journaling device driver (word)
JournalFlag	Journaling mode (word)

FILE MANAGER

Constants

```
CONST { Flags in file information used by the Finder }

      fHasBundle =  8192;  {set if file has a bundle}
      fInvisible =  16384; {set if file's icon is invisible}
      fTrash     = -3;     {file is in Trash window}
      fDesktop   = -2;     {file is on desktop}
      fDisk      =  0;     {file is in disk window}

      { Values for requesting read/write access }

      fsCurPerm  = 0;  {whatever is currently allowed}
      fsRdPerm   = 1;  {request to read only}
      fsWrPerm   = 2;  {request to write only}
      fsRdWrPerm = 3;  {request to read and write}

      { Positioning modes }

      fsAtMark    = 0;  {at current mark}
      fsFromStart = 1;  {offset relative to beginning of file}
      fsFromLEOF  = 2;  {offset relative to logical end-of-file}
      fsFromMark  = 3;  {offset relative to current mark}
      rdVerify    = 64; {add to above for read-verify}
```

Data Types

```
TYPE FInfo = RECORD
               fdType:     OSType;  {file type}
               fdCreator:  OSType;  {file's creator}
               fdFlags:    INTEGER; {flags}
               fdLocation: Point;   {file's location}
               fdFldr:     INTEGER  {file's window}
             END;

     ParamBlkType  = (ioParam,fileParam,volumeParam,cntrlParam);

     ParmBlkPtr    = ^ParamBlockRec;
     ParamBlockRec = RECORD
       qLink:        QElemPtr;    {next queue entry}
       qType:        INTEGER;     {queue type}
       ioTrap:       INTEGER;     {routine trap}
       ioCmdAddr:    Ptr;         {routine address}
       ioCompletion: ProcPtr;     {completion routine}
       ioResult:     OSErr;       {result code}
       ioNamePtr:    StringPtr;   {volume or file name}
       ioVRefNum:    INTEGER;     {volume reference or drive number}
```

```
CASE ParamBlkType OF
 ioParam:
  (ioRefNum:    INTEGER;    {path reference number}
   ioVersNum:   SignedByte; {version number}
   ioPermssn:   SignedByte; {read/write permission}
   ioMisc:      Ptr;        {miscellaneous}
   ioBuffer:    Ptr;        {data buffer}
   ioReqCount:  LONGINT;    {requested number of bytes}
   ioActCount:  LONGINT;    {actual number of bytes}
   ioPosMode:   INTEGER;    {positioning mode and newline character}
   ioPosOffset: LONGINT);   {positioning offset}
 fileParam:
  (ioFRefNum:    INTEGER;    {path reference number}
   ioFVersNum:   SignedByte; {version number}
   filler1:      SignedByte; {not used}
   ioFDirIndex:  INTEGER;    {sequence number of file}
   ioFlAttrib:   SignedByte; {file attributes}
   ioFlVersNum:  SignedByte  {version number}
   ioFlFndrInfo: FInfo;      {information used by the Finder}
   ioFlNum:      LONGINT;    {file number}
   ioFlStBlk:    INTEGER;    {first allocation block of data fork}
   ioFlLgLen:    LONGINT;    {logical end-of-file of data fork}
   ioFlPyLen:    LONGINT;    {physical end-of-file of data fork}
   ioFlRStBlk:   INTEGER;    {first allocation block of resource }
                             { fork}
   ioFlRLgLen:   LONGINT;    {logical end-of-file of resource fork}
   ioFlRPyLen:   LONGINT;    {physical end-of-file of resource }
                             { fork}
   ioFlCrDat:    LONGINT;    {date and time of creation}
   ioFlMdDat:    LONGINT);   {date and time of last modification}
 volumeParam:
  (filler2:      LONGINT;    {not used}
   ioVolIndex:   INTEGER;    {volume index}
   ioVCrDate:    LONGINT;    {date and time of initialization}
   ioVLsBkUp:    LONGINT;    {date and time of last backup}
   ioVAtrb:      INTEGER;    {bit 15=1 if volume locked}
   ioVNmFls:     INTEGER;    {number of files in directory}
   ioVDirSt:     INTEGER;    {first block of directory}
   ioVBlLn:      INTEGER;    {length of directory in blocks}
   ioVNmAlBlks:  INTEGER;    {number of allocation blocks}
   ioVAlBlkSiz:  LONGINT;    {size of allocation blocks}
   ioVClpSiz:    LONGINT;    {number of bytes to allocate}
   ioAlBlSt:     INTEGER;    {first allocation block in block map}
   ioVNxtFNum:   LONGINT;    {next unused file number}
   ioVFrBlk:     INTEGER);   {number of unused allocation blocks}
 cntrlParam:
  . . .  {used by Device Manager}
 END;
```

```
VCB = RECORD
         qLink:        QElemPtr;    {next queue entry}
         qType:        INTEGER;     {queue type}
         vcbFlags:     INTEGER;     {bit 15=1 if dirty}
         vcbSigWord:   INTEGER;     {always $D2D7}
         vcbCrDate:    LONGINT;     {date and time of initialization}
         vcbLsBkUp:    LONGINT;     {date and time of last backup}
         vcbAtrb:      INTEGER;     {volume attributes}
         vcbNmFls:     INTEGER;     {number of files in directory}
         vcbDirSt:     INTEGER;     {first block of directory}
         vcbBlLn:      INTEGER;     {length of directory in blocks}
         vcbNmBlks:    INTEGER;     {number of allocation blocks}
         vcbAlBlkSiz:  LONGINT;     {size of allocation blocks}
         vcbClpSiz:    LONGINT;     {number of bytes to allocate}
         vcbAlBlSt:    INTEGER;     {first allocation block in block map}
         vcbNxtFNum:   LONGINT;     {next unused file number}
         vcbFreeBks:   INTEGER;     {number of unused allocation blocks}
         vcbVN:        STRING[27];  {volume name}
         vcbDrvNum     INTEGER;     {drive number}
         vcbDRefNum:   INTEGER;     {driver reference number}
         vcbFSID:      INTEGER;     {file-system identifier}
         vcbVRefNum:   INTEGER;     {volume reference number}
         vcbMAdr:      Ptr;         {pointer to block map}
         vcbBufAdr:    Ptr;         {pointer to volume buffer}
         vcbMLen:      INTEGER;     {number of bytes in block map}
         vcbDirIndex:  INTEGER;     {used internally}
         vcbDirBlk:    INTEGER      {used internally}
      END;

DrvQEl = RECORD
            qLink:      QElemPtr; {next queue entry}
            qType:      INTEGER;  {queue type}
            dQDrive:    INTEGER;  {drive number}
            dQRefNum:   INTEGER;  {driver reference number}
            dQFSID:     INTEGER;  {file-system identifier}
            dQDrvSize:  INTEGER   {number of logical blocks}
         END;
```

High-Level Routines [Not in ROM]

Accessing Volumes

```
FUNCTION GetVInfo     (drvNum: INTEGER; volName: StringPtr; VAR vRefNum:
                       INTEGER; VAR freeBytes: LONGINT) : OSErr;
FUNCTION GetVRefNum   (pathRefNum: INTEGER; VAR vRefNum: INTEGER) : OSErr;
FUNCTION GetVol       (volName: StringPtr; VAR vRefNum: INTEGER) : OSErr;
FUNCTION SetVol       (volName: StringPtr; vRefNum: INTEGER) : OSErr;
FUNCTION FlushVol     (volName: StringPtr; vRefNum: INTEGER) : OSErr;
FUNCTION UnmountVol   (volName: StringPtr; vRefNum: INTEGER) : OSErr;
FUNCTION Eject        (volName: StringPtr; vRefNum: INTEGER) : OSErr;
```

Accessing Files

```
FUNCTION Create    (fileName: Str255; vRefNum: INTEGER; creator: OSType;
                    fileType: OSType) : OSErr;
FUNCTION FSOpen    (fileName: Str255; vRefNum: INTEGER; VAR refNum:
                    INTEGER) : OSErr;
FUNCTION OpenRF    (fileName: Str255; vRefNum: INTEGER; VAR refNum:
                    INTEGER) : OSErr;
FUNCTION FSRead    (refNum: INTEGER; VAR count: LONGINT; buffPtr: Ptr) :
                    OSErr;
FUNCTION FSWrite   (refNum: INTEGER; VAR count: LONGINT; buffPtr: Ptr) :
                    OSErr;
FUNCTION GetFPos   (refNum: INTEGER; VAR filePos: LONGINT) : OSErr;
FUNCTION SetFPos   (refNum: INTEGER; posMode: INTEGER; posOff: LONGINT) :
                    OSErr;
FUNCTION GetEOF    (refNum: INTEGER; VAR logEOF: LONGINT) : OSErr;
FUNCTION SetEOF    (refNum: INTEGER; logEOF: LONGINT) : OSErr;
FUNCTION Allocate  (refNum: INTEGER; VAR count: LONGINT) : OSErr;
FUNCTION FSClose   (refNum: INTEGER) : OSErr;
```

Changing Information About Files

```
FUNCTION GetFInfo  (fileName: Str255; vRefNum: INTEGER; VAR fndrInfo:
                    FInfo) : OSErr;
FUNCTION SetFInfo  (fileName: Str255; vRefNum: INTEGER; fndrInfo: FInfo):
                    OSErr;
FUNCTION SetFLock  (fileName: Str255; vRefNum: INTEGER) : OSErr;
FUNCTION RstFLock  (fileName: Str255; vRefNum: INTEGER) : OSErr;
FUNCTION Rename    (oldName: Str255; vRefNum: INTEGER; newName: Str255) :
                    OSErr;
FUNCTION FSDelete  (fileName: Str255; vRefNum: INTEGER) : OSErr;
```

Low-Level Routines

Initializing the File I/O Queue

```
PROCEDURE FInitQueue;
```

Accessing Volumes

```
FUNCTION PBMountVol (paramBlock: ParmBlkPtr) : OSErr;
```

←	16	ioResult	word
↔	22	ioVRefNum	word

```
FUNCTION PBGetVInfo (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
↔	18	ioNamePtr	pointer
↔	22	ioVRefNum	word
→	28	ioVolIndex	word
←	30	ioVCrDate	long word
←	34	ioVLsBkUp	long word
←	38	ioVAtrb	word
←	40	ioVNmFls	word
←	42	ioVDirSt	word
←	44	ioVBlLn	word
←	46	ioVNmAlBlks	word
←	48	ioVAlBlkSiz	long word
←	52	ioVClpSiz	long word
←	56	ioAlBlSt	word
←	58	ioVNxtFNum	long word
←	62	ioVFrBlk	word

```
FUNCTION PBGetVol (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
←	18	ioNamePtr	pointer
←	22	ioVRefNum	word

```
FUNCTION PBSetVol (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word

```
FUNCTION PBFlushVol (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word

```
FUNCTION PBUnmountVol (paramBlock: ParmBlkPtr) : OSErr;
```

←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word

```
FUNCTION PBOffLine (paramBlock: ParmBlkPtr) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word

FUNCTION PBEject (paramBlock: ParmBlkPtr) : OSErr;

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePt	pointer
→	22	ioVRefNum	word

Accessing Files

FUNCTION PBCreate (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word
→	26	ioFVersNum	byte

FUNCTION PBOpen (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word
←	24	ioRefNum	word
→	26	ioVersNum	byte
→	27	ioPermssn	byte
→	28	ioMisc	pointer

FUNCTION PBOpenRF (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word
←	24	ioRefNum	word
→	26	ioVersNum	byte
→	27	ioPermssn	byte
→	28	ioMisc	pointer

FUNCTION PBRead (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;

→	12	ioCompletion	pointer
←	16	ioResult	word
→	24	ioRefNum	word
→	32	ioBuffer	pointer
→	36	ioReqCount	long word
←	40	ioActCount	long word
→	44	ioPosMode	word
↔	46	ioPosOffset	long word

```
FUNCTION PBWrite (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	24	ioRefNum	word
→	32	ioBuffer	pointer
→	36	ioReqCount	long word
←	40	ioActCount	long word
→	44	ioPosMode	word
↔	46	ioPosOffset	long word

```
FUNCTION PBGetFPos (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	24	ioRefNum	word
←	36	ioReqCount	long word
←	40	ioActCount	long word
←	44	ioPosMode	word
←	46	ioPosOffset	long word

```
FUNCTION PBSetFPos (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	24	ioRefNum	word
→	44	ioPosMode	word
↔	46	ioPosOffset	long word

```
FUNCTION PBGetEOF (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	24	ioRefNum	word
←	28	ioMisc	long word

```
FUNCTION PBSetEOF (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	24	ioRefNum	word
→	28	ioMisc	long word

```
FUNCTION PBAllocate (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	24	ioRefNum	word
→	36	ioReqCount	long word
←	40	ioActCount	long word

```
FUNCTION PBFlushFile (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	24	ioRefNum	word

```
FUNCTION PBClose (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	24	ioRefNum	word

Changing Information About Files

```
FUNCTION PBGetFInfo (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
↔	18	ioNamePtr	pointer
→	22	ioVRefNum	word
←	24	ioFRefNum	word
→	26	ioFVersNum	byte
→	28	ioFDirIndex	word
←	30	ioFlAttrib	byte
←	31	ioFlVersNum	byte
←	32	ioFlFndrInfo	16 bytes
←	48	ioFlNum	long word
←	52	ioFlStBlk	word
←	54	ioFlLgLen	long word
←	58	ioFlPyLen	long word
←	62	ioFlRStBlk	word
←	64	ioFlRLgLen	long word
←	68	ioFlRPyLen	long word
←	72	ioFlCrDat	long word
←	76	ioFlMdDat	long word

```
FUNCTION PBSetFInfo (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word
→	26	ioFVersNum	byte
→	32	ioFlFndrInfo	16 bytes
→	72	ioFlCrDat	long word
→	76	ioFlMdDat	long word

```
FUNCTION PBSetFLock (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word
→	26	ioFVersNum	byte

```
FUNCTION PBRstFLock (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
```

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
→	22	ioVRefNum	word
→	26	ioFVersNum	byte

```
FUNCTION PBSetFVers (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
    →  12    ioCompletion    pointer
    ←  16    ioResult        word
    →  18    ioNamePtr       pointer
    →  22    ioVRefNum       word
    →  26    ioVersNum       byte
    →  28    ioMisc          byte

FUNCTION PBRename (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
    →  12    ioCompletion    pointer
    ←  16    ioResult        word
    →  18    ioNamePtr       pointer
    →  22    ioVRefNum       word
    →  26    ioVersNum       byte
    →  28    ioMisc          pointer

FUNCTION PBDelete (paramBlock: ParmBlkPtr; async: BOOLEAN) : OSErr;
    →  12    ioCompletion    pointer
    ←  16    ioResult        word
    →  18    ioNamePtr       pointer
    →  22    ioVRefNum       word
    →  26    ioFVersNum      byte
```

Accessing Queues [Not in ROM]

```
FUNCTION GetFSQHdr  : QHdrPtr;
FUNCTION GetVCBQHdr : QHdrPtr;
FUNCTION GetDrvQHdr : QHdrPtr;
```

Result Codes

Name	Value	Meaning
badMDBErr	–60	Master directory block is bad; must reinitialize volume
bdNamErr	–37	Bad file name or volume name (perhaps zero-length)
dirFulErr	–33	File directory full
dskFulErr	–34	All allocation blocks on the volume are full
dupFNErr	–48	A file with the specified name and version number already exists
eofErr	–39	Logical end-of-file reached during read operation
extFSErr	–58	External file system; file-system identifier is nonzero, or path reference number is greater than 1024
fBsyErr	–47	One or more files are open
fLckdErr	–45	File locked
fnfErr	–43	File not found
fnOpnErr	–38	File not open

Name	Value	Meaning
fsRnErr	–59	Problem during rename
gfpErr	–52	Error during GetFPos
ioErr	–36	I/O error
memFullErr	–108	Not enough room in heap zone
noErr	0	No error
noMacDskErr	–57	Volume lacks Macintosh-format directory
nsDrvErr	–56	Specified drive number doesn't match any number in the drive queue
nsvErr	–35	Specified volume doesn't exist
opWrErr	–49	The read/write permission of only one access path to a file can allow writing
paramErr	–50	Parameters don't specify an existing volume, and there's no default volume
permErr	–54	Attempt to open locked file for writing
posErr	–40	Attempt to position before start of file
rfNumErr	–51	Reference number specifies nonexistent access path
tmfoErr	–42	Too many files open
volOffLinErr	–53	Volume not on-line
volOnLinErr	–55	Specified volume is already mounted and on-line
vLckdErr	–46	Volume is locked by a software flag
wrPermErr	–61	Read/write permission doesn't allow writing
wPrErr	–44	Volume is locked by a hardware setting

Assembly-Language Information

Constants

```
; Flags in file information used by the Finder

fHasBundle      .EQU     13     ;set if file has a bundle
fInvisible      .EQU     14     ;set if file's icon is invisible

; Flags in trap words

asnycTrpBit     .EQU     10     ;set for an asynchronous call
noQueueBit      .EQU     9      ;set for immediate execution
```

```
; Values for requesting read/write access

fsCurPerm      .EQU    0     ;whatever is currently allowed
fsRdPerm       .EQU    1     ;request to read only
fsWrPerm       .EQU    2     ;request to write only
fsRdWrPerm     .EQU    3     ;request to read and write

; Positioning modes

fsAtMark       .EQU    0     ;at current mark
fsFromStart    .EQU    1     ;offset relative to beginning of file
fsFromLEOF     .EQU    2     ;offset relative to logical end-of-file
fsFromMark     .EQU    3     ;offset relative to current mark
rdVerify       .EQU    64    ;add to above for read-verify
```

Structure of File Information Used by the Finder

fdType	File type (long)
fdCreator	File's creator (long)
fdFlags	Flags (word)
fdLocation	File's location (point; long)
fdFldr	File's window (word)

Standard Parameter Block Data Structure

qLink	Pointer to next queue entry
qType	Queue type (word)
ioTrap	Routine trap (word)
ioCmdAddr	Routine address
ioCompletion	Address of completion routine
ioResult	Result code (word)
ioFileName	Pointer to file name (preceded by length byte)
ioVNPtr	Pointer to volume name (preceded by length byte)
ioVRefNum	Volume reference number (word)
ioDrvNum	Drive number (word)

I/O Parameter Block Data Structure

ioRefNum	Path reference number (word)
ioFileType	Version number (byte)
ioPermssn	Read/write permission (byte)
ioNewName	Pointer to new file or volume name for Rename
ioLEOF	Logical end-of-file for SetEOF (long)
ioOwnBuf	Pointer to access path buffer
ioNewType	New version number for SetFilType (byte)
ioBuffer	Pointer to data buffer
ioReqCount	Requested number of bytes (long)
ioActCount	Actual number of bytes (long)
ioPosMode	Positioning mode and newline character (word)

ioPosOffset	Positioning offset (long)
ioQElSize	Size in bytes of I/O parameter block

Structure of File Information Parameter Block

ioRefNum	Path reference number (word)
ioFileType	Version number (byte)
ioFDirIndex	Sequence number of file (word)
ioFlAttrib	File attributes (byte)
ioFFlType	Version number (byte)
ioFlUsrWds	Information used by the Finder (16 bytes)
ioFFlNum	File number (long)
ioFlStBlk	First allocation block of data fork (word)
ioFlLgLen	Logical end-of-file of data fork (long)
ioFlPyLen	Physical end-of-file of data fork (long)
ioFlRStBlk	First allocation block of resource fork (word)
ioFlRLgLen	Logical end-of-file of resource fork (long)
ioFlRPyLen	Physical end-of-file of resource fork (long)
ioFlCrDat	Date and time of creation (long)
ioFlMdDat	Date and time of last modification (long)
ioFQElSize	Size in bytes of file information parameter block

Structure of Volume Information Parameter Block

ioVolIndex	Volume index (word)
ioVCrDate	Date and time of initialization (long)
ioVLsBkUp	Date and time of last backup (long)
ioVAtrb	Volume attributes (word)
ioVNmFls	Number of files in directory (word)
ioVDirSt	First block of directory (word)
ioVBlLn	Length of directory in blocks (word)
ioVNmAlBlks	Number of allocation blocks on volume (word)
ioVAlBlkSiz	Size of allocation blocks (long)
ioVClpSiz	Number of bytes to allocate (long)
ioAlBlSt	First allocation block in block map (word)
ioVNxtFNum	Next unused file number (long)
ioVFrBlk	Number of unused allocation blocks (word)
ioVQElSize	Size in bytes of volume information parameter block

Volume Information Data Structure

drSigWord	Always $D2D7 (word)
drCrDate	Date and time of initialization (long)
drLsBkUp	Date and time of last backup (long)
drAtrb	Volume attributes (word)
drNmFls	Number of files in directory (word)
drDirSt	First block of directory (word)
drBlLn	Length of directory in blocks (word)
drNmAlBlks	Number of allocation blocks on volume (word)
drAlBlkSiz	Size of allocation blocks (long)

drClpSiz	Number of bytes to allocate (long)
drAlBlSt	First allocation block in block map (word)
drNxtFNum	Next unused file number (long)
drFreeBks	Number of unused allocation blocks (word)
drVN	Volume name preceded by length byte (28 bytes)

File Directory Entry Data Structure

flFlags	Bit 7=1 if entry used; bit 0=1 if file locked (byte)
flTyp	Version number (byte)
flUsrWds	Information used by the Finder (16 bytes)
flFlNum	File number (long)
flStBlk	First allocation block of data fork (word)
flLgLen	Logical end-of-file of data fork (long)
flPyLen	Physical end-of-file of data fork (long)
flRStBlk	First allocation block of resource fork (word)
flRLgLen	Logical end-of-file of resource fork (long)
flRPyLen	Physical end-of-file of resource fork (long)
flCrDat	Date and time file of creation (long)
flMdDat	Date and time of last modification (long)
flNam	File name preceded by length byte

Volume Control Block Data Structure

qLink	Pointer to next queue entry
qType	Queue type (word)
vcbFlags	Bit 15=1 if volume control block is dirty (word)
vcbSigWord	Always $D2D7 (word)
vcbCrDate	Date and time of initialization (word)
vcbLsBkUp	Date and time of last backup (long)
vcbAtrb	Volume attributes (word)
vcbNmFls	Number of files in directory (word)
vcbDirSt	First block of directory (word)
vcbBlLn	Length of directory in blocks (word)
vcbNmBlks	Number of allocation blocks on volume (word)
vcbAlBlkSiz	Size of allocation blocks (long)
vcbClpSiz	Number of bytes to allocate (long)
vcbAlBlSt	First allocation block in block map (word)
vcbNxtFNum	Next unused file number (long)
vcbFreeBks	Number of unused allocation blocks (word)
vcbVN	Volume name preceded by length byte (28 bytes)
vcbDrvNum	Drive number of drive in which volume is mounted (word)
vcbDRefNum	Driver reference number of driver for drive in which volume is mounted (word)
vcbFSID	File-system identifier (word)
vcbVRefNum	Volume reference number (word)
vcbMAdr	Pointer to volume block map
vcbBufAdr	Pointer to volume buffer
vcbMLen	Number of bytes in volume block map (word)

File Control Block Data Structure

fcbFlNum	File number (long)
fcbMdRByt	Flags (byte)
fcbTypByt	Version number (byte)
fcbSBlk	First allocation block of file (word)
fcbEOF	Logical end-of-file (long)
fcbPLen	Physical end-of-file (long)
fcbCrPs	Mark (long)
fcbVPtr	Pointer to volume control block (long)
fcbBfAdr	Pointer to access path buffer (long)

Drive Queue Entry Data Structure

qLink	Pointer to next queue entry
qType	Queue type (word)
dQDrive	Drive number (word)
dQRefNum	Driver reference number (word)
dQFSID	File-system identifier (word)
dQDrvSize	Number of logical blocks (word)

Macro Names

Pascal name	**Macro name**
FInitQueue	_InitQueue
PBMountVol	_MountVol
PBGetVInfo	_GetVolInfo
PBGetVol	_GetVol
PBSetVol	_SetVol
PBFlushVol	_FlushVol
PBUnmountVol	_UnmountVol
PBOffLine	_OffLine
PBEject	_Eject
PBCreate	_Create
PBOpen	_Open
PBOpenRF	_OpenRF
PBRead	_Read
PBWrite	_Write
PBGetFPos	_GetFPos
PBSetFPos	_SetFPos
PBGetEOF	_GetEOF
PBSetEOF	_SetEOF
PBAllocate	_Allocate
PBFlushFile	_FlushFile
PBClose	_Close
PBGetFInfo	_GetFileInfo
PBSetFInfo	_SetFileInfo
PBSetFLock	_SetFilLock
PBRstFLock	_RstFilLock

PBSetFVers	_SetFilType
PBRename	_Rename
PBDelete	_Delete

Variables

FSQHdr	File I/O queue header (10 bytes)
VCBQHdr	Volume-control-block queue header (10 bytes)
DefVCBPtr	Pointer to default volume control block
FCBSPtr	Pointer to file-control-block buffer
DrvQHdr	Drive queue header (10 bytes)
ToExtFS	Pointer to external file system

FONT MANAGER

Constants

```
CONST { Font numbers }

      systemFont  = 0;    {system font}
      applFont    = 1;    {application font}
      newYork     = 2;
      geneva      = 3;
      monaco      = 4;
      venice      = 5;
      london      = 6;
      athens      = 7;
      sanFran     = 8;
      toronto     = 9;
      cairo       = 11;
      losAngeles  = 12;
      times       = 20;
      helvetica   = 21;
      courier     = 22;
      symbol      = 23;
      taliesin    = 24;

      { Special characters }

      commandMark  = $11;   {Command key symbol}
      checkMark    = $12;   {check mark}
      diamondMark  = $13;   {diamond symbol}
      appleMark    = $14;   {apple symbol}

      { Font types }

      propFont   = $9000;   {proportional font}
      fixedFont  = $B000;   {fixed-width font}
      fontWid    = $ACB0;   {font width data}
```

Data Types

```
TYPE FMInput = PACKED RECORD
                 family:     INTEGER;   {font number}
                 size:       INTEGER;   {font size}
                 face:       Style;     {character style}
                 needBits:   BOOLEAN;   {TRUE if drawing}
                 device:     INTEGER;   {device-specific information}
                 numer:      Point;     {numerators of scaling factors}
                 denom:      Point      {denominators of scaling factors}
               END;
```

```
FMOutPtr = ^FMOutput;
FMOutput =
       PACKED RECORD
         errNum:      INTEGER;      {not used}
         fontHandle:  Handle;       {handle to font record}
         bold:        Byte;         {bold factor}
         italic:      Byte;         {italic factor}
         ulOffset:    Byte;         {underline offset}
         ulShadow:    Byte;         {underline shadow}
         ulThick:     Byte;         {underline thickness}
         shadow:      Byte;         {shadow factor}
         extra:       SignedByte;   {width of style}
         ascent:      Byte;         {ascent}
         descent:     Byte;         {descent}
         widMax:      Byte;         {maximum character width}
         leading:     SignedByte;   {leading}
         unused:      Byte;         {not used}
         numer:       Point;        {numerators of scaling factors}
         denom:       Point         {denominators of scaling factors}
       END;

 FontRec =
      RECORD
        fontType:     INTEGER;   {font type}
        firstChar:    INTEGER;   {ASCII code of first character}
        lastChar:     INTEGER;   {ASCII code of last character}
        widMax:       INTEGER;   {maximum character width}
        kernMax:      INTEGER;   {negative of maximum character kern}
        nDescent:     INTEGER;   {negative of descent}
        fRectWidth:   INTEGER;   {width of font rectangle}
        fRectHeight:  INTEGER;   {height of font rectangle}
        owTLoc:       INTEGER;   {offset to offset/width table}
        ascent:       INTEGER;   {ascent}
        descent:      INTEGER;   {descent}
        leading:      INTEGER;   {leading}
        rowWords:     INTEGER;   {row width of bit image / 2}
      { bitImage:     ARRAY[1..rowWords,1..fRectHeight] OF INTEGER; }
                                 {bit image}
      { locTable:     ARRAY[firstChar..lastChar+2] OF INTEGER; }
                                 {location table}
      { owTable:      ARRAY[firstChar..lastChar+2] OF INTEGER; }
                                 {offset/width table}
      END;
```

Routines

Initializing the Font Manager

```
PROCEDURE InitFonts;
```

Getting Font Information

```
PROCEDURE GetFontName (fontNum: INTEGER; VAR theName: Str255);
PROCEDURE GetFNum     (fontName: Str255; VAR theNum: INTEGER);
FUNCTION  RealFont    (fontNum: INTEGER; size: INTEGER) : BOOLEAN;
```

Keeping Fonts in Memory

```
PROCEDURE SetFontLock (lockFlag: BOOLEAN);
```

Advanced Routine

```
FUNCTION FMSwapFont (inRec: FMInput) : FMOutPtr;
```

Assembly-Language Information

Constants

```
; Font numbers

sysFont       .EQU   0      ;system font
applFont      .EQU   1      ;application font
newYork       .EQU   2
geneva        .EQU   3
monaco        .EQU   4
venice        .EQU   5
london        .EQU   6
athens        .EQU   7
sanFran       .EQU   8
toronto       .EQU   9
cairo         .EQU   11
losAngeles    .EQU   12
times         .EQU   20
helvetica     .EQU   21
courier       .EQU   22
symbol        .EQU   23
taliesin      .EQU   24

; Special characters

commandMark   .EQU   $11    ;Command key symbol
checkMark     .EQU   $12    ;check mark
diamondMark   .EQU   $13    ;diamond symbol
appleMark     .EQU   $14    ;apple symbol

; Font types

propFont      .EQU   $9000 ;proportional font
```

```
fixedFont     .EQU    $B000 ;fixed-width font
fontWid       .EQU    $ACB0 ;font width data

; Control and Status call code

fMgrCtl1      .EQU    8     ;code used to get and modify font
                            ; characterization table
```

Font Input Record Data Structure

fmInFamily	Font number (word)
fmInSize	Font size (word)
fmInFace	Character style (word)
fmInNeedBits	Nonzero if drawing (byte)
fmInDevice	Device-specific information (byte)
fmInNumer	Numerators of scaling factors (point; long)
fmInDenom	Denominators of scaling factors (point; long)

Font Output Record Data Structure

fmOutFontH	Handle to font record
fmOutBold	Bold factor (byte)
fmOutItalic	Italic factor (byte)
fmOutUlOffset	Underline offset (byte)
fmOutUlShadow	Underline shadow (byte)
fmOutUlThick	Underline thickness (byte)
fmOutShadow	Shadow factor (byte)
fmOutExtra	Width of style (byte)
fmOutAscent	Ascent (byte)
fmOutDescent	Descent (byte)
fmOutWidMax	Maximum character width (byte)
fmOutLeading	Leading (byte)
fmOutNumer	Numerators of scaling factors (point; long)
fmOutDenom	Denominators of scaling factors (point; long)

Font Record Data Structure

fFontType	Font type (word)
fFirstChar	ASCII code of first character (word)
fLastChar	ASCII code of last character (word)
fWidMax	Maximum character width (word)
fKernMax	Negative of maximum character kern (word)
fNDescent	Negative of descent (word)
fFRectWidth	Width of font rectangle (word)
fFRectHeight	Height of font rectangle (word)
fOWTLoc	Offset to offset/width table (word)
fAscent	Ascent (word)
fDescent	Descent (word)
fLeading	Leading (word)

fRowWords	Row width of bit image / 2 (word)

Special Macro Names

Pascal name	**Macro name**
GetFontName	_GetFName

Variables

ApFontID	Font number of application font (word)
FScaleDisable	Nonzero to disable scaling (byte)
ROMFont0	Handle to font record for system font

INTERNATIONAL UTILITIES PACKAGE

Constants

```
CONST { Masks for currency format }

      currSymLead    = 16;  {set if currency symbol leads}
      currNegSym     = 32;  {set if minus sign for negative}
      currTrailingZ  = 64;  {set if trailing decimal zeroes}
      currLeadingZ   = 128; {set if leading integer zero}

      { Order of short date elements }

      mdy = 0;      {month day year}
      dmy = 1;      {day month year}
      ymd = 2;      {year month day}

      { Masks for short date format }

      dayLdingZ = 32;  {set if leading zero for day}
      mntLdingZ = 64;  {set if leading zero for month}
      century   = 128; {set if century included}

      { Masks for time format }

      secLeadingZ  = 32;  {set if leading zero for seconds}
      minLeadingZ  = 64;  {set if leading zero for minutes}
      hrLeadingZ   = 128; {set if leading zero for hours}

      { High-order byte of version information }

      verUS           = 0;
      verFrance       = 1;
      verBritain      = 2;
      verGermany      = 3;
      verItaly        = 4;
      verNetherlands  = 5;
      verBelgiumLux   = 6;
      verSweden       = 7;
      verSpain        = 8;
      verDenmark      = 9;
      verPortugal     = 10;
      verFrCanada     = 11;
      verNorway       = 12;
      verIsrael       = 13;
      verJapan        = 14;
      verAustralia    = 15;
      verArabia       = 16;
      verFinland      = 17;
```

```
verFrSwiss      = 18;
verGrSwiss      = 19;
verGreece       = 20;
verIceland      = 21;
verMalta        = 22;
verCyprus       = 23;
verTurkey       = 24;
verYugoslavia   = 25;
```

Data Types

```
TYPE Intl0Hndl  = ^Intl0Ptr;
     Intl0Ptr   = ^Intl0Rec;
     Intl0Rec   =
          PACKED RECORD
            decimalPt:    CHAR; {decimal point character}
            thousSep:     CHAR; {thousands separator}
            listSep:      CHAR; {list separator}
            currSym1:     CHAR; {currency symbol}
            currSym2:     CHAR;
            currSym3:     CHAR;
            currFmt:      Byte; {currency format}
            dateOrder:    Byte; {order of short date elements}
            shrtDateFmt:  Byte; {short date format}
            dateSep:      CHAR; {date separator}
            timeCycle:    Byte; {0 if 24-hour cycle, 255 if 12-hour}
            timeFmt:      Byte; {time format}
            mornStr:      PACKED ARRAY[1..4] OF CHAR;
                                {trailing string for first 12-hour cycle}
            eveStr:       PACKED ARRAY[1..4] OF CHAR;
                                {trailing string for last 12-hour cycle}
            timeSep:      CHAR; {time separator}
            time1Suff:    CHAR; {trailing string for 24-hour cycle}
            time2Suff:    CHAR;
            time3Suff:    CHAR;
            time4Suff:    CHAR;
            time5Suff:    CHAR;
            time6Suff:    CHAR;
            time7Suff:    CHAR;
            time8Suff:    CHAR;
            metricSys:    Byte;   {255 if metric, 0 if not}
            intl0Vers:    INTEGER {version information}
          END;
```

```
Intl1Hndl = ^Intl1Ptr;
Intl1Ptr = ^Intl1Rec;
Intl1Rec =
         PACKED RECORD
           days:         ARRAY[1..7] OF STRING[15];   {day names}
           months:       ARRAY[1..12] OF STRING[15];  {month names}
           suppressDay: Byte;  {0 for day name, 255 for none}
           lngDateFmt:  Byte;  {order of long date elements}
           dayLeading0: Byte;  {255 for leading 0 in day number}
           abbrLen:     Byte;  {length for abbreviating names}
           st0:         PACKED ARRAY[1..4] OF CHAR;  {strings }
           st1:         PACKED ARRAY[1..4] OF CHAR;  { for }
           st2:         PACKED ARRAY[1..4] OF CHAR;  { long }
           st3:         PACKED ARRAY[1..4] OF CHAR;  { date }
           st4:         PACKED ARRAY[1..4] OF CHAR;  { format}
           intl1Vers:   INTEGER; {version information}
           localRtn:    INTEGER  {routine for localizing string }
                                 { comparison; actually may be }
                                 { longer than one integer}
         END;

DateForm = (shortDate,longDate,abbrevDate);
```

Routines

```
PROCEDURE IUDateString  (dateTime: LONGINT; form: DateForm; VAR result:
                         Str255);
PROCEDURE IUDatePString (dateTime: LONGINT; form: DateForm; VAR result:
                         Str255; intlParam: Handle);
PROCEDURE IUTimeString  (dateTime: LONGINT; wantSeconds: BOOLEAN; VAR
                         result: Str255);
PROCEDURE IUTimePString (dateTime: LONGINT; wantSeconds: BOOLEAN; VAR
                         result: Str255; intlParam: Handle);
FUNCTION  IUMetric :     BOOLEAN;
FUNCTION  IUGetIntl     (theID: INTEGER) : Handle;
PROCEDURE IUSetIntl     (refNum: INTEGER; theID: INTEGER; intlParam:
                         Handle);
FUNCTION  IUCompString  (aStr,bStr: Str255) : INTEGER;  [Not in ROM]
FUNCTION  IUMagString   (aPtr,bPtr: Ptr; aLen,bLen: INTEGER) : INTEGER;
FUNCTION  IUEqualString (aStr,bStr: Str255) : INTEGER;  [Not in ROM]
FUNCTION  IUMagIDString (aPtr,bPtr: Ptr; aLen,bLen: INTEGER) : INTEGER;
```

Assembly-Language Information

Constants

```
; Masks for currency format

currSymLead    .EQU   16    ;set if currency symbol leads
currNegSym     .EQU   32    ;set if minus sign for negative
currTrailingZ  .EQU   64    ;set if trailing decimal zeroes
currLeadingZ   .EQU   128   ;set if leading integer zero

; Order of short date elements

mdy            .EQU   0     ;month day year
dmy            .EQU   1     ;day month year
ymd            .EQU   2     ;year month day

; Masks for short date format

dayLdingZ      .EQU   32    ;set if leading zero for day
mntLdingZ      .EQU   64    ;set if leading zero for month
century        .EQU   128   ;set if century included

; Masks for time format

secLeadingZ    .EQU   32    ;set if leading zero for seconds
minLeadingZ    .EQU   64    ;set if leading zero for minutes
hrLeadingZ     .EQU   128   ;set if leading zero for hours

; High-order byte of version information

verUS           .EQU   0
verFrance       .EQU   1
verBritain      .EQU   2
verGermany      .EQU   3
verItaly        .EQU   4
verNetherlands  .EQU   5
verBelgiumLux   .EQU   6
verSweden       .EQU   7
verSpain        .EQU   8
verDenmark      .EQU   9
verPortugal     .EQU   10
verFrCanada     .EQU   11
verNorway       .EQU   12
verIsrael       .EQU   13
verJapan        .EQU   14
verAustralia    .EQU   15
verArabia       .EQU   16
verFinland      .EQU   17
verFrSwiss      .EQU   18
verGrSwiss      .EQU   19
```

```
verGreece      .EQU   20
verIceland     .EQU   21
verMalta       .EQU   22
verCyprus      .EQU   23
verTurkey      .EQU   24
verYugoslavia  .EQU   25

; Date form for IUDateString and IUDatePString

shortDate     .EQU    0      ;short form of date
longDate      .EQU    1      ;long form of date
abbrevDate    .EQU    2      ;abbreviated long form

; Routine selectors

iuDateString  .EQU    0
iuTimeString  .EQU    2
iuMetric      .EQU    4
iuGetIntl     .EQU    6
iuSetIntl     .EQU    8
iuMagString   .EQU    10
iuMagIDString .EQU    12
iuDatePString .EQU    14
iuTimePString .EQU    16
```

International Resource 0 Data Structure

decimalPt	Decimal point character (byte)
thousSep	Thousands separator (byte)
listSep	List separator (byte)
currSym	Currency symbol (3 bytes)
currFmt	Currency format (byte)
dateOrder	Order of short date elements (byte)
shrtDateFmt	Short date format (byte)
dateSep	Date separator (byte)
timeCycle	0 if 24-hour cycle, 255 if 12-hour (byte)
timeFmt	Time format (byte)
mornStr	Trailing string for first 12-hour cycle (long)
eveStr	Trailing string for last 12-hour cycle (long)
timeSep	Time separator (byte)
timeSuff	Trailing string for 24-hour cycle (8 bytes)
metricSys	255 if metric, 0 if not (byte)
intl0Vers	Version information (word)

International Resource 1 Data Structure

days	Day names (112 bytes)
months	Month names (192 bytes)
suppressDay	0 for day name, 255 for none (byte)
lngDateFmt	Order of long date elements (byte)

dayLeading0	255 for leading 0 in day number (byte)
abbrLen	Length for abbreviating names (byte)
st0	Strings for long date format (longs)
st1	
st2	
st3	
st4	
intl1Vers	Version information (word)
localRtn	Comparison localization routine

Trap Macro Name

_Pack6

MEMORY MANAGER

Constants

```
CONST { Result codes }

     memFullErr   = -108;    {not enough room in heap zone}
     memLockedErr = -117;    {block is locked}
     memPurErr    = -112;    {attempt to purge a locked block}
     memWZErr     = -111;    {attempt to operate on a free block}
     nilHandleErr = -109;    {NIL master pointer}
     noErr        =   0;     {no error}
```

Data Types

```
TYPE SignedByte  = -128..127;
     Byte        = 0..255;
     Ptr         = ^SignedByte;
     Handle      = ^Ptr;

     Str255        = STRING[255];
     StringPtr     = ^Str255;
     StringHandle  = ^StringPtr;

     ProcPtr = Ptr;

     Fixed = LONGINT;

     Size = LONGINT;

     THz  = ^Zone;
     Zone = RECORD
              bkLim:        Ptr;         {zone trailer block}
              purgePtr:     Ptr;         {used internally}
              hFstFree:     Ptr;         {first free master pointer}
              zcbFree:      LONGINT;     {number of free bytes}
              gzProc:       ProcPtr;     {grow zone function}
              moreMast:     INTEGER;     {master pointers to allocate}
              flags:        INTEGER;     {used internally}
              cntRel:       INTEGER;     {not used}
              maxRel:       INTEGER;     {not used}
              cntNRel:      INTEGER;     {not used}
              maxNRel:      INTEGER;     {not used}
              cntEmpty:     INTEGER;     {not used}
              cntHandles:   INTEGER;     {not used}
              minCBFree:    LONGINT;     {not used}
              purgeProc:    ProcPtr;     {purge warning procedure}
              sparePtr:     Ptr;         {used internally}
              allocPtr:     Ptr;         {used internally}
              heapData:     INTEGER      {first usable byte in zone}
            END;
```

Routines

Initialization and Allocation

```
PROCEDURE InitApplZone;
PROCEDURE SetApplBase    (startPtr: Ptr);
PROCEDURE InitZone       (pGrowZone: ProcPtr; cMoreMasters: INTEGER;
                          limitPtr,startPtr: Ptr);
FUNCTION  GetApplLimit :  Ptr;   [Not in ROM]
PROCEDURE SetApplLimit   (zoneLimit: Ptr);
PROCEDURE MaxApplZone;   [Not in ROM]
PROCEDURE MoreMasters;
```

Heap Zone Access

```
FUNCTION  GetZone :     THz;
PROCEDURE SetZone      (hz: THz);
FUNCTION  SystemZone : THz;   [Not in ROM]
FUNCTION  ApplicZone : THz;   [Not in ROM]
```

Allocating and Releasing Relocatable Blocks

```
FUNCTION  NewHandle      (logicalSize: Size) : Handle;
PROCEDURE DisposHandle   (h: Handle);
FUNCTION  GetHandleSize  (h: Handle) : Size;
PROCEDURE SetHandleSize  (h: Handle; newSize: Size);
FUNCTION  HandleZone     (h: Handle) : THz;
FUNCTION  RecoverHandle  (p: Ptr) : Handle;
PROCEDURE ReallocHandle  (h: Handle; logicalSize: Size);
```

Allocating and Releasing Nonrelocatable Blocks

```
FUNCTION  NewPtr     (logicalSize: Size) : Ptr;
PROCEDURE DisposPtr  (p: Ptr);
FUNCTION  GetPtrSize (p: Ptr) : Size;
PROCEDURE SetPtrSize (p: Ptr; newSize: Size);
FUNCTION  PtrZone    (p: Ptr) : THz;
```

Freeing Space in the Heap

```
FUNCTION  FreeMem :    LONGINT;
FUNCTION  MaxMem      (VAR grow: Size) : Size;
FUNCTION  CompactMem  (cbNeeded: Size) : Size;
PROCEDURE ResrvMem    (cbNeeded: Size);
PROCEDURE PurgeMem    (cbNeeded: Size);
PROCEDURE EmptyHandle (h: Handle);
```

Properties of Relocatable Blocks

```
PROCEDURE HLock     (h: Handle);
PROCEDURE HUnlock   (h: Handle);
PROCEDURE HPurge    (h: Handle);
PROCEDURE HNoPurge  (h: Handle);
```

Grow Zone Operations

```
PROCEDURE SetGrowZone  (growZone: ProcPtr);
FUNCTION  GZSaveHnd :   Handle;  [Not in ROM]
```

Miscellaneous Routines

```
PROCEDURE BlockMove   (sourcePtr,destPtr: Ptr; byteCount: Size);
FUNCTION  TopMem :     Ptr;  [Not in ROM]
PROCEDURE MoveHHi     (h: Handle);  [Not in ROM]
FUNCTION  MemError :   OSErr;  [Not in ROM]
```

Grow Zone Function

```
FUNCTION MyGrowZone (cbNeeded: Size) : LONGINT;
```

Assembly-Language Information

Constants

```
; Values for tag byte of a block header

tyBkFree       .EQU    0     ;free block
tyBkNRel       .EQU    1     ;nonrelocatable block
tyBkRel        .EQU    2     ;relocatable block

; Flags for the high-order byte of a master pointer

lock           .EQU    7     ;lock bit
purge          .EQU    6     ;purge bit
resourc        .EQU    5     ;resource bit

; Result codes

memFullErr     .EQU    -108    ;not enough room in heap zone
memLockedErr   .EQU    -117    ;block is locked
memPurErr      .EQU    -112    ;attempt to purge a locked block
memWZErr       .EQU    -111    ;attempt to operate on a free block
nilHandleErr   .EQU    -109    ;NIL master pointer
noErr          .EQU     0      ;no error
```

Zone Record Data Structure

bkLim	Pointer to zone trailer block
hFstFree	Pointer to first free master pointer
zcbFree	Number of free bytes (long)
gzProc	Address of grow zone function
mAllocCnt	Master pointers to allocate (word)
purgeProc	Address of purge warning procedure
heapData	First usable byte in zone

Block Header Data Structure

tagBC	Tag byte and physical block size (long)
handle	Relocatable block: relative handle Nonrelocatable block: zone pointer
blkData	First byte of block contents

Parameter Block Structure for InitZone

startPtr	Pointer to first byte in zone
limitPtr	Pointer to first byte beyond end of zone
cMoreMasters	Number of master pointers for zone (word)
pGrowZone	Address of grow zone function

Routines

Trap macro	On entry	On exit
_InitApplZone		D0: result code (word)
_SetApplBase	A0: startPtr (ptr)	D0: result code (word)
_InitZone	A0: ptr to parameter block 0 startPtr (ptr) 4 limitPtr (ptr) 8 cMoreMasters (word) 10 pGrowZone (ptr)	D0: result code (word)
_SetApplLimit	A0: zoneLimit (ptr)	D0: result code (word)
_MoreMasters		
_GetZone		A0: function result (ptr) D0: result code (word)
_SetZone	A0: hz (ptr)	D0: result code (word)
_NewHandle	D0: logicalSize (long)	A0: function result (handle) D0: result code (word)
_DisposHandle	A0: h (handle)	D0: result code (word)

Trap macro	**On entry**	**On exit**
_GetHandleSize	A0: h (handle)	D0: if >=0, function result (long) if <0, result code (word)
_SetHandleSize	A0: h (handle) D0: newSize (long)	D0: result code (word)
_HandleZone	A0: h (handle)	A0: function result (ptr) D0: result code (word)
_RecoverHandle	A0: p (ptr)	A0: function result (handle) D0: unchanged
_ReallocHandle	A0: h (handle) D0: logicalSize (long)	D0: result code (word)
_NewPtr	D0: logicalSize (long)	A0: function result (ptr) D0: result code (word)
_DisposPtr	A0: p (ptr)	D0: result code (word)
_GetPtrSize	A0: p (ptr)	D0: if >=0, function result (long) if <0, result code (word)
_SetPtrSize	A0: p (ptr) D0: newSize (long)	D0: result code (word)
_PtrZone	A0: p (ptr)	A0: function result (ptr) D0: result code (word)
_FreeMem		D0: function result (long)
_MaxMem		D0: function result (long) A0: grow (long)
_CompactMem	D0: cbNeeded (long)	D0: function result (long)
_ResrvMem	D0: cbNeeded (long)	D0: result code (word)
_PurgeMem	D0: cbNeeded (long)	D0: result code (word)
_EmptyHandle	A0: h (handle)	A0: h (handle) D0: result code (word)
_HLock	A0: h (handle)	D0: result code (word)
_HUnlock	A0: h (handle)	D0: result code (word)
_HPurge	A0: h (handle)	D0: result code (word)
_HNoPurge	A0: h (handle)	D0: result code (word)
_SetGrowZone	A0: growZone (ptr)	D0: result code (word)
_BlockMove	A0: sourcePtr (ptr) A1: destPtr (ptr) D0: byteCount (long)	D0: result code (word)

Variables

DefltStack	Default space allotment for stack (long)
MinStack	Minimum space allotment for stack (long)
MemTop	Address of end of RAM (on Macintosh XL, end of RAM available to applications)
ScrnBase	Address of main screen buffer
BufPtr	Address of end of jump table
CurrentA5	Address of boundary between application globals and application parameters
CurStackBase	Address of base of stack; start of application globals
ApplLimit	Application heap limit
HeapEnd	Address of end of application heap zone
ApplZone	Address of application heap zone
SysZone	Address of system heap zone
TheZone	Address of current heap zone
GZRootHnd	Handle to relocatable block not to be moved by grow zone function

MENU MANAGER

Constants

```
CONST { Value indicating item has no mark }

      noMark = 0;

      { Messages to menu definition procedure }

      mDrawMsg    = 0;      {draw the menu}
      mChooseMsg  = 1;      {tell which item was chosen and highlight it}
      mSizeMsg    = 2;      {calculate the menu's dimensions}

      { Resource ID of standard menu definition procedure }

      textMenuProc = 0;
```

Data Types

```
TYPE MenuHandle = ^MenuPtr;
     MenuPtr    = ^MenuInfo;
     MenuInfo   = RECORD
                    menuID:       INTEGER;   {menu ID}
                    menuWidth:    INTEGER;   {menu width in pixels}
                    menuHeight:   INTEGER;   {menu height in pixels}
                    menuProc:     Handle;    {menu definition procedure}
                    enableFlags:  LONGINT;   {tells if menu or items are }
                                             { enabled}
                    menuData:     Str255     {menu title (and other data)}
                  END;
```

Routines

Initialization and Allocation

```
PROCEDURE InitMenus;
FUNCTION  NewMenu      (menuID: INTEGER; menuTitle: Str255) : MenuHandle;
FUNCTION  GetMenu      (resourceID: INTEGER) : MenuHandle;
PROCEDURE DisposeMenu  (theMenu: MenuHandle);
```

Forming the Menus

```
PROCEDURE AppendMenu     (theMenu: MenuHandle; data: Str255);
PROCEDURE AddResMenu     (theMenu: MenuHandle; theType: ResType);
PROCEDURE InsertResMenu  (theMenu: MenuHandle; theType: ResType;
                          afterItem: INTEGER);
```

Forming the Menu Bar

```
PROCEDURE InsertMenu  (theMenu: MenuHandle; beforeID: INTEGER);
PROCEDURE DrawMenuBar;
PROCEDURE DeleteMenu  (menuID: INTEGER);
PROCEDURE ClearMenuBar;
FUNCTION  GetNewMBar  (menuBarID: INTEGER) : Handle;
FUNCTION  GetMenuBar : Handle;
PROCEDURE SetMenuBar  (menuList: Handle);
```

Choosing From a Menu

```
FUNCTION  MenuSelect (startPt: Point) : LONGINT;
FUNCTION  MenuKey    (ch: CHAR) : LONGINT;
PROCEDURE HiliteMenu (menuID: INTEGER);
```

Controlling the Appearance of Items

```
PROCEDURE SetItem      (theMenu: MenuHandle; item: INTEGER; itemString:
                        Str255);
PROCEDURE GetItem      (theMenu: MenuHandle; item: INTEGER; VAR
                        itemString: Str255);
PROCEDURE DisableItem  (theMenu: MenuHandle; item: INTEGER);
PROCEDURE EnableItem   (theMenu: MenuHandle; item: INTEGER);
PROCEDURE CheckItem    (theMenu: MenuHandle; item: INTEGER; checked:
                        BOOLEAN);
PROCEDURE SetItemMark  (theMenu: MenuHandle; item: INTEGER; markChar:
                        CHAR);
PROCEDURE GetItemMark  (theMenu: MenuHandle; item: INTEGER; VAR
                        markChar: CHAR);
PROCEDURE SetItemIcon  (theMenu: MenuHandle; item: INTEGER; icon: Byte);
PROCEDURE GetItemIcon  (theMenu: MenuHandle; item: INTEGER; VAR icon:
                        Byte);
PROCEDURE SetItemStyle (theMenu: MenuHandle; item: INTEGER; chStyle:
                        Style);
PROCEDURE GetItemStyle (theMenu: MenuHandle; item: INTEGER; VAR chStyle:
                        Style);
```

Miscellaneous Routines

```
PROCEDURE CalcMenuSize (theMenu: MenuHandle);
FUNCTION  CountMItems  (theMenu: MenuHandle) : INTEGER;
FUNCTION  GetMHandle   (menuID: INTEGER) : MenuHandle;
PROCEDURE FlashMenuBar (menuID: INTEGER);
PROCEDURE SetMenuFlash (count: INTEGER);
```

Meta-Characters for AppendMenu

Meta-character	Usage
; or Return	Separates multiple items
^	Followed by an icon number, adds that icon to the item
!	Followed by a character, marks the item with that character
<	Followed by B, I, U, O, or S, sets the character style of the item
/	Followed by a character, associates a keyboard equivalent with the item
(	Disables the item

Menu Definition Procedure

```
PROCEDURE MyMenu (message: INTEGER; theMenu: MenuHandle; VAR menuRect:
                  Rect; hitPt: Point; VAR whichItem: INTEGER);
```

Assembly-Language Information

Constants

```
; Value indicating item has no mark

noMark          .EQU      0

; Messages to menu definition procedure

mDrawMsg      .EQU    0     ;draw the menu
mChooseMsg    .EQU    1     ;tell which item was chosen and highlight it
mSizeMsg      .EQU    2     ;calculate the menu's dimensions

; Resource ID of standard menu definition procedure

textMenuProc    .EQU      0
```

Menu Record Data Structure

menuID	Menu ID (word)
menuWidth	Menu width in pixels (word)
menuHeight	Menu height in pixels (word)
menuDefHandle	Handle to menu definition procedure
menuEnable	Enable flags (long)
menuData	Menu title (preceded by length byte) followed by data defining the items
menuBlkSize	Size in bytes of menu record except menuData field

Special Macro Names

Pascal name	Macro name
DisposeMenu	_DisposMenu
GetItemIcon	_GetItmIcon
GetItemMark	_GetItmMark
GetItemStyle	_GetItmStyle
GetMenu	_GetRMenu
SetItemIcon	_SetItmIcon
SetItemMark	_SetItmMark
SetItemStyle	_SetItmStyle
SetMenuFlash	_SetMFlash

Variables

MenuList	Handle to current menu list
MBarEnable	Nonzero if menu bar belongs to a desk accessory (word)
MenuHook	Address of routine called repeatedly during MenuSelect
MBarHook	Address of routine called by MenuSelect before menu is drawn (see below)
TheMenu	Menu ID of currently highlighted menu (word)
MenuFlash	Count for duration of menu item blinking (word)

MBarHook routine

On entry	stack: pointer to menu rectangle
On exit	D0: 0 to continue MenuSelect 1 to abort MenuSelect

PACKAGE MANAGER

Constants

```
CONST { Resource IDs for packages }

      dskInit = 2;      {Disk Initialization}
      stdFile = 3;      {Standard File}
      flPoint = 4;      {Floating-Point Arithmetic}
      trFunc  = 5;      {Transcendental Functions}
      intUtil = 6;      {International Utilities}
      bdConv  = 7;      {Binary-Decimal Conversion}
```

Routines

```
PROCEDURE InitPack      (packID: INTEGER);
PROCEDURE InitAllPacks;
```

Assembly-Language Information

Constants

```
; Resource IDs for packages

dskInit     .EQU     2     ;Disk Initialization
stdFile     .EQU     3     ;Standard File
flPoint     .EQU     4     ;Floating-Point Arithmetic
trFunc      .EQU     5     ;Transcendental Functions
intUtil     .EQU     6     ;International Utilities
bdConv      .EQU     7     ;Binary-Decimal Conversion
```

Trap Macros for Packages

Disk Initialization	_Pack2	
Standard File	_Pack3	
Floating-Point Arithmetic	_Pack4	(synonym: _FP68K)
Transcendental Functions	_Pack5	(synonym: _Elems68K)
International Utilities	_Pack6	
Binary-Decimal Conversion	_Pack7	

PRINTING MANAGER

Constants

```
CONST { Printing methods }

      bDraftLoop = 0;     {draft printing}
      bSpoolLoop = 1;     {spool printing}

      { Printer specification in prStl field of print record }

      bDevCItoh = 1;      {Imagewriter printer}
      bDevLaser = 3;      {LaserWriter printer}

      { Maximum number of pages in a spool file }

      iPFMaxPgs = 128;

      { Result codes }

      noErr        =  0;     {no error}
      iPrSavPFil   = -1;     {saving spool file}
      controlErr   = -17;    {unimplemented control instruction}
      iIOAbort     = -27;    {I/O abort error}
      iMemFullErr  = -108;   {not enough room in heap zone}
      iPrAbort     =  128;   {application or user requested abort}

      { PrCtlCall parameters }

      iPrDevCtl    = 7;           {printer control}
      lPrReset     = $00010000;   {reset printer}
      lPrLineFeed  = $00030000;   {carriage return only}
      lPrLFSixth   = $0003FFFF;   {standard 1/6-inch line feed}
      lPrPageEnd   = $00020000;   {end page}
      iPrBitsCtl   = 4;           {bit map printing}
      lScreenBits  = 0;              {default for printer}
      lPaintBits   = 1;              {square dots (72 by 72)}
      iPrIOCtl     = 5;           {text streaming}

      { Printer Driver information }

      sPrDrvr    = '.Print';    {Printer Driver resource name}
      iPrDrvrRef = -3;          {Printer Driver reference number}
```

Data Types

```
TYPE TPPrPort = ^TPrPort;
     TPrPort  = RECORD
                  gPort: GrafPort;  {grafPort to draw in}
                  {more fields for internal use}
                END;

     THPrint =  ^TPPrint;
     TPPrint =  ^TPrint;
     TPrint  =  RECORD
                 iPrVersion:  INTEGER;  {Printing Manager version}
                 prInfo:      TPrInfo;  {printer information subrecord}
                 rPaper:      Rect;     {paper rectangle}
                 prStl:       TPrStl;   {additional device information}
                 prInfoPT:    TPrInfo;  {used internally}
                 prXInfo:     TPrXInfo; {additional device information}
                 prJob:       TPrJob;   {job subrecord}
                 printX:      ARRAY[1..19] OF INTEGER  {not used}
                END;

     TPrInfo =  RECORD
                 iDev:   INTEGER;   {used internally}
                 iVRes:  INTEGER;   {vertical resolution of printer}
                 iHRes:  INTEGER;   {horizontal resolution of printer}
                 rPage:  Rect       {page rectangle}
                END;

     TPrJob =
         RECORD
           iFstPage:  INTEGER;      {first page to print}
           iLstPage:  INTEGER;      {last page to print}
           iCopies:   INTEGER;      {number of copies}
           bJDocLoop: SignedByte;   {printing method}
           fFromUsr:  BOOLEAN;      {used internally}
           pIdleProc: ProcPtr;      {background procedure}
           pFileName: StringPtr;    {spool file name}
           iFileVol:  INTEGER;      {spool file volume reference number}
           bFileVers: SignedByte;   {spool file version number}
           bJobX:     SignedByte    {used internally}
         END;

     TPrStl = RECORD
                wDev: INTEGER;    {high byte specifies device}
                {more fields for internal use}
              END;
```

```
TPrXInfo = RECORD
              iRowBytes: INTEGER;     {used internally}
              iBandV:    INTEGER;     {used internally}
              iBandH:    INTEGER;     {used internally}
              iDevBytes: INTEGER;     {size of buffer}
              {more fields for internal use}
            END;

TPRect = ^Rect;

TPrStatus =  RECORD
                iTotPages:  INTEGER;  {number of pages in spool file}
                iCurPage:   INTEGER;  {page being printed}
                iTotCopies: INTEGER;  {number of copies requested}
                iCurCopy:   INTEGER;  {copy being printed}
                iTotBands:  INTEGER;  {used internally}
                iCurBand:   INTEGER;  {used internally}
                fPgDirty:   BOOLEAN;  {TRUE if started printing page}
                fImaging:   BOOLEAN;  {used internally}
                hPrint:     THPrint;  {print record}
                pPrPort:    TPPrPort; {printing grafPort}
                hPic:       PicHandle {used internally}
              END;
```

Routines [Not in ROM]

Initialization and Termination

```
PROCEDURE PrOpen;
PROCEDURE PrClose;
```

Print Records and Dialogs

```
PROCEDURE PrintDefault  (hPrint: THPrint);
FUNCTION  PrValidate    (hPrint: THPrint) : BOOLEAN;
FUNCTION  PrStlDialog   (hPrint: THPrint) : BOOLEAN;
FUNCTION  PrJobDialog   (hPrint: THPrint) : BOOLEAN;
PROCEDURE PrJobMerge    (hPrintSrc,hPrintDst: THPrint);
```

Printing

```
FUNCTION  PrOpenDoc    (hPrint: THPrint; pPrPort: TPPrPort; pIOBuf: Ptr) :
                        TPPrPort;
PROCEDURE PrOpenPage   (pPrPort: TPPrPort; pPageFrame: TPRect);
PROCEDURE PrClosePage  (pPrPort: TPPrPort);
PROCEDURE PrCloseDoc   (pPrPort: TPPrPort);
PROCEDURE PrPicFile    (hPrint: THPrint; pPrPort: TPPrPort; pIOBuf: Ptr;
                        pDevBuf: Ptr; VAR prStatus: TPrStatus);
```

Error Handling

```
FUNCTION  PrError :    INTEGER;
PROCEDURE PrSetError  (iErr: INTEGER);
```

Low-Level Driver Access

```
PROCEDURE PrDrvrOpen;
PROCEDURE PrDrvrClose;
PROCEDURE PrCtlCall     (iWhichCtl: INTEGER; lParam1,lParam2,lParam3:
                         LONGINT);
FUNCTION  PrDrvrDCE :    Handle;
FUNCTION  PrDrvrVers :   INTEGER;
```

Assembly-Language Information

Constants

```
; Printing methods

bDraftLoop   .EQU   0      ;draft printing
bSpoolLoop   .EQU   1      ;spool printing

; Result codes

noErr        .EQU    0     ;no error
iPrSavPFil   .EQU   -1     ;saving spool file
controlErr   .EQU   -17    ;unimplemented control instruction
iIOAbort     .EQU   -27    ;I/O abort error
iMemFullErr  .EQU   -108   ;not enough room in heap zone
iPrAbort     .EQU    128   ;application or user requested abort

; Printer Driver Control call parameters

iPrDevCtl    .EQU   7      ;printer control
lPrReset     .EQU   1      ; reset printer
iPrLineFeed  .EQU   3      ; carriage return/paper advance
iPrLFSixth   .EQU   3      ;standard 1/6-inch line feed
lPrPageEnd   .EQU   2      ; end page
iPrBitsCtl   .EQU   4      ;bit map printing
lScreenBits  .EQU   0      ; default for printer
lPaintBits   .EQU   1      ; square dots (72 by 72)
iPrIOCtl     .EQU   5      ;text streaming

; Printer Driver information

iPrDrvrRef   .EQU   -3     ;Printer Driver reference number
```

Printing GrafPort Data Structure

gPort	GrafPort to draw in (portRec bytes)
iPrPortSize	Size in bytes of printing grafPort

Print Record Data Structure

iPrVersion	Printing Manager version (word)
prInfo	Printer information subrecord (14 bytes)
rPaper	Paper rectangle (8 bytes)
prStl	Additional device information (8 bytes)
prXInfo	Additional device information (16 bytes)
prJob	Job subrecord (iPrJobSize bytes)
iPrintSize	Size in bytes of print record

Structure of Printer Information Subrecord

iVRes	Vertical resolution of printer (word)
iHRes	Horizontal resolution of printer (word)
rPage	Page rectangle (8 bytes)

Structure of Job Subrecord

iFstPage	First page to print (word)
iLstPage	Last page to print (word)
iCopies	Number of copies (word)
bJDocLoop	Printing method (byte)
pIdleProc	Address of background procedure
pFileName	Pointer to spool file name (preceded by length byte)
iFileVol	Spool file volume reference number (word)
bFileVers	Spool file version number (byte)
iPrJobSize	Size in bytes of job subrecord

Structure of PrXInfo Subrecord

iDevBytes	Size of buffer (word)

Structure of Printer Status Record

iTotPages	Number of pages in spool file (word)
iCurPage	Page being printed (word)
iTotCopies	Number of copies requested (word)
iCurCopy	Copy being printed (word)
fPgDirty	Nonzero if started printing page (byte)
hPrint	Handle to print record
pPrPort	Pointer to printing grafPort
iPrStatSize	Size in bytes of printer status record

Variables

PrintErr	Result code from last Printing Manager routine (word)

QUICKDRAW

Constants

```
CONST { Source transfer modes }

      srcCopy    = 0;
      srcOr      = 1;
      srcXor     = 2;
      srcBic     = 3;
      notSrcCopy = 4;
      notSrcOr   = 5;
      notSrcXor  = 6;
      notSrcBic  = 7;

      { Pattern transfer modes }

      patCopy    = 8;
      patOr      = 9;
      patXor     = 10;
      patBic     = 11;
      notPatCopy = 12;
      notPatOr   = 13;
      notPatXor  = 14;
      notPatBic  = 15;

      { Standard colors for ForeColor and BackColor }

      blackColor   = 33;
      whiteColor   = 30;
      redColor     = 205;
      greenColor   = 341;
      blueColor    = 409;
      cyanColor    = 273;
      magentaColor = 137;
      yellowColor  = 69;

      { Standard picture comments }

      picLParen = 0;
      picRParen = 1;
```

Data Types

```
TYPE StyleItem = (bold,italic,underline,outline,shadow,condense,extend);
     Style     = SET OF StyleItem;
```

```
VHSelect = (v,h);
Point    = RECORD CASE INTEGER OF

             0: (v: INTEGER;  {vertical coordinate}
                 h: INTEGER); {horizontal coordinate}

             1: (vh: ARRAY[VHSelect] OF INTEGER)

           END;

Rect = RECORD CASE INTEGER OF

         0: (top:    INTEGER;
             left:   INTEGER;
             bottom: INTEGER;
             right:  INTEGER);

         1: (topLeft:  Point;
             botRight: Point)

       END;

RgnHandle = ^RgnPtr;
RgnPtr    = ^Region;
Region    = RECORD
              rgnSize: INTEGER;  {size in bytes}
              rgnBBox: Rect;     {enclosing rectangle}
              {more data if not rectangular}
            END;

BitMap = RECORD
           baseAddr: Ptr;      {pointer to bit image}
           rowBytes: INTEGER;  {row width}
           bounds:   Rect      {boundary rectangle}
         END;

Pattern = PACKED ARRAY[0..7] OF 0..255;

Bits16 = ARRAY[0..15] OF INTEGER;

Cursor = RECORD
           data:     Bits16;  {cursor image}
           mask:     Bits16;  {cursor mask}
           hotSpot:  Point    {point aligned with mouse}
         END;
```

```
QDProcsPtr = ^QDProcs;
QDProcs    = RECORD
               textProc:     Ptr;  {text drawing}
               lineProc:     Ptr;  {line drawing}
               rectProc:     Ptr;  {rectangle drawing}
               rRectProc:    Ptr;  {roundRect drawing}
               ovalProc:     Ptr;  {oval drawing}
               arcProc:      Ptr;  {arc/wedge drawing}
               rgnProc:      Ptr;  {region drawing}
               bitsProc:     Ptr;  {bit transfer}
               commentProc:  Ptr;  {picture comment processing}
               txMeasProc:   Ptr;  {text width measurement}
               getPicProc:   Ptr;  {picture retrieval}
               putPicProc:   Ptr   {picture saving}
             END;

GrafPtr  = ^GrafPort;
GrafPort = RECORD
             device:      INTEGER;     {device-specific information}
             portBits:    BitMap;      {grafPort's bit map}
             portRect:    Rect;        {grafPort's rectangle}
             visRgn:      RgnHandle;   {visible region}
             clipRgn:     RgnHandle;   {clipping region}
             bkPat:       Pattern;     {background pattern}
             fillPat:     Pattern;     {fill pattern}
             pnLoc:       Point;       {pen location}
             pnSize:      Point;       {pen size}
             pnMode:      INTEGER;     {pen's transfer mode}
             pnPat:       Pattern;     {pen pattern}
             pnVis:       INTEGER;     {pen visibility}
             txFont:      INTEGER;     {font number for text}
             txFace:      Style;       {text's character style}
             txMode:      INTEGER;     {text's transfer mode}
             txSize:      INTEGER;     {font size for text}
             spExtra:     Fixed;       {extra space}
             fgColor:     LONGINT;     {foreground color}
             bkColor:     LONGINT;     {background color}
             colrBit:     INTEGER;     {color bit}
             patStretch:  INTEGER;     {used internally}
             picSave:     Handle;      {picture being saved}
             rgnSave:     Handle;      {region being saved}
             polySave:    Handle;      {polygon being saved}
             grafProcs:   QDProcsPtr   {low-level drawing routines}
           END;

PicHandle = ^PicPtr;
PicPtr    = ^Picture;
Picture   = RECORD
              picSize:    INTEGER;  {size in bytes}
              picFrame:   Rect;     {picture frame}
              {picture definition data}
            END;
```

```
     PolyHandle = ^PolyPtr;
     PolyPtr    = ^Polygon;
     Polygon    = RECORD
                     polySize:    INTEGER;  {size in bytes}
                     polyBBox:    Rect;     {enclosing rectangle}
                     polyPoints:  ARRAY[0..0] OF Point
                  END;

     PenState  = RECORD
                    pnLoc:   Point;     {pen location}
                    pnSize:  Point;     {pen size}
                    pnMode:  INTEGER;   {pen's transfer mode}
                    pnPat:   Pattern    {pen pattern}
                 END;

     FontInfo  = RECORD
                    ascent:   INTEGER;  {ascent}
                    descent:  INTEGER;  {descent}
                    widMax:   INTEGER;  {maximum character width}
                    leading:  INTEGER   {leading}
                 END;

     GrafVerb  = (frame,paint,erase,invert,fill);
```

Variables

```
VAR thePort:     GrafPtr;   {pointer to current grafPort}
    white:       Pattern;   {all-white pattern}
    black:       Pattern;   {all-black pattern}
    gray:        Pattern;   {50% gray pattern}
    ltGray:      Pattern;   {25% gray pattern}
    dkGray:      Pattern;   {75% gray pattern}
    arrow:       Cursor;    {standard arrow cursor}
    screenBits:  BitMap;    {the entire screen}
    randSeed:    LONGINT;   {determines where Random sequence begins}
```

Routines

GrafPort Routines

```
PROCEDURE InitGraf     (globalPtr: Ptr);
PROCEDURE OpenPort     (port: GrafPtr);
PROCEDURE InitPort     (port: GrafPtr);
PROCEDURE ClosePort    (port: GrafPtr);
PROCEDURE SetPort      (port: GrafPtr);
PROCEDURE GetPort      (VAR port: GrafPtr);
PROCEDURE GrafDevice   (device: INTEGER);
PROCEDURE SetPortBits  (bm: BitMap);
PROCEDURE PortSize     (width,height: INTEGER);
PROCEDURE MovePortTo   (leftGlobal,topGlobal: INTEGER);
PROCEDURE SetOrigin    (h,v: INTEGER);
PROCEDURE SetClip      (rgn: RgnHandle);
PROCEDURE GetClip      (rgn: RgnHandle);
PROCEDURE ClipRect     (r: Rect);
PROCEDURE BackPat      (pat: Pattern);
```

Cursor Handling

```
PROCEDURE InitCursor;
PROCEDURE SetCursor (crsr: Cursor);
PROCEDURE HideCursor;
PROCEDURE ShowCursor;
PROCEDURE ObscureCursor;
```

Pen and Line Drawing

```
PROCEDURE HidePen;
PROCEDURE ShowPen;
PROCEDURE GetPen       (VAR pt: Point);
PROCEDURE GetPenState  (VAR pnState: PenState);
PROCEDURE SetPenState  (pnState: PenState);
PROCEDURE PenSize      (width,height: INTEGER);
PROCEDURE PenMode      (mode: INTEGER);
PROCEDURE PenPat       (pat: Pattern);
PROCEDURE PenNormal;
PROCEDURE MoveTo       (h,v: INTEGER);
PROCEDURE Move         (dh,dv: INTEGER);
PROCEDURE LineTo       (h,v: INTEGER);
PROCEDURE Line         (dh,dv: INTEGER);
```

Text Drawing

```
PROCEDURE TextFont     (font: INTEGER);
PROCEDURE TextFace     (face: Style);
PROCEDURE TextMode     (mode: INTEGER);
PROCEDURE TextSize     (size: INTEGER);
PROCEDURE SpaceExtra   (extra: Fixed);
PROCEDURE DrawChar     (ch: CHAR);
PROCEDURE DrawString   (s: Str255);
PROCEDURE DrawText     (textBuf: Ptr; firstByte,byteCount: INTEGER);
FUNCTION  CharWidth    (ch: CHAR) : INTEGER;
FUNCTION  StringWidth  (s: Str255) : INTEGER;
FUNCTION  TextWidth    (textBuf: Ptr; firstByte,byteCount: INTEGER) :
                        INTEGER;
PROCEDURE GetFontInfo  (VAR info: FontInfo);
```

Drawing in Color

```
PROCEDURE ForeColor  (color: LONGINT);
PROCEDURE BackColor  (color: LONGINT);
PROCEDURE ColorBit   (whichBit: INTEGER);
```

Calculations with Rectangles

```
PROCEDURE SetRect      (VAR r: Rect; left,top,right,bottom: INTEGER);
PROCEDURE OffsetRect   (VAR r: Rect; dh,dv: INTEGER);
PROCEDURE InsetRect    (VAR r: Rect; dh,dv: INTEGER);
FUNCTION  SectRect     (src1,src2: Rect; VAR dstRect: Rect) : BOOLEAN;
PROCEDURE UnionRect    (src1,src2: Rect; VAR dstRect: Rect);
FUNCTION  PtInRect     (pt: Point; r: Rect) : BOOLEAN;
PROCEDURE Pt2Rect      (pt1,pt2: Point; VAR dstRect: Rect);
PROCEDURE PtToAngle    (r: Rect; pt: Point; VAR angle: INTEGER);
FUNCTION  EqualRect    (rect1,rect2: Rect) : BOOLEAN;
FUNCTION  EmptyRect    (r: Rect) : BOOLEAN;
```

Graphic Operations on Rectangles

```
PROCEDURE FrameRect    (r: Rect);
PROCEDURE PaintRect    (r: Rect);
PROCEDURE EraseRect    (r: Rect);
PROCEDURE InvertRect   (r: Rect);
PROCEDURE FillRect     (r: Rect; pat: Pattern);
```

Graphic Operations on Ovals

```
PROCEDURE FrameOval    (r: Rect);
PROCEDURE PaintOval    (r: Rect);
PROCEDURE EraseOval    (r: Rect);
PROCEDURE InvertOval   (r: Rect);
PROCEDURE FillOval     (r: Rect; pat: Pattern);
```

Graphic Operations on Rounded-Corner Rectangles

```
PROCEDURE FrameRoundRect   (r: Rect; ovalWidth,ovalHeight: INTEGER);
PROCEDURE PaintRoundRect   (r: Rect; ovalWidth,ovalHeight: INTEGER);
PROCEDURE EraseRoundRect   (r: Rect; ovalWidth,ovalHeight: INTEGER);
PROCEDURE InvertRoundRect  (r: Rect; ovalWidth,ovalHeight: INTEGER);
PROCEDURE FillRoundRect    (r: Rect; ovalWidth,ovalHeight: INTEGER; pat:
                            Pattern);
```

Graphic Operations on Arcs and Wedges

```
PROCEDURE FrameArc   (r: Rect; startAngle,arcAngle: INTEGER);
PROCEDURE PaintArc   (r: Rect; startAngle,arcAngle: INTEGER);
PROCEDURE EraseArc   (r: Rect; startAngle,arcAngle: INTEGER);
PROCEDURE InvertArc  (r: Rect; startAngle,arcAngle: INTEGER);
PROCEDURE FillArc    (r: Rect; startAngle,arcAngle: INTEGER; pat:
                     Pattern);
```

Calculations with Regions

```
FUNCTION  NewRgn :     RgnHandle;
PROCEDURE OpenRgn;
PROCEDURE CloseRgn     (dstRgn: RgnHandle);
PROCEDURE DisposeRgn   (rgn: RgnHandle);
PROCEDURE CopyRgn      (srcRgn,dstRgn: RgnHandle);
PROCEDURE SetEmptyRgn  (rgn: RgnHandle);
PROCEDURE SetRectRgn   (rgn: RgnHandle; left,top,right,bottom: INTEGER);
PROCEDURE RectRgn      (rgn: RgnHandle; r: Rect);
PROCEDURE OffsetRgn    (rgn: RgnHandle; dh,dv: INTEGER);
PROCEDURE InsetRgn     (rgn: RgnHandle; dh,dv: INTEGER);
PROCEDURE SectRgn      (srcRgnA,srcRgnB,dstRgn: RgnHandle);
PROCEDURE UnionRgn     (srcRgnA,srcRgnB,dstRgn: RgnHandle);
PROCEDURE DiffRgn      (srcRgnA,srcRgnB,dstRgn: RgnHandle);
PROCEDURE XorRgn       (srcRgnA,srcRgnB,dstRgn: RgnHandle);
FUNCTION  PtInRgn      (pt: Point; rgn: RgnHandle) : BOOLEAN;
FUNCTION  RectInRgn    (r: Rect; rgn: RgnHandle) : BOOLEAN;
FUNCTION  EqualRgn     (rgnA,rgnB: RgnHandle) : BOOLEAN;
FUNCTION  EmptyRgn     (rgn: RgnHandle) : BOOLEAN;
```

Graphic Operations on Regions

```
PROCEDURE FrameRgn   (rgn: RgnHandle);
PROCEDURE PaintRgn   (rgn: RgnHandle);
PROCEDURE EraseRgn   (rgn: RgnHandle);
PROCEDURE InvertRgn  (rgn: RgnHandle);
PROCEDURE FillRgn    (rgn: RgnHandle; pat: Pattern);
```

Bit Transfer Operations

```
PROCEDURE ScrollRect  (r: Rect; dh,dv: INTEGER; updateRgn: RgnHandle);
PROCEDURE CopyBits    (srcBits,dstBits: BitMap; srcRect,dstRect: Rect;
                       mode: INTEGER; maskRgn: RgnHandle);
```

Pictures

```
FUNCTION  OpenPicture   (picFrame: Rect) : PicHandle;
PROCEDURE PicComment    (kind,dataSize: INTEGER; dataHandle: Handle);
PROCEDURE ClosePicture;
PROCEDURE DrawPicture   (myPicture: PicHandle; dstRect: Rect);
PROCEDURE KillPicture   (myPicture: PicHandle);
```

Calculations with Polygons

```
FUNCTION  OpenPoly :   PolyHandle;
PROCEDURE ClosePoly;
PROCEDURE KillPoly    (poly: PolyHandle);
PROCEDURE OffsetPoly  (poly: PolyHandle; dh,dv: INTEGER);
```

Graphic Operations on Polygons

```
PROCEDURE FramePoly   (poly: PolyHandle);
PROCEDURE PaintPoly   (poly: PolyHandle);
PROCEDURE ErasePoly   (poly: PolyHandle);
PROCEDURE InvertPoly  (poly: PolyHandle);
PROCEDURE FillPoly    (poly: PolyHandle; pat: Pattern);
```

Calculations with Points

```
PROCEDURE AddPt          (srcPt: Point; VAR dstPt: Point);
PROCEDURE SubPt          (srcPt: Point; VAR dstPt: Point);
PROCEDURE SetPt          (VAR pt: Point; h,v: INTEGER);
FUNCTION  EqualPt        (pt1,pt2: Point) : BOOLEAN;
PROCEDURE LocalToGlobal  (VAR pt: Point);
PROCEDURE GlobalToLocal  (VAR pt: Point);
```

Miscellaneous Routines

```
FUNCTION  Random :   INTEGER;
FUNCTION  GetPixel  (h,v: INTEGER) : BOOLEAN;
PROCEDURE StuffHex  (thingPtr: Ptr; s: Str255);
PROCEDURE ScalePt   (VAR pt: Point; srcRect,dstRect: Rect);
PROCEDURE MapPt     (VAR pt: Point; srcRect,dstRect: Rect);
PROCEDURE MapRect   (VAR r: Rect; srcRect,dstRect: Rect);
PROCEDURE MapRgn    (rgn: RgnHandle; srcRect,dstRect: Rect);
PROCEDURE MapPoly   (poly: PolyHandle; srcRect,dstRect: Rect);
```

Customizing QuickDraw Operations

```
PROCEDURE SetStdProcs   (VAR procs: QDProcs);
PROCEDURE StdText       (byteCount: INTEGER; textBuf: Ptr; numer,denom:
                         Point);
PROCEDURE StdLine       (newPt: Point);
PROCEDURE StdRect       (verb: GrafVerb; r: Rect);
PROCEDURE StdRRect      (verb: GrafVerb; r: Rect; ovalwidth,ovalHeight:
                         INTEGER);
PROCEDURE StdOval       (verb: GrafVerb; r: Rect);
PROCEDURE StdArc        (verb: GrafVerb; r: Rect; startAngle,arcAngle:
                         INTEGER);
PROCEDURE StdPoly       (verb: GrafVerb; poly: PolyHandle);
PROCEDURE StdRgn        (verb: GrafVerb; rgn: RgnHandle);
PROCEDURE StdBits       (VAR srcBits: BitMap; VAR srcRect,dstRect: Rect;
                         mode: INTEGER; maskRgn: RgnHandle);
PROCEDURE StdComment    (kind,dataSize: INTEGER; dataHandle: Handle);
FUNCTION  StdTxMeas     (byteCount: INTEGER; textAddr: Ptr; VAR numer,
                         denom: Point; VAR info: FontInfo) : INTEGER;
PROCEDURE StdGetPic     (dataPtr: Ptr; byteCount: INTEGER);
PROCEDURE StdPutPic     (dataPtr: Ptr; byteCount: INTEGER);
```

Assembly-Language Information

Constants

```
; Size in bytes of QuickDraw global variables

grafSize        .EQU      206

; Source transfer modes

srcCopy         .EQU      0
srcOr           .EQU      1
srcXor          .EQU      2
srcBic          .EQU      3
notSrcCopy      .EQU      4
notSrcOr        .EQU      5
notSrcXor       .EQU      6
notSrcBic       .EQU      7

; Pattern transfer modes

patCopy         .EQU      8
patOr           .EQU      9
patXor          .EQU      10
patBic          .EQU      11
notPatCopy      .EQU      12
notPatOr        .EQU      13
notPatXor       .EQU      14
notPatBic       .EQU      15
```

```
; Standard colors for ForeColor and BackColor

blackColor      .EQU        33
whiteColor      .EQU        30
redColor        .EQU        205
greenColor      .EQU        341
blueColor       .EQU        409
cyanColor       .EQU        273
magentaColor    .EQU        137
yellowColor     .EQU        69

; Standard picture comments

picLParen       .EQU        0
picRParen       .EQU        1

; Character style

boldBit         .EQU        0
italicBit       .EQU        1
ulineBit        .EQU        2
outlineBit      .EQU        3
shadowBit       .EQU        4
condenseBit     .EQU        5
extendBit       .EQU        6

; Graphic operations

frame           .EQU        0
paint           .EQU        1
erase           .EQU        2
invert          .EQU        3
fill            .EQU        4
```

Point Data Structure

v	Vertical coordinate (word)
h	Horizontal coordinate (word)

Rectangle Data Structure

top	Vertical coordinate of top left corner (word)
left	Horizontal coordinate of top left corner (word)
bottom	Vertical coordinate of bottom right corner (word)
right	Horizontal coordinate of bottom right corner (word)
topLeft	Top left corner (point; long)
botRight	Bottom right corner (point; long)

Region Data Structure

rgnSize	Size in bytes (word)
rgnBBox	Enclosing rectangle (8 bytes)
rgnData	More data if not rectangular

Bit Map Data Structure

baseAddr	Pointer to bit image
rowBytes	Row width (word)
bounds	Boundary rectangle (8 bytes)
bitMapRec	Size in bytes of bit map data structure

Cursor Data Structure

data	Cursor image (32 bytes)
mask	Cursor mask (32 bytes)
hotSpot	Point aligned with mouse (long)
cursRec	Size in bytes of cursor data structure

Structure of QDProcs Record

textProc	Address of text-drawing routine
lineProc	Address of line-drawing routine
rectProc	Address of rectangle-drawing routine
rRectProc	Address of roundRect-drawing routine
ovalProc	Address of oval-drawing routine
arcProc	Address of arc/wedge-drawing routine
polyProc	Address of polygon-drawing routine
rgnProc	Address of region-drawing routine
bitsProc	Address of bit-transfer routine
commentProc	Address of routine for processing picture comments
txMeasProc	Address of routine for measuring text width
getPicProc	Address of picture-retrieval routine
putPicProc	Address of picture-saving routine
qdProcsRec	Size in bytes of QDProcs record

GrafPort Data Structure

device	Font-specific information (word)
portBits	GrafPort's bit map (bitMapRec bytes)
portBounds	Boundary rectangle of grafPort's bit map (8 bytes)
portRect	GrafPort's rectangle (8 bytes)
visRgn	Handle to visible region
clipRgn	Handle to clipping region
bkPat	Background pattern (8 bytes)
fillPat	Fill pattern (8 bytes)
pnLoc	Pen location (point; long)

pnSize	Pen size (point; long)
pnMode	Pen's transfer mode (word)
pnPat	Pen pattern (8 bytes)
pnVis	Pen visibility (word)
txFont	Font number for text (word)
txFace	Text's character style (word)
txMode	Text's transfer mode (word)
txSize	Font size for text (word)
spExtra	Extra space (long)
fgColor	Foreground color (long)
bkColor	Background color (long)
colrBit	Color bit (word)
picSave	Handle to picture being saved
rgnSave	Handle to region being saved
polySave	Handle to polygon being saved
grafProcs	Pointer to QDProcs record
portRec	Size in bytes of grafPort

Picture Data Structure

picSize	Size in bytes (word)
picFrame	Picture frame (rectangle; 8 bytes)
picData	Picture definition data

Polygon Data Structure

polySize	Size in bytes (word)
polyBBox	Enclosing rectangle (8 bytes)
polyPoints	Polygon points

Pen State Data Structure

psLoc	Pen location (point; long)
psSize	Pen size (point; long)
psMode	Pen's transfer mode (word)
psPat	Pen pattern (8 bytes)
psRec	Size in bytes of pen state data structure

Font Information Data Structure

ascent	Ascent (word)
descent	Descent (word)
widMax	Maximum character width (word)
leading	Leading (word)

Special Macro Names

Pascal name	Macro name
SetPortBits	_SetPBits
InvertRect	_InverRect
InvertRoundRect	_InverRoundRect
DisposeRgn	_DisposRgn
SetRectRgn	_SetRecRgn
OffsetRgn	_OfSetRgn
InvertRgn	_InverRgn
ClosePoly	_ClosePgon

Variables

RndSeed	Random number seed (long)

RESOURCE MANAGER

Constants

```
CONST  { Masks for resource attributes }

       resSysHeap    = 64;     {set if read into system heap}
       resPurgeable  = 32;     {set if purgeable}
       resLocked     = 16;     {set if locked}
       resProtected  = 8;      {set if protected}
       resPreload    = 4;      {set if to be preloaded}
       resChanged    = 2;      {set if to be written to resource file}

       { Resource Manager result codes }

       resNotFound   = -192;     {resource not found}
       resFNotFound  = -193;     {resource file not found}
       addResFailed  = -194;     {AddResource failed}
       rmvResFailed  = -196;     {RmveResource failed}

       { Masks for resource file attributes }

       mapReadOnly = 128;     {set if file is read-only}
       mapCompact  = 64;      {set to compact file on update}
       mapChanged  = 32;      {set to write map on update}
```

Data Types

```
TYPE ResType = PACKED ARRAY[1..4] OF CHAR;
```

Routines

Initialization

```
FUNCTION  InitResources : INTEGER;
PROCEDURE RsrcZoneInit;
```

Opening and Closing Resource Files

```
PROCEDURE CreateResFile (fileName: Str255);
FUNCTION  OpenResFile   (fileName: Str255) : INTEGER;
PROCEDURE CloseResFile  (refNum: INTEGER);
```

Checking for Errors

```
FUNCTION ResError : INTEGER;
```

Setting the Current Resource File

```
FUNCTION  CurResFile : INTEGER;
FUNCTION  HomeResFile (theResource: Handle) : INTEGER;
PROCEDURE UseResFile  (refNum: INTEGER);
```

Getting Resource Types

```
FUNCTION  CountTypes : INTEGER;
PROCEDURE GetIndType  (VAR theType: ResType; index: INTEGER);
```

Getting and Disposing of Resources

```
PROCEDURE SetResLoad       (load: BOOLEAN);
FUNCTION  CountResources   (theType: ResType) : INTEGER;
FUNCTION  GetIndResource   (theType: ResType; index: INTEGER) : Handle;
FUNCTION  GetResource      (theType: ResType; theID: INTEGER) : Handle;
FUNCTION  GetNamedResource (theType: ResType; name: Str255) : Handle;
PROCEDURE LoadResource     (theResource: Handle);
PROCEDURE ReleaseResource  (theResource: Handle);
PROCEDURE DetachResource   (theResource: Handle);
```

Getting Resource Information

```
FUNCTION  UniqueID      (theType: ResType) : INTEGER;
PROCEDURE GetResInfo    (theResource: Handle; VAR theID: INTEGER; VAR
                         theType: ResType; VAR name: Str255);
FUNCTION  GetResAttrs   (theResource: Handle) : INTEGER;
FUNCTION  SizeResource  (theResource: Handle) : LONGINT;
```

Modifying Resources

```
PROCEDURE SetResInfo       (theResource: Handle; theID: INTEGER; name:
                            Str255);
PROCEDURE SetResAttrs      (theResource: Handle; attrs: INTEGER);
PROCEDURE ChangedResource  (theResource: Handle);
PROCEDURE AddResource      (theData: Handle; theType: ResType; theID:
                            INTEGER; name: Str255);
PROCEDURE RmveResource     (theResource: Handle);
PROCEDURE UpdateResFile    (refNum: INTEGER);
PROCEDURE WriteResource    (theResource: Handle);
PROCEDURE SetResPurge      (install: BOOLEAN);
```

Advanced Routines

```
FUNCTION  GetResFileAttrs (refNum: INTEGER) : INTEGER;
PROCEDURE SetResFileAttrs (refNum: INTEGER; attrs: INTEGER);
```

Assembly-Language Information

Constants

```
; Resource attributes

resSysHeap      .EQU    6     ;set if read into system heap
resPurgeable    .EQU    5     ;set if purgeable
resLocked       .EQU    4     ;set if locked
resProtected    .EQU    3     ;set if protected
resPreload      .EQU    2     ;set if to be preloaded
resChanged      .EQU    1     ;set if to be written to resource file

; Resource Manager result codes

resNotFound     .EQU    -192    ;resource not found
resFNotFound    .EQU    -193    ;resource file not found
addResFailed    .EQU    -194    ;AddResource failed
rmvResFailed    .EQU    -196    ;RmveResource failed

; Resource file attributes

mapReadOnly     .EQU    7     ;set if file is read-only
mapCompact      .EQU    6     ;set to compact file on update
mapChanged      .EQU    5     ;set to write map on update
```

Special Macro Names

Pascal name	Macro name
SizeResource	_SizeRsrc

Variables

TopMapHndl	Handle to resource map of most recently opened resource file
SysMapHndl	Handle to map of system resource file
SysMap	Reference number of system resource file (word)
CurMap	Reference number of current resource file (word)
ResLoad	Current SetResLoad state (word)
ResErr	Current value of ResError (word)
ResErrProc	Address of resource error procedure
SysResName	Name of system resource file (length byte followed by up to 19 characters)

SCRAP MANAGER

Constants

```
CONST { Scrap Manager result codes }

      noScrapErr = -100;  {desk scrap isn't initialized}
      noTypeErr  = -102;  {no data of the requested type}
```

Data Types

```
TYPE PScrapStuff  = ^ScrapStuff;
     ScrapStuff   = RECORD
                        scrapSize:   LONGINT;   {size of desk scrap}
                        scrapHandle: Handle;    {handle to desk scrap}
                        scrapCount:  INTEGER;   {count changed by ZeroScrap}
                        scrapState:  INTEGER;   {tells where desk scrap is}
                        scrapName:   StringPtr  {scrap file name}
                    END;
```

Routines

Getting Desk Scrap Information

```
FUNCTION InfoScrap : PScrapStuff;
```

Keeping the Desk Scrap on the Disk

```
FUNCTION UnloadScrap : LONGINT;
FUNCTION LoadScrap :   LONGINT;
```

Writing to the Desk Scrap

```
FUNCTION ZeroScrap : LONGINT;
FUNCTION PutScrap   (length: LONGINT; theType: ResType; source: Ptr) :
                     LONGINT;
```

Reading from the Desk Scrap

```
FUNCTION GetScrap (hDest: Handle; theType: ResType; VAR offset: LONGINT)
                   : LONGINT;
```

Assembly-Language Information

Constants

```
; Scrap Manager result codes

noScrapErr      .EQU    -100    ;desk scrap isn't initialized
noTypeErr       .EQU    -102    ;no data of the requested type
```

Special Macro Names

Pascal name	**Macro name**
LoadScrap	_LodeScrap
UnloadScrap	_UnlodeScrap

Variables

ScrapSize	Size in bytes of desk scrap (long)
ScrapHandle	Handle to desk scrap in memory
ScrapCount	Count changed by ZeroScrap (word)
ScrapState	Tells where desk scrap is (word)
ScrapName	Pointer to scrap file name (preceded by length byte)

SEGMENT LOADER

Constants

```
CONST { Message returned by CountAppleFiles }

      appOpen  = 0;  {open the document(s)}
      appPrint = 1;  {print the document(s)}
```

Data Types

```
TYPE AppFile =  RECORD
                  vRefNum: INTEGER; {volume reference number}
                  fType:   OSType;  {file type}
                  versNum: INTEGER; {version number}
                  fName:   Str255   {file name}
                END;
```

Routines

```
PROCEDURE CountAppFiles (VAR message: INTEGER; VAR count: INTEGER);   [Not
                         in ROM]
PROCEDURE GetAppFiles   (index: INTEGER; VAR theFile: AppFile);  [Not in ROM]
PROCEDURE ClrAppFiles   (index: INTEGER);  [Not in ROM]
PROCEDURE GetAppParms   (VAR apName: Str255; VAR apRefNum: INTEGER; VAR
                         apParam: Handle);
PROCEDURE UnloadSeg     (routineAddr: Ptr);
PROCEDURE ExitToShell;
```

Assembly-Language Information

Advanced Routines

Trap macro	On entry
_Chain	(A0): pointer to application's file name (preceded by length byte) 4(A0): configuration of sound and screen buffers (word)
_Launch	(A0): pointer to application's file name (preceded by length byte) 4(A0): configuration of sound and screen buffers (word)
_LoadSeg	stack: segment number (word)

Variables

AppParmHandle	Handle to Finder information
CurApName	Name of current application (length byte followed by up to 31 characters)
CurApRefNum	Reference number of current application's resource file (word)
CurPageOption	Sound/screen buffer configuration passed to Chain or Launch (word)
CurJTOffset	Offset to jump table from location pointed to by A5 (word)
FinderName	Name of the Finder (length byte followed by up to 15 characters)

SERIAL DRIVERS

Constants

```
CONST { Driver reset information }

      baud300     =  380;      {300 baud}
      baud600     =  189;      {600 baud}
      baud1200    =  94;       {1200 baud}
      baud1800    =  62;       {1800 baud}
      baud2400    =  46;       {2400 baud}
      baud3600    =  30;       {3600 baud}
      baud4800    =  22;       {4800 baud}
      baud7200    =  14;       {7200 baud}
      baud9600    =  10;       {9600 baud}
      baud19200   =  4;        {19200 baud}
      baud57600   =  0;        {57600 baud}
      stop10      =  16384;    {1 stop bit}
      stop15      = -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}

      { Masks for errors }

      swOverrunErr = 1;   {set if software overrun error}
      parityErr    = 16;  {set if parity error}
      hwOverrunErr = 32;  {set if hardware overrun error}
      framingErr   = 64;  {set if framing error}

      { Masks for changes that cause events to be posted }

      ctsEvent   = 32;    {set if CTS change will cause event to be }
                          { posted}
      breakEvent = 128;   {set if break status change will cause event }
                          { to be posted}

      { Indication that an XOff character was sent }

      xOffWasSent = $80;

      { Result codes }

      noErr   =  0;   {no error}
      openErr = -23;  {attempt to open RAM Serial Driver failed}
```

Data Types

```
TYPE  SPortSel = (sPortA, {modem port}
                  sPortB  {printer port});

      SerShk = PACKED RECORD
                  fXOn: Byte; {XOn/XOff output flow control flag}
                  fCTS: Byte; {CTS hardware handshake flag}
                  xOn:  CHAR; {XOn character}
                  xOff: CHAR; {XOff character}
                  errs: Byte; {errors that cause abort}
                  evts: Byte; {status changes that cause events}
                  fInX: Byte; {XOn/XOff input flow control flag}
                  null: Byte  {not used}
               END;

      SerStaRec =  PACKED RECORD
                     cumErrs:  Byte; {cumulative errors}
                     xOffSent: Byte; {XOff sent as input flow control}
                     rdPend:   Byte; {read pending flag}
                     wrPend:   Byte; {write pending flag}
                     ctsHold:  Byte; {CTS flow control hold flag}
                     xOffHold: Byte  {XOff flow control hold flag}
                   END;
```

Routines [Not in ROM]

Opening and Closing the RAM Serial Driver

```
FUNCTION  RAMSDOpen  (whichPort: SPortSel) : OSErr;
PROCEDURE RAMSDClose (whichPort: SPortSel);
```

Changing Serial Driver Information

```
FUNCTION SerReset  (refNum: INTEGER; serConfig: INTEGER) : OSErr;
FUNCTION SerSetBuf (refNum: INTEGER; serBPtr: Ptr; serBLen: INTEGER) :
                    OSErr;
FUNCTION SerHShake (refNum: INTEGER; flags: SerShk) : OSErr;
FUNCTION SerSetBrk (refNum: INTEGER) : OSErr;
FUNCTION SerClrBrk (refNum: INTEGER) : OSErr;
```

Getting Serial Driver Information

```
FUNCTION SerGetBuf (refNum: INTEGER; VAR count: LONGINT) : OSErr;
FUNCTION SerStatus (refNum: INTEGER; VAR serSta: SerStaRec) : OSErr;
```

Advanced Control Calls (RAM Serial Driver)

csCode	csParam	Effect
13	baudRate	Set baud rate (actual rate, as an integer)
19	char	Replace parity errors
21		Unconditionally set XOff for output flow control
22		Unconditionally clear XOff for input flow control
23		Send XOn for input flow control if XOff was sent last
24		Unconditionally send XOn for input flow control
25		Send XOff for input flow control if XOn was sent last
26		Unconditionally send XOff for input flow control
27		Reset SCC channel

Driver Names and Reference Numbers

Driver	Driver name	Reference number
Modem port input	.AIn	–6
Modem port output	.AOut	–7
Printer port input	.BIn	–8
Printer port output	.BOut	–9

Assembly-Language Information

Constants

```
; Result codes

noErr      .EQU    0    ;no error
openErr    .EQU  -23    ;attempt to open RAM Serial Driver failed
```

Structure of Control Information for SerHShake

shFXOn	XOn/XOff output flow control flag (byte)
shFCTS	CTS hardware handshake flag (byte)
shXOn	XOn character (byte)
shXOff	XOff character (byte)
shErrs	Errors that cause abort (byte)
shEvts	Status changes that cause events (byte)
shFInX	XOn/XOff input flow control flag (byte)

Structure of Status Information for SerStatus

ssCumErrs	Cumulative errors (byte)
ssXOffSent	XOff sent as input flow control (byte)
ssRdPend	Read pending flag (byte)
ssWrPend	Write pending flag (byte)
ssCTSHold	CTS flow control hold flag (byte)
ssXOffHold	XOff flow control hold flag (byte)

Equivalent Device Manager Calls

Pascal routine	**Call**
SerReset	Control with csCode=8, csParam=serConfig
SerSetBuf	Control with csCode=8, csParam=serBPtr, csParam+4=serBLen
SerHShake	Control with csCode=10, csParam through csParam+6=flags
SerSetBrk	Control with csCode=12
SerClrBrk	Control with csCode=11
SerGetBuf	Status with csCode=2; count returned in csParam
SerStatus	Status with csCode=8; serSta returned in csParam through csParam+5

SOUND DRIVER

Constants

```
CONST { Mode values for synthesizers }

   swMode = -1;   {square-wave synthesizer}
   ftMode =  1;   {four-tone synthesizer}
   ffMode =  0;   {free-form synthesizer}
```

Data Types

```
TYPE  { Free-form synthesizer }

      FFSynthPtr =  ^FFSynthRec;
      FFSynthRec =  RECORD
                      mode:       INTEGER;   {always ffMode}
                      count:      Fixed;     {"sampling" factor}
                      waveBytes:  FreeWave   {waveform description}
                    END;

      FreeWave    = PACKED ARRAY[0..30000] OF Byte;

      { Square-wave synthesizer }

      SWSynthPtr = ^SWSynthRec;
      SWSynthRec = RECORD
                     mode:       INTEGER;  {always swMode}
                     triplets:   Tones     {sounds}
                   END;

      Tones = ARRAY[0..5000] OF Tone;
      Tone  = RECORD
                count:      INTEGER;  {frequency}
                amplitude:  INTEGER;  {amplitude, 0-255}
                duration:   INTEGER   {duration in ticks}
              END;

      { Four-tone synthesizer }

      FTSynthPtr   =  ^FTSynthRec;
      FTSynthRec   =  RECORD
                        mode:   INTEGER;       {always ftMode}
                        sndRec: FTSndRecPtr    {tones to play}
                      END;
```

```
     FTSndRecPtr = ^FTSoundRec;
     FTSoundRec  = RECORD
                     duration:    INTEGER;   {duration in ticks}
                     sound1Rate:  Fixed;     {tone 1 cycle rate}
                     sound1Phase: LONGINT;   {tone 1 byte offset}
                     sound2Rate:  Fixed;     {tone 2 cycle rate}
                     sound2Phase: LONGINT;   {tone 2 byte offset}
                     sound3Rate:  Fixed;     {tone 3 cycle rate}
                     sound3Phase: LONGINT;   {tone 3 byte offset}
                     sound4Rate:  Fixed;     {tone 4 cycle rate}
                     sound4Phase: LONGINT;   {tone 4 byte offset}
                     sound1Wave:  WavePtr;   {tone 1 waveform}
                     sound2Wave:  WavePtr;   {tone 2 waveform}
                     sound3Wave:  WavePtr;   {tone 3 waveform}
                     sound4Wave:  WavePtr    {tone 4 waveform}
                   END;

     WavePtr = ^Wave;
     Wave    = PACKED ARRAY[0..255] OF Byte;
```

Routines [Not in ROM]

```
PROCEDURE StartSound  (synthRec: Ptr; numBytes: LONGINT; completionRtn:
                       ProcPtr);
PROCEDURE StopSound;
FUNCTION  SoundDone :  BOOLEAN;
PROCEDURE GetSoundVol (VAR level: INTEGER);
PROCEDURE SetSoundVol (level: INTEGER);
```

Assembly-Language Information

Routines

Pascal name	Equivalent for assembly language
StartSound	Call Write with ioRefNum=–4, ioBuffer=synthRec, ioReqCount=numBytes
StopSound	Call KillIO and (for square-wave) set CurPitch to 0
SoundDone	Poll ioResult field of most recent Write call's parameter block
GetSoundVol	Get low-order three bits of variable SdVolume
SetSoundVol	Call this Pascal procedure from your program

Variables

SdVolume	Speaker volume (byte: low-order three bits only)
SoundPtr	Pointer to four-tone record
SoundLevel	Amplitude in 740-byte buffer (byte)
CurPitch	Value of count in square-wave synthesizer buffer (word)

Sound Driver Values for Notes

The following table contains values for the rate field of a four-tone synthesizer and the count field of a square-wave synthesizer. A just-tempered scale—in the key of C, as an example—is given in the first four columns; you can use a just-tempered scale for perfect tuning in a particular key. The last four columns give an equal-tempered scale, for applications that may use any key; this scale is appropriate for most Macintosh sound applications. Following this table is a list of the ratios used in calculating these values, and instructions on how to calculate them for a just-tempered scale in any key.

	Just-Tempered Scale				Equal-Tempered Scale			
	Rate for Four-Tone		Count for Square-Wave		Rate for Four-Tone		Count for Square-Wave	
Note	**Long**	**Fixed**	**Word**	**Integer**	**Long**	**Fixed**	**Word**	**Integer**
3 octaves below middle C								
C	612B	0.37956	5CBA	23738	604C	0.37616	5D92	23954
C#	667C	0.40033	57EB	22507	6606	0.39853	5851	22609
Db	67A6	0.40488	56EF	22255				
D	6D51	0.42702	526D	21101	6C17	0.42223	535C	21340
Ebb	6E8F	0.43187	5180	20864				
D#	71DF	0.44481	4F21	20257	7284	0.44733	4EAF	20143
Eb	749A	0.45547	4D46	19782				
E	7976	0.47446	4A2F	18991	7953	0.47392	4A44	19012
F	818F	0.50609	458C	17804	808A	0.50211	4619	17945
F#	88A5	0.53377	41F0	16880	882F	0.53197	422A	16938
Gb	8A32	0.53983	4133	16691				
G	91C1	0.56935	3DD1	15825	9048	0.56360	3E73	15987
G#	97D4	0.59308	3B58	15192	98DC	0.59711	3AF2	15090
Ab	9B79	0.60732	39F4	14836				
A	A1F3	0.63261	37A3	14243	A1F3	0.63261	37A3	14243
Bbb	A3CA	0.63980	3703	14083				
A#	AA0C	0.66425	34FD	13565	AB94	0.67023	3484	13444
Bb	ACBF	0.67479	3429	13353				
B	B631	0.71169	3174	12660	B5C8	0.71008	3191	12689
2 octaves below middle C								
C	C257	0.75914	2E5D	11869	C097	0.75230	2EC9	11977
C#	CCF8	0.80066	2BF6	11254	CC0B	0.79704	2C29	11305
Db	CF4C	0.80975	2B77	11127				
D	DAA2	0.85403	2936	10550	D82D	0.84444	29AE	10670
Ebb	DD1D	0.86372	28C0	10432				
D#	E3BE	0.88962	2790	10128	E508	0.89465	2757	10071
Eb	E935	0.91096	26A3	9891				
E	F2ED	0.94893	2517	9495	F2A6	0.94785	2522	9506
F	1031E	1.01218	22C6	8902	10114	1.00421	230C	8972
F#	1114A	1.06754	20F8	8440	1105D	1.06392	2115	8469
Gb	11465	1.07967	2099	8345				
G	12382	1.13870	1EE9	7913	12090	1.12720	1F3A	7994

Note	Long	Fixed	Word	Integer	Long	Fixed	Word	Integer
2 octaves below middle C								
G#	12FA8	1.18616	1DAC	7596	131B8	1.19421	1D79	7545
Ab	136F1	1.21461	1CFA	7418				
A	143E6	1.26523	1BD1	7121	143E6	1.26523	1BD1	7121
Bbb	14794	1.27960	1B81	7041				
A#	15418	1.32849	1A7E	6782	15729	1.34047	1A42	6722
Bb	1597E	1.34958	1A14	6676				
B	16C63	1.42339	18BA	6330	16B90	1.42017	18C8	6344
1 octave below middle C								
C	184AE	1.51828	172F	5935	1812F	1.50462	1764	5988
C#	199EF	1.60130	15FB	5627	19816	1.59409	1614	5652
Db	19E97	1.61949	15BC	5564				
D	1B543	1.70805	149B	5275	1B05A	1.68887	14D7	5335
Ebb	1BA3B	1.72746	1460	5216				
D#	1C77B	1.77922	13C8	5064	1CA10	1.78931	13AC	5036
Eb	1D26A	1.82193	1351	4945				
E	1E5D9	1.89784	128C	4748	1E54D	1.89571	1291	4753
F	2063D	2.02437	1163	4451	20228	2.00842	1186	4486
F#	22294	2.13507	107C	4220	220BB	2.12785	108A	4234
Gb	228C9	2.15932	104D	4173				
G	24704	2.27740	F74	3956	2411F	2.25438	F9D	3997
G#	25F4F	2.37230	ED6	3798	26370	2.38843	EBC	3772
Ab	26DE3	2.42924	E7D	3709				
A	287CC	2.53046	DE9	3561	287CC	2.53046	DE9	3561
Bbb	28F28	2.55920	DC1	3521				
A#	2A830	2.65698	D3F	3391	2AE51	2.68092	D21	3361
Bb	2B2FC	2.69916	D0A	3338				
B	2D8C6	2.84677	C5D	3165	2D721	2.84035	C64	3172
Middle C								
C	3095B	3.03654	B97	2967	3025D	3.00923	BB2	2994
C#	333DE	3.20261	AFD	2813	3302C	3.18817	B0A	2826
Db	33D2E	3.23898	ADE	2782				
D	36A87	3.41612	A4E	2638	360B5	3.37776	A6C	2668
Ebb	37476	3.45493	A30	2608				
D#	38EF7	3.55846	9E4	2532	39420	3.57861	9D6	2518
Eb	3A4D4	3.64386	9A9	2473				
E	3CBB2	3.79568	946	2374	3CA99	3.79140	949	2377
F	40C7A	4.04874	8B1	2225	40450	4.01685	8C3	2243
F#	44528	4.27014	83E	2110	44176	4.25571	845	2117
Gb	45193	4.31865	826	2086				
G	48E09	4.55482	7BA	1978	4823E	4.50876	7CE	1998
G#	4BE9F	4.74461	76B	1899	4C6E1	4.77687	75E	1886
Ab	4DBC5	4.85847	73F	1855				
A	50F98	5.06091	6F4	1780	50F98	5.06091	6F4	1780

Note	Long	Fixed	Word	Integer	Long	Fixed	Word	Integer
Middle C								
Bbb	51E4F	5.11839	6E0	1760				
A#	55060	5.31396	6A0	1696	55CA2	5.36185	690	1680
Bb	565F8	5.39832	685	1669				
B	5B18B	5.69353	62F	1583	5AE41	5.68068	632	1586
1 octave above middle C								
C	612B7	6.07310	5CC	1484	604BB	6.01848	5D9	1497
C#	667BD	6.40523	57F	1407	66059	6.37636	585	1413
Db	67A5C	6.47797	56F	1391				
D	6D50D	6.83223	527	1319	6C169	6.75551	536	1334
Ebb	6E8EB	6.90984	518	1304				
D#	71DEE	7.11691	4F2	1266	7283F	7.15721	4EB	1259
Eb	749A8	7.28772	4D4	1236				
E	79764	7.59137	4A3	1187	79533	7.58281	4A4	1188
F	818F3	8.09746	459	1113	808A1	8.03371	462	1122
F#	88A51	8.54030	41F	1055	882EC	8.51141	423	1059
Gb	8A326	8.63730	413	1043				
G	91C12	9.10965	3DD	989	9047D	9.01753	3E7	999
G#	97D3D	9.48921	3B6	950	98DC2	9.55374	3AF	943
Ab	9B78B	9.71696	39F	927				
A	A1F30	10.12183	37A	890	A1F30	10.12183	37A	890
Bbb	A3C9F	10.23680	370	880				
A#	AA0BF	10.62791	350	848	AB945	10.72371	348	840
Bb	ACBEF	10.79662	343	835				
B	B6316	11.38705	317	791	B5C83	11.36137	319	793
2 octaves above middle C								
C	C256D	12.14619	2E6	742	C0976	12.03696	2ED	749
C#	CCF79	12.81044	2BF	703	CC0B1	12.75270	2C3	707
Db	CF4B9	12.95595	2B7	695				
D	DAA1B	13.66447	293	659	D82D2	13.51102	29B	667
Ebb	DD1D6	13.81967	28C	652				
D#	E3BDC	14.23383	279	633	E507E	14.31442	275	629
Eb	E9350	14.57544	26A	618				
E	F2EC8	15.18274	251	593	F2A65	15.16560	252	594
F	1031E7	16.19493	22C	556	101141	16.06740	231	561
F#	1114A1	17.08058	210	528	1105D8	17.02283	211	529
Gb	11464C	17.27460	20A	522				
G	123824	18.21930	1EF	495	1208F9	18.03505	1F4	500
G#	12FA7B	18.97844	1DB	475	131B83	19.10747	1D8	472
Ab	136F15	19.43391	1D0	464				
A	143E61	20.24367	1BD	445	143E61	20.24367	1BD	445
Bbb	14793D	20.47359	1B8	440				
A#	15417F	21.25584	1A8	424	15728A	21.44742	1A4	420
Bb	1597DE	21.59323	1A1	417				
B	16C62D	22.77412	18C	396	16B906	22.72275	18D	397

Note	Long	Fixed	Word	Integer	Long	Fixed	Word	Integer
3 octaves above middle C								
C	184ADA	24.29239	173	371	1812EB	24.07390	176	374
C#	199EF2	25.62088	160	352	198163	25.50542	161	353
Db	19E971	25.91188	15C	348				
D	1B5436	27.32895	14A	330	1B05A5	27.02205	14D	333
Ebb	1BA3AC	27.63934	146	326				
D#	1C77B8	28.46765	13D	317	1CA0FD	28.62886	13B	315
Eb	1D26A0	29.15088	135	309				
E	1E5D91	30.36549	129	297	1E54CB	30.33122	129	297
F	2063CE	32.38986	116	278	202283	32.13481	118	280
F#	222943	34.16118	108	264	220BAF	34.04564	109	265
Gb	228C97	34.54918	105	261				
G	247047	36.43858	F7	247	2411F2	36.07010	FA	250
G#	25F4F5	37.95686	ED	237	263706	38.21494	EC	236
Ab	26DE2A	38.86783	E8	232				
A	287CC1	40.48732	DF	223	287CC1	40.48732	DF	223
Bbb	28F27A	40.94717	DC	220				
A#	2A82FE	42.51169	D4	212	2AE513	42.89482	D2	210
Bb	2B2FBD	43.18648	D1	209				
B	2D8C59	45.54823	C6	198	2D720B	45.44548	C6	198

The following table gives the ratios used in calculating the above values. It shows the relationship between the notes making up the just-tempered scale in the key of C; should you need to implement a just-tempered scale in some other key, you can do so as follows: First get the value of the root note in the proper octave in the equal-tempered scale (from the above table). Then use the following table to determine the values of the intervals for the other notes in the key by multiplying the ratio by the root note.

Chromatic interval	Note	Just-tempered frequency ratio	Equal-tempered frequency ratio	Interval type
0	C	1.00000	1.00000	Unison
1	C#	1.05469	1.05946	Minor second as chromatic semitone
	Db	1.06667		Minor second as diatonic semitone
2	D	1.11111	1.12246	Major second as minor tone
	D	1.12500		Major second as major tone
	Ebb	1.13778		Diminished third
3	D#	1.17188	1.18921	Augmented second
	Eb	1.20000		Minor third
4	E	1.25000	1.25992	Major third
5	F	1.33333	1.33484	Fourth
6	F#	1.40625	1.41421	Tritone as augmented fourth
	Gb	1.42222		Tritone as diminished fifth
7	G	1.50000	1.49831	Fifth

Chromatic interval	Note	Just-tempered frequency ratio	Equal-tempered frequency ratio	Interval type
8	G#	1.56250	1.58740	Augmented fifth
	Ab	1.60000		Minor sixth
9	A	1.66667	1.68179	Major sixth
	Bbb	1.68560		Diminished seventh
10	A#	1.75000	1.78180	Augmented sixth
	Bb	1.77778		Minor seventh
11	B	1.87500	1.88775	Major seventh
12	C	2.00000	2.00000	Octave

STANDARD FILE PACKAGE

Constants

```
CONST { SFPutFile dialog template ID }

      putDlgID = -3999;

      { Item numbers of enabled items in SFPutFile dialog }

      putSave   = 1;    {Save button}
      putCancel = 2;    {Cancel button}
      putEject  = 5;    {Eject button}
      putDrive  = 6;    {Drive button}
      putName   = 7;    {editText item for file name}

      { SFGetFile dialog template ID }

      getDlgID = -4000;

      { Item numbers of enabled items in SFGetFile dialog }

      getOpen   = 1;    {Open button}
      getCancel = 3;    {Cancel button}
      getEject  = 5;    {Eject button}
      getDrive  = 6;    {Drive button}
      getNmList = 7;    {userItem for file name list}
      getScroll = 8;    {userItem for scroll bar}
```

Data Types

```
TYPE  SFReply = RECORD
                  good:     BOOLEAN;      {FALSE if ignore command}
                  copy:     BOOLEAN;      {not used}
                  fType:    OSType;       {file type or not used}
                  vRefNum:  INTEGER;      {volume reference number}
                  version:  INTEGER;      {file's version number}
                  fName:    STRING[63]    {file name}
                END;

      SFTypeList = ARRAY[0..3] OF OSType;
```

Routines

```
PROCEDURE SFPutFile    (where: Point; prompt: Str255; origName: Str255;
                        dlgHook: ProcPtr; VAR reply: SFReply);
PROCEDURE SFPPutFile   (where: Point; prompt: Str255; origName: Str255;
                        dlgHook: ProcPtr; VAR reply: SFReply; dlgID:
                        INTEGER; filterProc: ProcPtr);
PROCEDURE SFGetFile    (where: Point; prompt: Str255; fileFilter: ProcPtr;
                        numTypes: INTEGER; typeList: SFTypeList; dlgHook:
                        ProcPtr; VAR reply: SFReply);
PROCEDURE SFPGetFile   (where: Point; prompt: Str255; fileFilter: ProcPtr;
                        numTypes: INTEGER; typeList: SFTypeList; dlgHook:
                        ProcPtr; VAR reply: SFReply; dlgID: INTEGER;
                        filterProc: ProcPtr);
```

DlgHook Function

```
FUNCTION MyDlg (item: INTEGER; theDialog: DialogPtr) : INTEGER;
```

FileFilter Function

```
FUNCTION MyFileFilter (paramBlock: ParmBlkPtr) : BOOLEAN;
```

Standard SFPutFile Items

Item number	Item	Standard display rectangle
1	Save button	(12,74)(82,92)
2	Cancel button	(114,74)(184,92)
3	Prompt string (statText)	(12,12)(184,28)
4	UserItem for disk name	(209,16)(295,34)
5	Eject button	(217,43)(287,61)
6	Drive button	(217,74)(287,92)
7	EditText item for file name	(14,34)(182,50)
8	UserItem for dotted line	(200,16)(201,88)

Resource IDs of SFPutFile Alerts

Alert	Resource ID
Disk not found	–3994
System error	–3995
Existing file	–3996
Locked disk	–3997

Standard SFGetFile Items

Item number	Item	Standard display rectangle
1	Open button	(152,28)(232,46)
2	Invisible button	(1152,59)(1232,77)
3	Cancel button	(152,90)(232,108)
4	UserItem for disk name	(248,28)(344,46)
5	Eject button	(256,59)(336,77)
6	Drive button	(256,90)(336,108)
7	UserItem for file name list	(12,11)(125,125)
8	UserItem for scroll bar	(124,11)(140,125)
9	UserItem for dotted line	(244,20)(245,116)
10	Invisible text (statText)	(1044,20)(1145,116)

Assembly-Language Information

Constants

```
; SFPutFile dialog template ID

putDlgID    .EQU   -3999

; Item numbers of enabled items in SFPutFile dialog

putSave     .EQU    1    ;Save button
putCancel   .EQU    2    ;Cancel button
putEject    .EQU    5    ;Eject button
putDrive    .EQU    6    ;Drive button
putName     .EQU    7    ;editText item for file name

; SFGetFile dialog template ID

getDlgID    .EQU   -4000
```

```
; Item numbers of enabled items in SFGetFile dialog

getOpen      .EQU    1     ;Open button
getCancel    .EQU    3     ;Cancel button
getEject     .EQU    5     ;Eject button
getDrive     .EQU    6     ;Drive button
getNmList    .EQU    7     ;userItem for file name list
getScroll    .EQU    8     ;userItem for scroll bar

; Routine selectors

sfPutFile    .EQU    1
sfGetFile    .EQU    2
sfPPutFile   .EQU    3
sfPGetFile   .EQU    4
```

Reply Record Data Structure

rGood	0 if ignore command (byte)
rType	File type (long)
rVolume	Volume reference number (word)
rVersion	File's version number (word)
rName	File name (length byte followed by up to 63 characters)

Trap Macro Name

_Pack3

Variables

SFSaveDisk	Negative of volume reference number used by Standard File Package (word)

SYSTEM ERROR HANDLER

Routines

```
PROCEDURE SysError (errorCode: INTEGER);
```

User Alerts

ID	Explanation
1	Bus error: Invalid memory reference; happens only on a Macintosh XL
2	Address error: Word or long-word reference made to an odd address
3	Illegal instruction: The MC68000 received an instruction it didn't recognize.
4	Zero divide: Signed Divide (DIVS) or Unsigned Divide (DIVU) instruction with a divisor of 0 was executed.
5	Check exception: Check Register Against Bounds (CHK) instruction was executed and failed. Pascal "value out of range" errors are usually reported in this way.
6	TrapV exception: Trap On Overflow (TRAPV) instruction was executed and failed.
7	Privilege violation: Macintosh always runs in supervisor mode; perhaps an erroneous Return From Execution (RTE) instruction was executed.
8	Trace exception: The trace bit in the status register is set.
9	Line 1010 exception: The 1010 trap dispatcher has failed.
10	Line 1111 exception: Unimplemented instruction
11	Miscellaneous exception: All other MC68000 exceptions
12	Unimplemented core routine: An unimplemented trap number was encountered.
13	Spurious interrupt: The interrupt vector table entry for a particular level of interrupt is NIL; usually occurs with level 4, 5, 6, or 7 interrupts.
14	I/O system error: The File Manager is attempting to dequeue an entry from an I/O request queue that has a bad queue type field; perhaps the queue entry is unlocked. Or, the dCtlQHead field was NIL during a Fetch or Stash call. Or, a needed device control entry has been purged.
15	Segment Loader error: A GetResource call to read a segment into memory failed.
16	Floating point error: The halt bit in the floating-point environment word was set.
17-24	Can't load package: A GetResource call to read a package into memory failed.
25	Can't allocate requested memory block in the heap
26	Segment Loader error: A GetResource call to read 'CODE' resource 0 into memory failed; usually indicates a nonexecutable file.

27	File map destroyed: A logical block number was found that's greater than the number of the last logical block on the volume or less than the logical block number of the first allocation block on the volume.
28	Stack overflow error: The stack has expanded into the heap.
30	"Please insert the disk:" File Manager alert
41	The file named "Finder" can't be found on the disk.
100	Can't mount system startup volume. The system couldn't read the system resource file into memory.
32767	"Sorry, a system error occurred": Default alert message

System Startup Alerts

"Welcome to Macintosh"
"Disassembler installed"
"MacsBug installed"
"Warning—this startup disk is not usable"

Assembly-Language Information

Constants

```
; System error IDs

dsBusError       .EQU     1          ;bus error
dsAddressErr     .EQU     2          ;address error
dsIllInstErr     .EQU     3          ;illegal instruction
dsZeroDivErr     .EQU     4          ;zero divide
dsChkErr         .EQU     5          ;check exception
dsOvflowErr      .EQU     6          ;trapV exception
dsPrivErr        .EQU     7          ;privilege violation
dsTraceErr       .EQU     8          ;trace exception
dsLineAErr       .EQU     9          ;line 1010 exception
dsLineFErr       .EQU     10         ;line 1111 exception
dsMiscErr        .EQU     11         ;miscellaneous exception
dsCoreErr        .EQU     12         ;unimplemented core routine
dsIrqErr         .EQU     13         ;spurious interrupt
dsIOCoreErr      .EQU     14         ;I/O system error
dsLoadErr        .EQU     15         ;Segment Loader error
dsFPErr          .EQU     16         ;floating point error
dsNoPackErr      .EQU     17         ;can't load package 0
dsNoPk1          .EQU     18         ;can't load package 1
dsNoPk2          .EQU     19         ;can't load package 2
dsNoPk3          .EQU     20         ;can't load package 3
dsNoPk4          .EQU     21         ;can't load package 4
dsNoPk5          .EQU     22         ;can't load package 5
dsNoPk6          .EQU     23         ;can't load package 6
```

```
dsNoPk7         .EQU    24       ;can't load package 7
dsMemFullErr    .EQU    25       ;can't allocate requested block
dsBadLaunch     .EQU    26       ;Segment Loader error
dsFSErr         .EQU    27       ;file map destroyed
dsStkNHeap      .EQU    28       ;stack overflow error
dsReinsert      .EQU    30       ;"Please insert the disk:"
dsSysErr        .EQU    32767    ;undifferentiated system error
```

Routines

Trap macro	On entry	On exit
_SysError	D0: errorCode (word)	All registers changed

Variables

DSErrCode	Current system error ID (word)
DSAlertTab	Pointer to system error alert table in use
DSAlertRect	Rectangle enclosing system error alert (8 bytes)

TEXTEDIT

Constants

```
CONST { Text justification }

      teJustLeft    =  0;
      teJustCenter  =  1;
      teJustRight   = -1;
```

Data Types

```
TYPE TEHandle = ^TEPtr;
     TEPtr    = ^TERec;
     TERec = RECORD
                destRect:    Rect;      {destination rectangle}
                viewRect:    Rect;      {view rectangle}
                selRect:     Rect;      {used from assembly language}
                lineHeight   INTEGER;   {for line spacing
                fontAscent:  INTEGER;   {caret/highlighting position}
                selPoint:    Point;     {used from assembly language}
                selStart:    INTEGER;   {start of selection range}
                selEnd:      INTEGER;   {end of selection range
                active:      INTEGER;   {used internally}
                wordBreak:   ProcPtr;   {for word break routine}
                clikLoop:    ProcPtr;   {for click loop routine}
                clickTime:   LONGINT;   {used internally}
                clickLoc:    INTEGER;   {used internally}
                caretTime:   LONGINT;   {used internally}
                caretState:  INTEGER;   {used internally}
                just:        INTEGER;   {justification of text}
                teLength:    INTEGER;   {length of text}
                hText:       Handle;    {text to be edited}
                recalBack:   INTEGER;   {used internally}
                recalLines:  INTEGER;   {used internally}
                clikStuff:   INTEGER;   {used internally}
                crOnly:      INTEGER;   {if <0, new line at Return only}
                txFont:      INTEGER;   {text font}
                txFace:      Style;     {character style}
                txMode:      INTEGER;   {pen mode}
                txSize:      INTEGER;   {font size}
                inPort:      GrafPtr;   {grafPort}
                highHook:    ProcPtr;   {used from assembly language}
                caretHook:   ProcPtr;   {used from assembly language}
                nLines:      INTEGER;   {number of lines}
                lineStarts:  ARRAY[0..16000] OF INTEGER
                                        {positions of line starts}
             END;
     CharsHandle = ^CharsPtr;
     CharsPtr    = ^Chars;
     Chars       = PACKED ARRAY[0..32000] OF CHAR;
```

Routines

Initialization and Allocation

```
PROCEDURE TEInit;
FUNCTION  TENew     (destRect,viewRect: Rect) : TEHandle;
PROCEDURE TEDispose (hTE: TEHandle);
```

Accessing the Text of an Edit Record

```
PROCEDURE TESetText (text: Ptr; length: LONGINT; hTE: TEHandle);
FUNCTION  TEGetText (hTE: TEHandle) : CharsHandle;
```

Insertion Point and Selection Range

```
PROCEDURE TEIdle        (hTE: TEHandle);
PROCEDURE TEClick       (pt: Point; extend: BOOLEAN; hTE: TEHandle);
PROCEDURE TESetSelect   (selStart,selEnd: LONGINT; hTE: TEHandle);
PROCEDURE TEActivate    (hTE: TEHandle);
PROCEDURE TEDeactivate  (hTE: TEHandle);
```

Editing

```
PROCEDURE TEKey     (key: CHAR; hTE: TEHandle);
PROCEDURE TECut     (hTE: TEHandle);
PROCEDURE TECopy    (hTE: TEHandle);
PROCEDURE TEPaste   (hTE: TEHandle);
PROCEDURE TEDelete  (hTE: TEHandle);
PROCEDURE TEInsert  (text: Ptr; length: LONGINT; hTE: TEHandle);
```

Text Display and Scrolling

```
PROCEDURE TESetJust (just: INTEGER; hTE: TEHandle);
PROCEDURE TEUpdate  (rUpdate: Rect; hTE: TEHandle);
PROCEDURE TextBox   (text: Ptr; length: LONGINT; box: Rect; just:
                     INTEGER);
PROCEDURE TEScroll  (dh,dv: INTEGER; hTE: TEHandle);
```

Scrap Handling [Not in ROM]

```
FUNCTION  TEFromScrap  :   OSErr;
FUNCTION  TEToScrap  :     OSErr;
FUNCTION  TEScrapHandle :  Handle;
FUNCTION  TEGetScrapLen :  LONGINT;
PROCEDURE TESetScrapLen : (length: LONGINT);
```

Advanced Routines

```
PROCEDURE SetWordBreak (wBrkProc: ProcPtr; hTE: TEHandle);   [Not in ROM]
PROCEDURE SetClikLoop  (clikProc: ProcPtr; hTE: TEHandle);   [Not in ROM]
PROCEDURE TECalText    (hTE: TEHandle);
```

Word Break Routine

```
FUNCTION MyWordBreak (text: Ptr; charPos: INTEGER) : BOOLEAN;
```

Click Loop Routine

```
FUNCTION MyClikLoop : BOOLEAN;
```

Assembly-Language Information

Constants

```
; Text justification

teJustLeft      .EQU    0
teJustCenter    .EQU    1
teJustRight     .EQU   -1
```

Edit Record Data Structure

teDestRect	Destination rectangle (8 bytes)
teViewRect	View rectangle (8 bytes)
teSelRect	Selection rectangle (8 bytes)
teLineHite	For line spacing (word)
teAscent	Caret/highlighting position (word)
teSelPoint	Point selected with mouse (long)
teSelStart	Start of selection range (word)
teSelEnd	End of selection range (word)
teWordBreak	Address of word break routine (see below)
teClikProc	Address of click loop routine (see below)
teJust	Justification of text (word)

teLength	Length of text (word)
teTextH	Handle to text
teCROnly	If <0, new line at Return only (byte)
teFont	Text font (word)
teFace	Character style (word)
teMode	Pen mode (word)
teSize	Font size (word)
teGrafPort	Pointer to grafPort
teHiHook	Address of text highlighting routine (see below)
teCarHook	Address of routine to draw caret (see below)
teNLines	Number of lines (word)
teLines	Positions of line starts (teNLines*2 bytes)
teRecSize	Size in bytes of edit record except teLines field

Word break routine

On entry	A0:	pointer to text
	D0:	character position (word)
On exit	Z condition code:	0 to break at specified character
		1 not to break there

Click loop routine

On exit	D0:	1
	D2:	must be preserved

Text highlighting routine

On entry	A3:	pointer to locked edit record

Caret drawing routine

On entry	A3:	pointer to locked edit record

Variables

TEScrpHandle	Handle to TextEdit scrap
TEScrpLength	Size in bytes of TextEdit scrap (word)
TERecal	Address of routine to recalculate line starts (see below)
TEDoText	Address of multi-purpose routine (see below)

TERecal routine

On entry	A3:	pointer to locked edit record
	D7:	change in length of edit record (word)
On exit	D2:	line start of line containing first character to be redrawn (word)
	D3:	position of first character to be redrawn (word)
	D4:	position of last character to be redrawn (word)

TEDoText routine

On entry	A3:	pointer to locked edit record
	D3:	position of first character to be redrawn (word)
	D4:	position of last character to be redrawn (word)
	D7:	(word) 0 to hit-test a character
		1 to highlight selection range
		–1 to display text
		–2 to position pen to draw caret
On exit	A0:	pointer to current grafPort
	D0:	if hit-testing, character position or –1 for none (word)

UTILITIES, OPERATING SYSTEM

Constants

```
CONST { Values returned by Environs procedure }

      macXLMachine  = 0;      {Macintosh XL}
      macMachine    = 1;      {Macintosh 128K or 512K}

      { Result codes }

      clkRdErr      = -85;      {unable to read clock}
      clkWrErr      = -86;      {time written did not verify}
      memFullErr    = -108;     {not enough room in heap zone}
      memWZErr      = -111;     {attempt to operate on a free block}
      nilHandleErr  = -109;     {NIL master pointer}
      noErr         =  0;       {no error}
      prInitErr     = -88;      {validity status is not $A8}
      prWrErr       = -87;      {parameter RAM written did not verify}
      qErr          = -1;       {entry not in specified queue}
```

Data Types

```
TYPE OSType = PACKED ARRAY[1..4] OF CHAR;

     OSErr = INTEGER;

     SysPPtr     = ^SysParmType;
     SysParmType =
       RECORD
         valid:     Byte;     {validity status}
         aTalkA:    Byte;     {AppleTalk node ID hint for modem port}
         aTalkB:    Byte;     {AppleTalk node ID hint for printer port}
         config:    Byte;     {use types for serial ports}
         portA:     INTEGER;  {modem port configuration}
         portB:     INTEGER;  {printer port configuration}
         alarm:     LONGINT;  {alarm setting}
         font:      INTEGER;  {application font number minus 1}
         kbdPrint:  INTEGER;  {auto-key settings, printer connection}
         volClik:   INTEGER;  {speaker volume, double-click, caret blink}
         misc:      INTEGER   {mouse scaling, startup disk, menu blink}
       END;

     QHdrPtr  = ^QHdr;
     QHdr     = RECORD
                  qFlags:  INTEGER;      {queue flags}
                  qHead:   QElemPtr;     {first queue entry}
                  qTail:   QElemPtr      {last queue entry}
                END;
```

```
QTypes = (dummyType,
          vType,          {vertical retrace queue type}
          ioQType,        {file I/O or driver I/O queue type}
          drvQType,       {drive queue type}
          evType,         {event queue type}
          fsQType);       {volume-control-block queue type}

QElemPtr = ^QElem;
QElem    = RECORD
             CASE QTypes OF
               vType:    (vblQElem: VBLTask);
               ioQType:  (ioQElem:  ParamBlockRec);
               drvQType: (drvQElem: DrvQEl);
               evType:   (evQElem:  EvQEl);
               fsQType:  (vcbQElem: VCB)
           END;

DateTimeRec =
            RECORD
              year:      INTEGER;  {1904 to 2040}
              month:     INTEGER;  {1 to 12 for January to December}
              day:       INTEGER;  {1 to 31}
              hour:      INTEGER;  {0 to 23}
              minute:    INTEGER;  {0 to 59}
              second:    INTEGER;  {0 to 59}
              dayOfWeek: INTEGER   {1 to 7 for Sunday to Saturday}
            END;
```

Routines

Pointer and Handle Manipulation

```
FUNCTION HandToHand  (VAR theHndl: Handle) : OSErr;
FUNCTION PtrToHand   (srcPtr: Ptr; VAR dstHndl: Handle; size: LONGINT) :
                      OSErr;
FUNCTION PtrToXHand  (srcPtr: Ptr; dstHndl: Handle; size: LONGINT) :
                      OSErr;
FUNCTION HandAndHand (aHndl,bHndl: Handle) : OSErr;
FUNCTION PtrAndHand  (pntr: Ptr; hndl: Handle; size: LONGINT) : OSErr;
```

String Comparison

```
FUNCTION  EqualString (aStr,bStr: Str255; caseSens,diacSens: BOOLEAN) :
                       BOOLEAN;
PROCEDURE UprString   (VAR theString: Str255; diacSens: BOOLEAN);
```

Date and Time Operations

```
FUNCTION  ReadDateTime   (VAR secs: LONGINT) : OSErr;
PROCEDURE GetDateTime    (VAR secs: LONGINT);  [Not in ROM]
FUNCTION  SetDateTime    (secs: LONGINT) : OSErr;
PROCEDURE Date2Secs      (date: DateTimeRec; VAR secs: LONGINT);
PROCEDURE Secs2Date      (secs: LONGINT; VAR date: DateTimeRec);
PROCEDURE GetTime        (VAR date: DateTimeRec);  [Not in ROM]
PROCEDURE SetTime        (date: DateTimeRec);  [Not in ROM]
```

Parameter RAM Operations

```
FUNCTION  InitUtil :    OSErr;
FUNCTION  GetSysPPtr :  SysPPtr;  [Not in ROM]
FUNCTION  WriteParam :  OSErr;
```

Queue Manipulation

```
PROCEDURE Enqueue (qEntry: QElemPtr; theQueue: QHdrPtr);
FUNCTION  Dequeue (qEntry: QElemPtr; theQueue: QHdrPtr) : OSErr;
```

Trap Dispatch Table Utilities

```
PROCEDURE SetTrapAddress (trapAddr: LONGINT; trapNum: INTEGER);
FUNCTION  GetTrapAddress (trapNum: INTEGER) : LONGINT;
```

Miscellaneous Utilities

```
PROCEDURE Delay        (numTicks: LONGINT; VAR finalTicks: LONGINT);
PROCEDURE SysBeep      (duration: INTEGER);
PROCEDURE Environs     (VAR rom,machine: INTEGER);  [Not in ROM]
PROCEDURE Restart;      [Not in ROM]
PROCEDURE SetUpA5;      [Not in ROM]
PROCEDURE RestoreA5;    [Not in ROM]
```

Default Parameter RAM Values

Parameter	Default value
Validity status	$A8
Node ID hint for modem port	0
Node ID hint for printer port	0
Use types for serial ports	0 (both ports)
Modem port configuration	9600 baud, 8 data bits, 2 stop bits, no parity

Parameter	Default value
Printer port configuration	Same as for modem port
Alarm setting	0 (midnight, January 1, 1904)
Application font number minus 1	2 (Geneva)
Auto-key threshold	6 (24 ticks)
Auto-key rate	3 (6 ticks)
Printer connection	0 (printer port)
Speaker volume	3 (medium)
Double-click time	8 (32 ticks)
Caret-blink time	8 (32 ticks)
Mouse scaling	1 (on)
Preferred system startup disk	0 (internal drive)
Menu blink	3

Assembly-Language Information

Constants

```
; Result codes

clkRdErr       .EQU    -85     ;unable to read clock
clkWrErr       .EQU    -86     ;time written did not verify
memFullErr     .EQU    -108    ;not enough room in heap zone
memWZErr       .EQU    -111    ;attempt to operate on a free block
nilHandleErr   .EQU    -109    ;NIL master pointer
noErr          .EQU     0      ;no error
prInitErr      .EQU    -88     ;validity status is not $A8
prWrErr        .EQU    -87     ;parameter RAM written did not verify
qErr           .EQU    -1      ;entry not in specified queue

; Queue types

vType          .EQU     1     ;vertical retrace queue type
ioQType        .EQU     2     ;file I/O or driver I/O queue type
drvQType       .EQU     3     ;drive queue type
evType         .EQU     4     ;event queue type
fsQType        .EQU     5     ;volume-control-block queue type
```

Queue Data Structure

qFlags	Queue flags (word)
qHead	Pointer to first queue entry
qTail	Pointer to last queue entry

Date/Time Record Data Structure

dtYear	1904 to 2040 (word)
dtMonth	1 to 12 for January to December (word)
dtDay	1 to 31 (word)
dtHour	0 to 23 (word)
dtMinute	0 to 59 (word)
dtSecond	0 to 59 (word)
dtDayOfWeek	1 to 7 for Sunday to Saturday (word)

Routines

Trap macro	On entry	On exit
_HandToHand	A0: theHndl (handle)	A0: theHndl (handle) D0: result code(word)
_PtrToHand	A0: srcPtr (ptr) D0: size (long)	A0: dstHndl (handle) D0: result code (word)
_PtrToXHand	A0: srcPtr (ptr) A1: dstHndl (handle) D0: size (long)	A0: dstHndl (handle) D0: result code (word)
_HandAndHand	A0: aHndl (handle) A1: bHndl (handle)	A0: bHndl (handle) D0: result code (word)
_PtrAndHand	A0: pntr (ptr) A1: hndl (handle) D0: size (long)	A0: hndl (handle) D0: result code (word)
_CmpString	_CmpString ,MARKS sets bit 9, for diacSens=FALSE _CmpString ,CASE sets bit 10, for caseSens=TRUE _CmpString ,MARKS,CASE sets bits 9 and 10 A0: ptr to first string A1: ptr to second string D0: high word: length of first string low word: length of second string	D0: 0 if equal, 1 if not equal (long)
_UprString	_UprString ,MARKS sets bit 9, for diacSens=FALSE A0: ptr to string D0: length of string (word)	A0: ptr to string
_ReadDateTime	A0: ptr to long word secs	A0: ptr to long word secs D0: result code (word)
_SetDateTime	D0: secs (long)	D0: result code (word)
_Date2Secs	A0: ptr to date/time record	D0: secs (long)
_Secs2Date	D0: secs (long)	A0: ptr to date/time record
_InitUtil		D0: result code (word)
_WriteParam	A0: SysParam (ptr) D0: MinusOne (long)	D0: result code (word)

Trap macro	On entry	On exit
_Enqueue	A0: qEntry (ptr) A1: theQueue (ptr)	A1: theQueue (ptr)
_Dequeue	A0: qEntry (ptr) A1: theQueue (ptr)	A1: theQueue (ptr) D0: result code (word)
_GetTrapAddress	D0: trapNum (word)	A0: address of routine
_SetTrapAddress	A0: trapAddr (address) D0: trapNum (word)	
_Delay	A0: numTicks (long)	D0: finalTicks (long)
_SysBeep	stack: duration (word)	

Variables

SysParam	Low-memory copy of parameter RAM (20 bytes)
SPValid	Validity status (byte)
SPATalkA	AppleTalk node ID hint for modem port (byte)
SPATalkB	AppleTalk node ID hint for printer port (byte)
SPConfig	Use types for serial ports (byte)
SPPortA	Modem port configuration (word)
SPPortB	Printer port configuration (word)
SPAlarm	Alarm setting (long)
SPFont	Application font number minus 1 (word)
SPKbd	Auto-key threshold and rate (byte)
SPPrint	Printer connection (byte)
SPVolCtl	Speaker volume (byte)
SPClikCaret	Double-click and caret-blink times (byte)
SPMisc2	Mouse scaling, system startup disk, menu blink (byte)
CrsrThresh	Mouse-scaling threshold (word)
Time	Seconds since midnight, January 1, 1904 (long)

UTILITIES, TOOLBOX

Constants

```
CONST { Resource ID of standard pattern list }

      sysPatListID = 0;

      { Resource IDs of standard cursors }

      iBeamCursor = 1;  {to select text}
      crossCursor = 2;  {to draw graphics}
      plusCursor  = 3;  {to select cells in structured documents}
      watchCursor = 4;  {to indicate a long wait}
```

Data Types

```
TYPE Int64Bit =  RECORD
                   hiLong: LONGINT;
                   loLong: LONGINT
                 END;

     CursPtr    = ^Cursor;
     CursHandle = ^CursPtr;

     PatPtr    = ^Pattern;
     PatHandle = ^PatPtr;
```

Routines

Fixed-Point Arithmetic

```
FUNCTION FixRatio (numer,denom: INTEGER) : Fixed;
FUNCTION FixMul   (a,b: Fixed) : Fixed;
FUNCTION FixRound (x: Fixed) : INTEGER;
```

String Manipulation

```
FUNCTION  NewString     (theString: Str255) : StringHandle;
PROCEDURE SetString     (h: StringHandle; theString: Str255);
FUNCTION  GetString     (stringID: INTEGER) : StringHandle;
PROCEDURE GetIndString  (VAR theString: Str255; strListID: INTEGER;
                         index: INTEGER);   [Not in ROM]
```

Byte Manipulation

```
FUNCTION  Munger      (h: Handle; offset: LONGINT; ptr1: Ptr; len1:
                       LONGINT; ptr2: Ptr; len2: LONGINT) : LONGINT;
PROCEDURE PackBits    (VAR srcPtr,dstPtr: Ptr; srcBytes: INTEGER);
PROCEDURE UnpackBits  (VAR srcPtr,dstPtr: Ptr; dstBytes: INTEGER);
```

Bit Manipulation

```
FUNCTION  BitTst (bytePtr: Ptr; bitNum: LONGINT) : BOOLEAN;
PROCEDURE BitSet (bytePtr: Ptr; bitNum: LONGINT);
PROCEDURE BitClr (bytePtr: Ptr; bitNum: LONGINT);
```

Logical Operations

```
FUNCTION BitAnd   (value1,value2: LONGINT) : LONGINT;
FUNCTION BitOr    (value1,value2: LONGINT) : LONGINT;
FUNCTION BitXor   (value1,value2: LONGINT) : LONGINT;
FUNCTION BitNot   (value: LONGINT) : LONGINT;
FUNCTION BitShift (value: LONGINT; count: INTEGER) : LONGINT;
```

Other Operations on Long Integers

```
FUNCTION  HiWord  (x: LONGINT) : INTEGER;
FUNCTION  LoWord  (x: LONGINT) : INTEGER;
PROCEDURE LongMul (a,b: LONGINT; VAR dest: Int64Bit);
```

Graphics Utilities

```
PROCEDURE ScreenRes      (VAR scrnHRes,scrnVRes: INTEGER);   [Not in ROM]
FUNCTION  GetIcon        (iconID: INTEGER) : Handle;
PROCEDURE PlotIcon       (theRect: Rect; theIcon: Handle);
FUNCTION  GetPattern     (patID: INTEGER) : PatHandle;
PROCEDURE GetIndPattern  (VAR thePattern: Pattern; patListID: INTEGER;
                          index: INTEGER);  [Not in ROM]
FUNCTION  GetCursor      (cursorID: INTEGER) : CursHandle;
PROCEDURE ShieldCursor   (shieldRect: Rect; offsetPt: Point);
FUNCTION  GetPicture     (picID: INTEGER) : PicHandle;
```

Miscellaneous Utilities

```
FUNCTION  DeltaPoint     (ptA,ptB: Point) : LONGINT;
FUNCTION  SlopeFromAngle (angle: INTEGER) : Fixed;
FUNCTION  AngleFromSlope (slope: Fixed) : INTEGER;
```

Assembly-Language Information

Constants

```
; Resource ID of standard pattern list

sysPatListID    .EQU    0

; Resource IDs of standard cursors

iBeamCursor     .EQU    1     ;to select text
crossCursor     .EQU    2     ;to draw graphics
plusCursor      .EQU    3     ;to select cells in structured documents
watchCursor     .EQU    4     ;to indicate a long wait
```

Variables

ScrVRes	Pixels per inch vertically (word)
ScrHRes	Pixels per inch horizontally (word)

VERTICAL RETRACE MANAGER

Constants

```
CONST { Result codes }

      noErr   =  0;   {no error}
      qErr    = -1;   {task entry isn't in the queue}
      vTypErr = -2;   {qType field isn't ORD(vType)}
```

Data Types

```
TYPE VBLTask = RECORD
                 qLink:     QElemPtr;    {next queue entry}
                 qType:     INTEGER;     {queue type}
                 vblAddr:   ProcPtr;     {pointer to task}
                 vblCount:  INTEGER;     {task frequency}
                 vblPhase:  INTEGER      {task phase}
               END;
```

Routines

```
FUNCTION VInstall      (vblTaskPtr: QElemPtr) : OSErr;
FUNCTION VRemove       (vblTaskPtr: QElemPtr) : OSErr;
FUNCTION GetVBLQHdr :   QHdrPtr;   [Not in ROM]
```

Assembly-Language Information

Constants

```
inVBL    .EQU   6   ;set if Vertical Retrace Manager is executing a task

; Result codes

noErr    .EQU    0  ;no error
qErr     .EQU   -1  ;task entry isn't in the queue
vTypErr  .EQU   -2  ;qType field isn't vType
```

Structure of Vertical Retrace Queue Entry

qLink	Pointer to next queue entry
qType	Queue type (word)
vblAddr	Address of task
vblCount	Task frequency (word)
vblPhase	Task phase (word)

Routines

Trap macro	On entry	On exit
_VInstall	A0: vblTaskPtr (ptr)	D0: result code (word)
_VRemove	A0: vblTaskPtr (ptr)	D0: result code (word)

Variables

VBLQueue	Vertical retrace queue header (10 bytes)

WINDOW MANAGER

Constants

```
CONST { Window definition IDs }

      documentProc   = 0;  {standard document window}
      dBoxProc       = 1;  {alert box or modal dialog box}
      plainDBox      = 2;  {plain box}
      altDBoxProc    = 3;  {plain box with shadow}
      noGrowDocProc  = 4;  {document window without size box}
      rDocProc       = 16; {rounded-corner window}

      { Window class, in windowKind field of window record }

      dialogKind  = 2;   {dialog or alert window}
      userKind    = 8;   {window created directly by the application}

      { Values returned by FindWindow }

      inDesk       = 0;  {none of the following}
      inMenuBar    = 1;  {in menu bar}
      inSysWindow  = 2;  {in system window}
      inContent    = 3;  {in content region (except grow, if active)}
      inDrag       = 4;  {in drag region}
      inGrow       = 5;  {in grow region (active window only)}
      inGoAway     = 6;  {in go-away region (active window only)}

      { Axis constraints for DragGrayRgn }

      noConstraint  = 0;  {no constraint}
      hAxisOnly     = 1;  {horizontal axis only}
      vAxisOnly     = 2;  {vertical axis only}

      { Messages to window definition function }

      wDraw       = 0;   {draw window frame}
      wHit        = 1;   {tell what region mouse button was pressed in}
      wCalcRgns   = 2;   {calculate strucRgn and contRgn}
      wNew        = 3;   {do any additional window initialization}
      wDispose    = 4;   {take any additional disposal actions}
      wGrow       = 5;   {draw window's grow image}
      wDrawGIcon  = 6;   {draw size box in content region}

      { Values returned by window definition function's hit routine }

      wNoHit      = 0;   {none of the following}
      wInContent  = 1;   {in content region (except grow, if active)}
      wInDrag     = 2;   {in drag region}
      wInGrow     = 3;   {in grow region (active window only)}
      wInGoAway   = 4;   {in go-away region (active window only)}
```

```
    { Resource ID of desktop pattern }

    deskPatID  = 16;
```

Data Types

```
TYPE WindowPtr  = GrafPtr;
     WindowPeek = ^WindowRecord;

     WindowRecord =
            RECORD
              port:          GrafPort;       {window's grafPort}
              windowKind:    INTEGER;        {window class}
              visible:       BOOLEAN;        {TRUE if visible}
              hilited:       BOOLEAN;        {TRUE if highlighted}
              goAwayFlag:    BOOLEAN;        {TRUE if has go-away region}
              spareFlag:     BOOLEAN;        {reserved for future use}
              strucRgn:      RgnHandle;      {structure region}
              contRgn:       RgnHandle;      {content region}
              updateRgn:     RgnHandle;      {update region}
              windowDefProc: Handle;         {window definition function}
              dataHandle:    Handle;         {data used by windowDefProc}
              titleHandle:   StringHandle;   {window's title}
              titleWidth:    INTEGER;        {width of title in pixels}
              controlList:   ControlHandle;  {window's control list}
              nextWindow:    WindowPeek;     {next window in window list}
              windowPic:     PicHandle;      {picture for drawing window}
              refCon:        LONGINT         {window's reference value}
            END;
```

Routines

Initialization and Allocation

```
PROCEDURE InitWindows;
PROCEDURE GetWMgrPort     (VAR wPort: GrafPtr);
FUNCTION  NewWindow       (wStorage: Ptr; boundsRect: Rect; title: Str255;
                           visible: BOOLEAN; procID: INTEGER; behind:
                           WindowPtr; goAwayFlag: BOOLEAN; refCon:
                           LONGINT) : WindowPtr;
FUNCTION  GetNewWindow    (windowID: INTEGER; wStorage: Ptr; behind:
                           WindowPtr) : WindowPtr;
PROCEDURE CloseWindow     (theWindow: WindowPtr);
PROCEDURE DisposeWindow   (theWindow: WindowPtr);
```

Window Display

```
PROCEDURE SetWTitle     (theWindow: WindowPtr; title: Str255);
PROCEDURE GetWTitle     (theWindow: WindowPtr; VAR title: Str255);
PROCEDURE SelectWindow  (theWindow: WindowPtr);
PROCEDURE HideWindow    (theWindow: WindowPtr);
PROCEDURE ShowWindow    (theWindow: WindowPtr);
PROCEDURE ShowHide      (theWindow: WindowPtr; showFlag: BOOLEAN);
PROCEDURE HiliteWindow  (theWindow: WindowPtr; fHilite: BOOLEAN);
PROCEDURE BringToFront  (theWindow: WindowPtr);
PROCEDURE SendBehind    (theWindow,behindWindow: WindowPtr);
FUNCTION  FrontWindow :  WindowPtr;
PROCEDURE DrawGrowIcon  (theWindow: WindowPtr);
```

Mouse Location

```
FUNCTION FindWindow   (thePt: Point; VAR whichWindow: WindowPtr) :
                       INTEGER;
FUNCTION TrackGoAway  (theWindow: WindowPtr; thePt: Point) : BOOLEAN;
```

Window Movement and Sizing

```
PROCEDURE MoveWindow  (theWindow: WindowPtr; hGlobal,vGlobal: INTEGER;
                       front: BOOLEAN);
PROCEDURE DragWindow  (theWindow: WindowPtr; startPt: Point; boundsRect:
                       Rect);
FUNCTION  GrowWindow  (theWindow: WindowPtr; startPt: Point; sizeRect:
                       Rect) : LONGINT;
PROCEDURE SizeWindow  (theWindow: WindowPtr; w,h: INTEGER; fUpdate:
                       BOOLEAN);
```

Update Region Maintenance

```
PROCEDURE InvalRect    (badRect: Rect);
PROCEDURE InvalRgn     (badRgn: RgnHandle);
PROCEDURE ValidRect    (goodRect: Rect);
PROCEDURE ValidRgn     (goodRgn: RgnHandle);
PROCEDURE BeginUpdate  (theWindow: WindowPtr);
PROCEDURE EndUpdate    (theWindow: WindowPtr);
```

Miscellaneous Routines

```
PROCEDURE SetWRefCon    (theWindow: WindowPtr; data: LONGINT);
FUNCTION  GetWRefCon    (theWindow: WindowPtr) : LONGINT;
PROCEDURE SetWindowPic  (theWindow: WindowPtr; pic: PicHandle);
FUNCTION  GetWindowPic  (theWindow: WindowPtr) : PicHandle;
FUNCTION  PinRect       (theRect: Rect; thePt: Point) : LONGINT;
FUNCTION  DragGrayRgn   (theRgn: RgnHandle; startPt: Point; limitRect,
                         slopRect: Rect; axis: INTEGER; actionProc:
                         ProcPtr) : LONGINT;
```

Low-Level Routines

```
FUNCTION  CheckUpdate     (VAR theEvent: EventRecord) : BOOLEAN;
PROCEDURE ClipAbove       (window: WindowPeek);
PROCEDURE SaveOld         (window: WindowPeek);
PROCEDURE DrawNew         (window: WindowPeek; update: BOOLEAN);
PROCEDURE PaintOne        (window: WindowPeek; clobberedRgn: RgnHandle);
PROCEDURE PaintBehind     (startWindow: WindowPeek; clobberedRgn:
                           RgnHandle);
PROCEDURE CalcVis         (window: WindowPeek);
PROCEDURE CalcVisBehind   (startWindow: WindowPeek; clobberedRgn:
                           RgnHandle);
```

Diameters of Curvature for Rounded-Corner Windows

Window definition ID	Diameters of curvature
rDocProc	16, 16
rDocProc + 1	4, 4
rDocProc + 2	6, 6
rDocProc + 3	8, 8
rDocProc + 4	10, 10
rDocProc + 5	12, 12
rDocProc + 6	20, 20
rDocProc + 7	24, 24

Window Definition Function

```
FUNCTION MyWindow (varCode: INTEGER; theWindow: WindowPtr; message:
                   INTEGER; param: LONGINT) : LONGINT;
```

Assembly-Language Information

Constants

```
; Window definition IDs

documentProc       .EQU  0  ;standard document window
dBoxProc           .EQU  1  ;alert box or modal dialog box
plainDBox          .EQU  2  ;plain box
```

```
altDBoxProc       .EQU  3  ;plain box with shadow
noGrowDocProc     .EQU  4  ;document window without size box
rDocProc          .EQU  16 ;rounded-corner window

; Window class, in windowKind field of window record

dialogKind        .EQU  2   ;dialog or alert window
userKind          .EQU  8   ;window created directly by the application

; Values returned by FindWindow

inDesk            .EQU  0   ;none of the following
inMenuBar         .EQU  1   ;in menu bar
inSysWindow       .EQU  2   ;in system window
inContent         .EQU  3   ;in content region (except grow, if active)
inDrag            .EQU  4   ;in drag region
inGrow            .EQU  5   ;in grow region (active window only)
inGoAway          .EQU  6   ;in go-away region (active window only)

; Axis constraints for DragGrayRgn

noConstraint      .EQU  0   ;no constraint
hAxisOnly         .EQU  1   ;horizontal axis only
vAxisOnly         .EQU  2   ;vertical axis only

; Messages to window definition function

wDrawMsg          .EQU  0   ;draw window frame
wHitMsg           .EQU  1   ;tell what region mouse button was pressed in
wCalcRgnMsg       .EQU  2   ;calculate strucRgn and contRgn
wInitMsg          .EQU  3   ;do any additional window initialization
wDisposeMsg       .EQU  4   ;take any additional disposal actions
wGrowMsg          .EQU  5   ;draw window's grow image
wGIconMsg         .EQU  6   ;draw size box in content region

; Value returned by window definition function's hit routine

wNoHit            .EQU  0   ;none of the following
wInContent        .EQU  1   ;in content region (except grow, if active)
wInDrag           .EQU  2   ;in drag region
wInGrow           .EQU  3   ;in grow region (active window only)
wInGoAway         .EQU  4   ;in go-away region (active window only)

; Resource ID of desktop pattern

deskPatID         .EQU  16
```

Window Record Data Structure

windowPort	Window's grafPort (portRec bytes)
windowKind	Window class (word)
wVisible	Nonzero if window is visible (byte)
wHilited	Nonzero if window is highlighted (byte)

wGoAway	Nonzero if window has go-away region (byte)
structRgn	Handle to structure region of window
contRgn	Handle to content region of window
updateRgn	Handle to update region of window
windowDef	Handle to window definition function
wDataHandle	Handle to data used by window definition function
wTitleHandle	Handle to window's title (preceded by length byte)
wTitleWidth	Width of title in pixels (word)
wControlList	Handle to window's control list
nextWindow	Pointer to next window in window list
windowPic	Picture handle for drawing window
wRefCon	Window's reference value (long)
windowSize	Size in bytes of window record

Special Macro Names

Pascal name	Macro name
CalcVisBehind	_CalcVBehind
DisposeWindow	_DisposWindow
DragGrayRgn	_DragGrayRgn or, after setting the global variable DragPattern, _DragTheRgn

Variables

WindowList	Pointer to first window in window list
SaveUpdate	Flag for whether to generate update events (word)
PaintWhite	Flag for whether to paint window white before update event (word)
CurActivate	Pointer to window to receive activate event
CurDeactive	Pointer to window to receive deactivate event
GrayRgn	Handle to region drawn as desktop
DeskPattern	Pattern with which desktop is painted (8 bytes)
DeskHook	Address of procedure for painting desktop or responding to clicks on desktop
WMgrPort	Pointer to Window Manager port
GhostWindow	Pointer to window never to be considered frontmost
DragHook	Address of procedure to execute during TrackGoAway, DragWindow, GrowWindow, and DragGrayRgn
DragPattern	Pattern of dragged region's outline (8 bytes)
OldStructure	Handle to saved structure region
OldContent	Handle to saved content region
SaveVisRgn	Handle to saved visRgn

ASSEMBLY LANGUAGE

Variables

OneOne	$00010001
MinusOne	$FFFFFFFF
Lo3Bytes	$00FFFFFF
Scratch20	20-byte scratch area
Scratch8	8-byte scratch area
ToolScratch	8-byte scratch area
ApplScratch	12-byte scratch area reserved for use by applications
ROMBase	Base address of ROM
RAMBase	Trap dispatch table's base address for routines in RAM
CurrentA5	Address of boundary between application globals and application parameters

Hardware

Warning: This information applies only to the Macintosh 128K and 512K, not to the Macintosh XL.

Constants

```
; VIA base addresses

vBase     .EQU   $EFE1FE    ;main base for VIA chip (in variable VIA)
aVBufB    .EQU   vBase      ;register B base
aVBufA    .EQU   $EFFFFE    ;register A base
aVBufM    .EQU   aVBufB     ;register containing mouse signals
aVIFR     .EQU   $EFFBFE    ;interrupt flag register
aVIER     .EQU   $EFFDFE    ;interrupt enable register

; Offsets from vBase

vBufB     .EQU   512*0      ;register B (zero offset)
vDirB     .EQU   512*2      ;register B direction register
vDirA     .EQU   512*3      ;register A direction register
vT1C      .EQU   512*4      ;timer 1 counter (low-order byte)
vT1CH     .EQU   512*5      ;timer 1 counter (high-order byte)
vT1L      .EQU   512*6      ;timer 1 latch (low-order byte)
vT1LH     .EQU   512*7      ;timer 1 latch (high-order byte)
vT2C      .EQU   512*8      ;timer 2 counter (low-order byte)
vT2CH     .EQU   512*9      ;timer 2 counter (high-order byte)
vSR       .EQU   512*10     ;shift register (keyboard)
vACR      .EQU   512*11     ;auxiliary control register
vPCR      .EQU   512*12     ;peripheral control register
vIFR      .EQU   512*13     ;interrupt flag register
```

```
vIER        .EQU    512*14      ;interrupt enable register
vBufA       .EQU    512*15      ;register A

; VIA register A constants

vAOut       .EQU    $7F         ;direction register A:  1 bits = outputs
vAInit      .EQU    $7B         ;initial value for vBufA (medium volume)
vSound      .EQU    7           ;sound volume bits

; VIA register A bit numbers

vSndPg2     .EQU    3           ;0 = alternate sound buffer
vOverlay    .EQU    4           ;1 = ROM overlay (system startup only)
vHeadSel    .EQU    5           ;disk SEL control line
vPage2      .EQU    6           ;0 = alternate screen buffer
vSCCWReq    .EQU    7           ;SCC wait/request line

; VIA register B constants

vBOut       .EQU    $87         ;direction register B:  1 bits = outputs
vBInit      .EQU    $07         ;initial value for vBufB

; VIA register B bit numbers

rTCData     .EQU    0           ;real-time clock serial data line
rTCClk      .EQU    1           ;real-time clock data-clock line
rTCEnb      .EQU    2           ;real-time clock serial enable
vSW         .EQU    3           ;0 = mouse button is down
vX2         .EQU    4           ;mouse X quadrature level
vY2         .EQU    5           ;mouse Y quadrature level
vH4         .EQU    6           ;1 = horizontal blanking
vSndEnb     .EQU    7           ;0 = sound enabled, 1 = disabled

; SCC base addresses

sccRBase    .EQU    $9FFFF8     ;SCC base read address (in variable SCCRd)
sccWBase    .EQU    $BFFFF9     ;SCC base write address (in variable SCCWr)

; Offsets from SCC base addresses

aData       .EQU    6           ;channel A data in or out
aCtl        .EQU    2           ;channel A control
bData       .EQU    4           ;channel B data in or out
bCtl        .EQU    0           ;channel B control

; Bit numbers for control register RR0

rxBF        .EQU    0           ;1 = SCC receive buffer full
txBE        .EQU    2           ;1 = SCC send buffer empty
```

```
; IWM base address

dBase       .EQU    $DFE1FF    ;IWM base address (in variable IWM)

; Offsets from dBase

ph0L        .EQU    512*0      ;CA0 off (0)
ph0H        .EQU    512*1      ;CA0 on (1)
ph1L        .EQU    512*2      ;CA1 off (0)
ph1H        .EQU    512*3      ;CA1 on (1)
ph2L        .EQU    512*4      ;CA2 off (0)
ph2H        .EQU    512*5      ;CA2 on (1)
ph3L        .EQU    512*6      ;LSTRB off (low)
ph3H        .EQU    512*7      ;LSTRB on (high)
mtrOff      .EQU    512*8      ;disk enable off
mtrOn       .EQU    512*9      ;disk enable on
intDrive    .EQU    512*10     ;select internal drive
extDrive    .EQU    512*11     ;select external drive
q6L         .EQU    512*12     ;Q6 off
q6H         .EQU    512*13     ;Q6 on
q7L         .EQU    512*14     ;Q7 off
q7H         .EQU    512*15     ;Q7 on

; Screen and sound addresses for 512K Macintosh (will also work for
; 128K, since addresses wrap)

screenLow   .EQU    $7A700     ;top left corner of main screen buffer
soundLow    .EQU    $7FD00     ;main sound buffer (in variable SoundBase)
pwmBuffer   .EQU    $7FD01     ;main disk speed buffer
ovlyRAM     .EQU    $600000    ;RAM start address when overlay is set
ovlyScreen  .EQU    $67A700    ;screen start with overlay set
romStart    .EQU    $400000    ;ROM start address (in variable ROMBase)
```

Variables

ROMBase	Base address of ROM
SoundBase	Address of main sound buffer
SCCRd	SCC read base address
SCCWr	SCC write base address
IWM	IWM base address
VIA	VIA base address

Exception Vectors

Location	Purpose
$00	Reset: initial stack pointer (not a vector)
$04	Reset: initial vector
$08	Bus error

Location	Purpose
$0C	Address error
$10	Illegal instruction
$14	Divide by zero
$18	CHK instruction
$1C	TRAPV instruction
$20	Privilege violation
$24	Trace interrupt
$28	Line 1010 emulator
$2C	Line 1111 emulator
$30-$3B	Unassigned (reserved)
$3C	Uninitialized interrupt
$40-$5F	Unassigned (reserved)
$60	Spurious interrupt
$64	VIA interrupt
$68	SCC interrupt
$6C	VIA+SCC vector (temporary)
$70	Interrupt switch
$74	Interrupt switch + VIA
$78	Interrupt switch + SCC
$7C	Interrupt switch + VIA + SCC
$80-$BF	TRAP instructions
$C0-$FF	Unassigned (reserved)

APPENDIX A: RESULT CODES

This appendix lists all the result codes returned by the Macintosh system software. They're ordered by value, for convenience when debugging; the names you should actually use in your program are also listed.

The result codes are grouped roughly according to the lowest level at which the error may occur. This doesn't mean that only routines at that level may cause those errors; higher-level software may yield the same result codes. For example, an Operating System Utility routine that calls the Memory Manager may return one of the Memory Manager result codes. Where a different or more specific meaning is appropriate in a different context, that meaning is also listed.

Value	Name	Meaning
0	noErr	No error

Operating System Event Manager Error

1	evtNotEnb	Event type not designated in system event mask

Printing Manager Errors

128	iPrAbort	Application or user requested abort
–1	iPrSavPFil	Saving spool file

Queuing Errors

–1	qErr	Entry not in queue
–2	vTypErr	QType field of entry in vertical retrace queue isn't vType (in Pascal, ORD(vType))

Device Manager Errors

–17	controlErr	Driver can't respond to this Control call Unimplemented control instruction (Printing Manager)
–18	statusErr	Driver can't respond to this Status call
–19	readErr	Driver can't respond to Read calls
–20	writErr	Driver can't respond to Write calls
–21	badUnitErr	Driver reference number doesn't match unit table
–22	unitEmptyErr	Driver reference number specifies NIL handle in unit table
–23	openErr	Requested read/write permission doesn't match driver's open permission Attempt to open RAM Serial Driver failed
–25	dRemovErr	Attempt to remove an open driver
–26	dInstErr	Couldn't find driver in resource file

–27	abortErr iIOAbort	I/O request aborted by KillIO I/O abort error (Printing Manager)
–28	notOpenErr	Driver isn't open

File Manager Errors

–33	dirFulErr	File directory full
–34	dskFulErr	All allocation blocks on the volume are full
–35	nsvErr	Specified volume doesn't exist
–36	ioErr	I/O error
–37	bdNamErr	Bad file name or volume name (perhaps zero-length)
–38	fnOpnErr	File not open
–39	eofErr	Logical end-of-file reached during read operation
–40	posErr	Attempt to position before start of file
–42	tmfoErr	Too many files open
–43	fnfErr	File not found
–44	wPrErr	Volume is locked by a hardware setting
–45	fLckdErr	File is locked
–46	vLckdErr	Volume is locked by a software flag
–47	fBsyErr	File is busy; one or more files are open
–48	dupFNErr	File with specified name and version number already exists
–49	opWrErr	The read/write permission of only one access path to a file can allow writing
–50	paramErr	Error in parameter list Parameters don't specify an existing volume, and there's no default volume (File Manager) Bad positioning information (Disk Driver) Bad drive number (Disk Initialization Package)
–51	rfNumErr	Path reference number specifies nonexistent access path
–52	gfpErr	Error during GetFPos
–53	volOffLinErr	Volume not on-line
–54	permErr	Attempt to open locked file for writing
–55	volOnLinErr	Specified volume is already mounted and on-line
–56	nsDrvErr	No such drive; specified drive number doesn't match any number in the drive queue
–57	noMacDskErr	Not a Macintosh disk; volume lacks Macintosh-format directory
–58	extFSErr	External file system; file-system identifier is nonzero, or path reference number is greater than 1024

–59	fsRnErr	Problem during rename
–60	badMDBErr	Bad master directory block; must reinitialize volume
–61	wrPermErr	Read/write permission doesn't allow writing

Low-Level Disk Errors

–64	noDriveErr	Drive isn't connected
–65	offLinErr	No disk in drive
–66	noNybErr	Disk is probably blank
–67	noAdrMkErr	Can't find an address mark
–68	dataVerErr	Read-verify failed
–69	badCksmErr	Bad address mark
–70	badBtSlpErr	Bad address mark
–71	noDtaMkErr	Can't find a data mark
–72	badDCksum	Bad data mark
–73	badDBtSlp	Bad data mark
–74	wrUnderrun	Write underrun occurred
–75	cantStepErr	Drive error
–76	tk0BadErr	Can't find track 0
–77	initIWMErr	Can't initialize disk controller chip
–78	twoSideErr	Tried to read side 2 of a disk in a single-sided drive
–79	spdAdjErr	Can't correctly adjust disk speed
–80	seekErr	Drive error
–81	sectNFErr	Can't find sector

Also, to check for any low-level disk error:

–84	firstDskErr	First of the range of low-level disk errors
–64	lastDskErr	Last of the range of low-level disk errors

Clock Chip Errors

–85	clkRdErr	Unable to read clock
–86	clkWrErr	Time written did not verify
–87	prWrErr	Parameter RAM written did not verify
–88	prInitErr	Validity status is not $A8

AppleTalk Manager Errors

–91	ddpSktErr	DDP socket error: socket already active; not a well-known socket; socket table full; all dynamic socket numbers in use
–92	ddpLenErr	DDP datagram or ALAP data length too big
–93	noBridgeErr	No bridge found
–94	lapProtErr	ALAP error attaching/detaching ALAP protocol type: attach error when ALAP protocol type is negative, not in range, or already in table, or when table is full; detach error when ALAP protocol type isn't in table
–95	excessCollsns	ALAP no CTS received after 32 RTS's, or line sensed in use 32 times (not necessarily caused by collisions)
–97	portInUse	Driver Open error, port already in use
–98	portNotCf	Driver Open error, port not configured for this connection

Scrap Manager Errors

–100	noScrapErr	Desk scrap isn't initialized
–102	noTypeErr	No data of the requested type

Memory Manager Errors

–108	memFullErr iMemFullErr	Not enough room in heap zone Not enough room in heap zone (Printing Manager)
–109	nilHandleErr	NIL master pointer
–111	memWZErr	Attempt to operate on a free block
–112	memPurErr	Attempt to purge a locked block
–117	memLockedErr	Block is locked

Resource Manager Errors

–192	resNotFound	Resource not found
–193	resFNotFound	Resource file not found
–194	addResFailed	AddResource failed
–196	rmvResFailed	RmveResource failed

Additional AppleTalk Manager Errors

–1024	nbpBuffOvr	NBP buffer overflow
–1025	nbpNoConfirm	NBP name not confirmed
–1026	nbpConfDiff	NBP name confirmed for different socket
–1027	nbpDuplicate	NBP duplicate name already exists
–1028	nbpNotFound	NBP name not found

–1029	nbpNISErr	NBP names information socket error
–1096	reqFailed	ATPSndRequest failed: retry count exceeded
–1097	tooManyReqs	ATP too many concurrent requests
–1098	tooManySkts	ATP too many responding sockets
–1099	badATPSkt	ATP bad responding socket
–1100	badBuffNum	ATP bad sequence number
–1101	noRelErr	ATP no release received
–1102	cbNotFound	ATP control block not found
–1103	noSendResp	ATPAddRsp issued before ATPSndRsp
–1104	noDataArea	Too many outstanding ATP calls
–1105	reqAborted	Request aborted
–3101	buf2SmallErr	ALAP frame too large for buffer DDP datagram too large for buffer
–3102	noMPPError	MPP driver not installed
–3103	cksumErr	DDP bad checksum
–3104	extractErr	NBP can't find tuple in buffer
–3105	readQErr	Socket or protocol type invalid or not found in table
–3106	atpLenErr	ATP response message too large
–3107	atpBadRsp	Bad response from ATPRequest
–3108	recNotFnd	ABRecord not found
–3109	sktClosedErr	Asynchronous call aborted because socket was closed before call was completed

APPENDIX B: ROUTINES THAT MAY MOVE OR PURGE MEMORY

This appendix lists all the routines that may move or purge blocks in the heap. As described in chapter 1 of Volume II, calling these routines may cause problems if a handle has been dereferenced. None of these routines may be called from within an interrupt, such as in a completion routine or a VBL task.

The Pascal name of each routine is shown, except for a few cases where there's no Pascal interface corresponding to a particular trap; in those cases, the trap macro name is shown instead (without its initial underscore character).

AddResMenu
Alert
AppendMenu
ATPAddRsp
ATPCloseSocket
ATPGetRequest
ATPLoad
ATPOpenSocket
ATPReqCancel
ATPRequest
ATPResponse
ATPRspCancel
ATPSndRequest
ATPSndRsp
ATPUnload
BeginUpdate
BringToFront
Button
CalcMenuSize
CalcVis
CalcVisBehind
CautionAlert
Chain
ChangedResource
CharWidth
CheckItem
CheckUpdate
ClipAbove
ClipRect
CloseDialog
ClosePicture
ClosePoly
ClosePort
CloseResFile
CloseRgn
CloseWindow
CompactMem
Control
CopyBits
CopyRgn
CouldAlert
CouldDialog
CreateResFile
DDPCloseSocket
DDPOpenSocket
DDPRdCancel
DDPRead
DDPWrite
DialogSelect
DIBadMount
DiffRgn
DIFormat
DILoad
DiskEject
DisposDialog
DisposeControl
DisposeMenu
DisposeRgn
DisposeWindow
DisposHandle
DisposPtr
DIUnload
DIVerify
DIZero
DlgCopy
DlgCut
DlgDelete
DlgPaste
DragControl
DragGrayRgn
DragWindow
DrawChar
DrawDialog
DrawGrowIcon
DrawMenuBar
DrawNew
DrawPicture
DrawString
DrawText
DriveStatus
DrvrInstall
DrvrRemove
Eject
EmptyHandle
EndUpdate
EraseArc
EraseOval
ErasePoly
EraseRect
EraseRgn
EraseRoundRect
EventAvail
ExitToShell
FillArc
FillOval
FillPoly
FillRect
FillRgn
FillRoundRect
FindControl
FlashMenuBar
FlushVol
FMSwapFont
FrameArc
FrameOval
FramePoly
FrameRect
FrameRgn
FrameRoundRect
FreeAlert
FreeDialog
FreeMem
GetClip
GetCursor

GetDCtlEntry
GetDItem
GetFNum
GetFontInfo
GetFontName
GetIcon
GetIndPattern
GetIndResource
GetIndString
GetKeys
GetMenu
GetMenuBar
GetMouse
GetNamedResource
GetNewControl
GetNewDialog
GetNewMBar
GetNewWindow
GetNextEvent
GetPattern
GetPicture
GetResource
GetScrap
GetString
GrowWindow
HandAndHand
HandToHand
HideControl
HideWindow
HiliteControl
HiliteMenu
HiliteWindow
InitAllPacks
InitApplZone
InitFonts
InitMenus
InitPack
InitPort
InitResources
InitWindows
InitZone
InsertMenu
InsertResMenu
InsetRgn
InvalRect
InvalRgn
InvertArc
InvertOval
InvertPoly
InvertRect
InvertRgn
InvertRoundRect
IUCompString
IUDatePString
IUDateString
IUEqualString
IUGetIntl
IUMagIDString
IUMagString
IUMetric
IUSetIntl
IUTimePString
IUTimeString
KillControls
KillPicture
KillPoly
LAPCloseProtocol
LAPOpenProtocol
LAPRdCancel
LAPRead
LAPWrite
Launch
Line
LineTo
LoadResource
LoadScrap
LoadSeg
MapRgn
MenuKey
MenuSelect
ModalDialog
MoreMasters
MoveControl
MoveHHi
MoveWindow
MPPClose
MPPOpen
Munger
NBPConfirm
NBPExtract
NBPLoad
NBPLookup
NBPRegister
NBPRemove
NBPUnload
NewControl
NewDialog
NewHandle
NewMenu
NewPtr
NewRgn
NewString
NewWindow
NoteAlert
NumToString
OpenDeskAcc
OpenPicture
OpenPoly
OpenPort
OpenResFile
OpenRgn
PaintArc
PaintBehind
PaintOne
PaintOval
PaintPoly
PaintRect
PaintRgn
PaintRoundRect
ParamText
PBControl
PBEject
PBFlushVol
PBMountVol
PBOffLine
PBOpen
PBOpenRF
PBStatus
PicComment
PlotIcon
PrClose
PrCloseDoc
PrClosePage
PrCtlCall
PrDrvrDCE
PrDrvrVers
PrintDefault
PrJobDialog
PrJobMerge
PrOpen
PrOpenDoc
PrOpenPage
PrPicFile
PrStlDialog
PrValidate
PtrAndHand
PtrToHand
PtrToXHand
PurgeMem
PutScrap
RAMSDClose
RAMSDOpen
RealFont
ReallocHandle
RecoverHandle
RectRgn

ReleaseResource
ResrvMem
Restart
RmveResource
RsrcZoneInit
SaveOld
ScrollRect
SectRgn
SelectWindow
SelIText
SendBehind
SerClrBrk
SerGetBrk
SerHShake
SerReset
SerSetBrk
SerSetBuf
SerStatus
SetApplBase
SetClip
SetCTitle
SetCtlMax
SetCtlMin
SetCtlValue
SetDItem
SetEmptyRgn
SetFontLock
SetHandleSize
SetItem
SetItemIcon
SetItemMark
SetItemStyle
SetIText
SetPtrSize
SetRectRgn
SetResInfo
SetString
SetTagBuffer
SetWTitle
SFGetFile
SFPGetFile
SFPPutFile
SFPutFile
ShowControl
ShowHide
ShowWindow
SizeControl
SizeWindow
StartSound
Status
StdArc
StdBits
StdComment
StdLine
StdOval
StdPoly
StdPutPic
StdRect
StdRgn
StdRRect
StdText
StdTxMeas
StillDown
StopAlert
StopSound
StringToNum
StringWidth
SysBeep
SysError
SystemClick
SystemEdit
SystemMenu
TEActivate
TECalText
TEClick
TECopy
TECut
TEDeactivate
TEDelete
TEDispose
TEFromScrap
TEGetText
TEIdle
TEInit
TEInsert
TEKey
TENew
TEPaste
TEScroll
TESetJust
TESetSelect
TESetText
TestControl
TEToScrap
TEUpdate
TextBox
TextWidth
TickCount
TrackControl
TrackGoAway
UnionRgn
UnloadScrap
UnloadSeg
ValidRect
ValidRgn
WaitMouseUp
XorRgn
ZeroScrap

APPENDIX C: SYSTEM TRAPS

This appendix lists the trap macros for the Toolbox and Operating System routines and their corresponding trap word values in hexadecimal. The "Name" column gives the trap macro name (without its initial underscore character). In those cases where the name of the equivalent Pascal call is different, the Pascal name appears indented under the main entry. The routines in Macintosh packages are listed under the macros they invoke after pushing a routine selector onto the stack; the routine selector follows the Pascal routine name in parentheses.

There are two tables: The first is ordered alphabetically by name; the second is ordered numerically by trap number, for use when debugging. (The trap number is the last two digits of the trap word unless the trap word begins with A9, in which case the trap number is 1 followed by the last two digits of the trap word.)

> **Note:** The Operating System Utility routines GetTrapAddress and SetTrapAddress take a trap number as a parameter, not a trap word.

> **Warning:** Traps that aren't currently used by the system are reserved for future use.

Name	Trap word
AddDrive	A04E
(internal use only)	
AddPt	A87E
AddResMenu	A94D
AddResource	A9AB
Alert	A985
Allocate	A010
PBAllocate	
AngleFromSlope	A8C4
AppendMenu	A933
BackColor	A863
BackPat	A87C
BeginUpdate	A922
BitAnd	A858
BitClr	A85F
BitNot	A85A
BitOr	A85B
BitSet	A85E
BitShift	A85C
BitTst	A85D
BitXor	A859
BlockMove	A02E
BringToFront	A920
Button	A974
CalcMenuSize	A948
CalcVBehind	A90A
CalcVisBehind	
CalcVis	A909
CautionAlert	A988
Chain	A9F3
ChangedResource	A9AA
CharWidth	A88D
CheckItem	A945
CheckUpdate	A911
ClearMenuBar	A934
ClipAbove	A90B
ClipRect	A87B
Close	A001
PBClose	
CloseDeskAcc	A9B7
CloseDialog	A982
ClosePgon	A8CC
ClosePoly	
ClosePicture	A8F4
ClosePort	A87D
CloseResFile	A99A
CloseRgn	A8DB
CloseWindow	A92D
CmpString	A03C
EqualString	
ColorBit	A864
CompactMem	A04C
Control	A004
PBControl	
CopyBits	A8EC
CopyRgn	A8DC
CouldAlert	A989
CouldDialog	A979
CountMItems	A950
CountResources	A99C

Name	Trap word
CountTypes	A99E
Create	A008
PBCreate	
CreateResFile	A9B1
CurResFile	A994
Date2Secs	A9C7
Delay	A03B
Delete	A009
PBDelete	
DeleteMenu	A936
DeltaPoint	A94F
Dequeue	A96E
DetachResource	A992
DialogSelect	A980
DiffRgn	A8E6
DisableItem	A93A
DisposControl	A955
DisposeControl	
DisposDialog	A983
DisposHandle	A023
DisposMenu	A932
DisposeMenu	
DisposPtr	A01F
DisposRgn	A8D9
DisposeRgn	
DisposWindow	A914
DisposeWindow	
DragControl	A967
DragGrayRgn	A905
DragTheRgn	A926
DragWindow	A925
DrawChar	A883
DrawControls	A969
DrawDialog	A981
DrawGrowIcon	A904
DrawMenuBar	A937
DrawNew	A90F
DrawPicture	A8F6
DrawString	A884
DrawText	A885
DrvrInstall	A03D
(internal use only)	
DrvrRemove	A03E
(internal use only)	
Eject	A017
PBEject	
Elems68K	A9EC
EmptyHandle	A02B
EmptyRect	A8AE
EmptyRgn	A8E2
EnableItem	A939

Name	Trap word
EndUpdate	A923
Enqueue	A96F
EqualPt	A881
EqualRect	A8A6
EqualRgn	A8E3
EraseArc	A8C0
EraseOval	A8B9
ErasePoly	A8C8
EraseRect	A8A3
EraseRgn	A8D4
EraseRoundRect	A8B2
ErrorSound	A98C
EventAvail	A971
ExitToShell	A9F4
FillArc	A8C2
FillOval	A8BB
FillPoly	A8CA
FillRect	A8A5
FillRgn	A8D6
FillRoundRect	A8B4
FindControl	A96C
FindWindow	A92C
FixMul	A868
FixRatio	A869
FixRound	A86C
FlashMenuBar	A94C
FlushEvents	A032
FlushFile	A045
PBFlushFile	
FlushVol	A013
PBFlushVol	
FMSwapFont	A901
ForeColor	A862
FP68K	A9EB
FrameArc	A8BE
FrameOval	A8B7
FramePoly	A8C6
FrameRect	A8A1
FrameRgn	A8D2
FrameRoundRect	A8B0
FreeAlert	A98A
FreeDialog	A97A
FreeMem	A01C
FrontWindow	A924
GetAppParms	A9F5
GetClip	A87A
GetCRefCon	A95A
GetCTitle	A95E
GetCtlAction	A96A
GetCtlValue	A960
GetCursor	A9B9

Name	Trap word
GetDItem	A98D
GetEOF	A011
PBGetEOF	
GetFileInfo	A00C
PBGetFInfo	
GetFName	A8FF
GetFontName	
GetFNum	A900
GetFontInfo	A88B
GetFPos	A018
PBGetFPos	
GetHandleSize	A025
GetIcon	A9BB
GetIndResource	A99D
GetIndType	A99F
GetItem	A946
GetIText	A990
GetItmIcon	A93F
GetItemIcon	
GetItmMark	A943
GetItemMark	
GetItmStyle	A941
GetItemStyle	
GetKeys	A976
GetMaxCtl	A962
GetCtlMax	
GetMenuBar	A93B
GetMHandle	A949
GetMinCtl	A961
GetCtlMin	
GetMouse	A972
GetNamedResource	A9A1
GetNewControl	A9BE
GetNewDialog	A97C
GetNewMBar	A9C0
GetNewWindow	A9BD
GetNextEvent	A970
GetOSEvent	A031
GetPattern	A9B8
GetPen	A89A
GetPenState	A898
GetPicture	A9BC
GetPixel	A865
GetPort	A874
GetPtrSize	A021
GetResAttrs	A9A6
GetResFileAttrs	A9F6
GetResInfo	A9A8
GetResource	A9A0
GetRMenu	A9BF
GetMenu	

Name	Trap word
GetScrap	A9FD
GetString	A9BA
GetTrapAddress	A146
GetVol	A014
PBGetVol	
GetVolInfo	A007
PBGetVInfo	
GetWindowPic	A92F
GetWMgrPort	A910
GetWRefCon	A917
GetWTitle	A919
GetZone	A11A
GlobalToLocal	A871
GrafDevice	A872
GrowWindow	A92B
HandAndHand	A9E4
HandleZone	A126
HandToHand	A9E1
HideControl	A958
HideCursor	A852
HidePen	A896
HideWindow	A916
HiliteControl	A95D
HiliteMenu	A938
HiliteWindow	A91C
HiWord	A86A
HLock	A029
HNoPurge	A04A
HomeResFile	A9A4
HPurge	A049
HUnlock	A02A
InfoScrap	A9F9
InitAllPacks	A9E6
InitApplZone	A02C
InitCursor	A850
InitDialogs	A97B
InitFonts	A8FE
InitGraf	A86E
InitMenus	A930
InitPack	A9E5
InitPort	A86D
InitQueue	A016
FInitQueue	
InitResources	A995
InitUtil	A03F
InitWindows	A912
InitZone	A019
InsertMenu	A935
InsertResMenu	A951
InsetRect	A8A9
InsetRgn	A8E1

Name	Trap word
InvalRect	A928
InvalRgn	A927
InverRect	A8A4
InvertRect	
InverRgn	A8D5
InvertRgn	
InverRoundRect	A8B3
InvertRoundRect	
InvertArc	A8C1
InvertOval	A8BA
InvertPoly	A8C9
IsDialogEvent	A97F
KillControls	A956
KillIO	A006
PBKillIO	
KillPicture	A8F5
KillPoly	A8CD
Launch	A9F2
Line	A892
LineTo	A891
LoadResource	A9A2
LoadSeg	A9F0
LocalToGlobal	A870
LodeScrap	A9FB
LoadScrap	
LongMul	A867
LoWord	A86B
MapPoly	A8FC
MapPt	A8F9
MapRect	A8FA
MapRgn	A8FB
MaxMem	A11D
MenuKey	A93E
MenuSelect	A93D
ModalDialog	A991
MoreMasters	A036
MountVol	A00F
PBMountVol	
Move	A894
MoveControl	A959
MovePortTo	A877
MoveTo	A893
MoveWindow	A91B
Munger	A9E0
NewControl	A954
NewDialog	A97D
NewHandle	A122
NewMenu	A931
NewPtr	A11E
NewRgn	A8D8
NewString	A906

Name	Trap word
NewWindow	A913
NoteAlert	A987
ObscureCursor	A856
Offline	A035
PBOffline	
OffsetPoly	A8CE
OffsetRect	A8A8
OfsetRgn	A8E0
OffsetRgn	
Open	A000
PBOpen	
OpenDeskAcc	A9B6
OpenPicture	A8F3
OpenPoly	A8CB
OpenPort	A86F
OpenResFile	A997
OpenRF	A00A
PBOpenRF	
OpenRgn	A8DA
OSEventAvail	A030
Pack0	A9E7
(reserved for future use)	
Pack1	A9E8
(reserved for future use)	
Pack2	A9E9
DIBadMount (0)	
DIFormat (6)	
DILoad (2)	
DIUnload (4)	
DIVerify (8)	
DIZero (10)	
Pack3	A9EA
SFGetFile (2)	
SFPGetFile (4)	
SFPPutFile (3)	
SFPutFile (1)	
Pack4	A9EB
Pack5	A9EC
Pack6	A9ED
IUDatePString (14)	
IUDateString (0)	
IUGetIntl (6)	
IUMagIDString (12)	
IUMagString (10)	
IUMetric (4)	
IUSetIntl (8)	
IUTimePString (16)	
IUTimeString (2)	
Pack7	A9EE
NumToString (0)	
StringToNum (1)	

Name	Trap word	Name	Trap word
PackBits	A8CF	ScrollRect	A8EF
PaintArc	A8BF	Secs2Date	A9C6
PaintBehind	A90D	SectRect	A8AA
PaintOne	A90C	SectRgn	A8E4
PaintOval	A8B8	SelectWindow	A91F
PaintPoly	A8C7	SelIText	A97E
PaintRect	A8A2	SendBehind	A921
PaintRgn	A8D3	SetAppBase	A057
PaintRoundRect	A8B1	SetApplBase	
ParamText	A98B	SetApplLimit	A02D
PenMode	A89C	SetClip	A879
PenNormal	A89E	SetCRefCon	A95B
PenPat	A89D	SetCTitle	A95F
PenSize	A89B	SetCtlAction	A96B
PicComment	A8F2	SetCtlValue	A963
PinRect	A94E	SetCursor	A851
PlotIcon	A94B	SetDateTime	A03A
PortSize	A876	SetDItem	A98E
PostEvent	A02F	SetEmptyRgn	A8DD
Pt2Rect	A8AC	SetEOF	A012
PtInRect	A8AD	PBSetEOF	
PtInRgn	A8E8	SetFileInfo	A00D
PtrAndHand	A9EF	PBSetFInfo	
PtrToHand	A9E3	SetFilLock	A041
PtrToXHand	A9E2	PBSetFLock	
PtrZone	A148	SetFilType	A043
PtToAngle	A8C3	PBSetFVers	
PurgeMem	A04D	SetFontLock	A903
PutScrap	A9FE	SetFPos	A044
Random	A861	PBSetFPos	
RDrvrInstall	A04F	SetGrowZone	A04B
(internal use only)		SetHandleSize	A024
Read	A002	SetItem	A947
PBRead		SetIText	A98F
ReadDateTime	A039	SetItmIcon	A940
RealFont	A902	SetItemIcon	
ReallocHandle	A027	SetItmMark	A944
RecoverHandle	A128	SetItemMark	
RectInRgn	A8E9	SetItmStyle	A942
RectRgn	A8DF	SetItemStyle	
ReleaseResource	A9A3	SetMaxCtl	A965
Rename	A00B	SetCtlMax	
PBRename		SetMenuBar	A93C
ResError	A9AF	SetMFlash	A94A
ResrvMem	A040	SetMenuFlash	
RmveResource	A9AD	SetMinCtl	A964
RsrcZoneInit	A996	SetCtlMin	
RstFilLock	A042	SetOrigin	A878
PBRstFLock		SetPBits	A875
SaveOld	A90E	SetPortBits	
ScalePt	A8F8	SetPenState	A899

Name	Trap word	Name	Trap word
SetPort	A873	SubPt	A87F
SetPt	A880	SysBeep	A9C8
SetPtrSize	A020	SysEdit	A9C2
SetRecRgn	A8DE	SystemEdit	
SetRectRgn		SysError	A9C9
SetRect	A8A7	SystemClick	A9B3
SetResAttrs	A9A7	SystemEvent	A9B2
SetResFileAttrs	A9F7	SystemMenu	A9B5
SetResInfo	A9A9	SystemTask	A9B4
SetResLoad	A99B	TEActivate	A9D8
SetResPurge	A993	TECalText	A9D0
SetStdProcs	A8EA	TEClick	A9D4
SetString	A907	TECopy	A9D5
SetTrapAddress	A047	TECut	A9D6
SetVol	A015	TEDeactivate	A9D9
PBSetVol		TEDelete	A9D7
SetWindowPic	A92E	TEDispose	A9CD
SetWRefCon	A918	TEGetText	A9CB
SetWTitle	A91A	TEIdle	A9DA
SetZone	A01B	TEInit	A9CC
ShieldCursor	A855	TEInsert	A9DE
ShowControl	A957	TEKey	A9DC
ShowCursor	A853	TENew	A9D2
ShowHide	A908	TEPaste	A9DB
ShowPen	A897	TEScroll	A9DD
ShowWindow	A915	TESetJust	A9DF
SizeControl	A95C	TESetSelect	A9D1
SizeRsrc	A9A5	TESetText	A9CF
SizeResource		TestControl	A966
SizeWindow	A91D	TEUpdate	A9D3
SlopeFromAngle	A8BC	TextBox	A9CE
SpaceExtra	A88E	TextFace	A888
Status	A005	TextFont	A887
PBStatus		TextMode	A889
StdArc	A8BD	TextSize	A88A
StdBits	A8EB	TextWidth	A886
StdComment	A8F1	TickCount	A975
StdGetPic	A8EE	TrackControl	A968
StdLine	A890	TrackGoAway	A91E
StdOval	A8B6	UnionRect	A8AB
StdPoly	A8C5	UnionRgn	A8E5
StdPutPic	A8F0	UniqueID	A9C1
StdRect	A8A0	UnloadSeg	A9F1
StdRgn	A8D1	UnlodeScrap	A9FA
StdRRect	A8AF	UnloadScrap	
StdText	A882	UnmountVol	A00E
StdTxMeas	A8ED	PBUnmountVol	
StillDown	A973	UnpackBits	A8D0
StopAlert	A986	UpdateResFile	A999
StringWidth	A88C	UprString	A054
StuffHex	A866	UseResFile	A998

Name	Trap word	Name	Trap word
ValidRect	A92A	Write	A003
ValidRgn	A929	PBWrite	
VInstall	A033	WriteParam	A038
VRemove	A034	WriteResource	A9B0
WaitMouseUp	A977	XorRgn	A8E7
		ZeroScrap	A9FC

Trap word	Name	Trap word	Name
A000	Open	A014	GetVol
	PBOpen		PBGetVol
A001	Close	A015	SetVol
	PBClose		PBSetVol
A002	Read	A016	InitQueue
	PBRead	A017	Eject
A003	Write		PBEject
	PBWrite	A018	GetFPos
A004	Control		PBGetFPos
	PBControl	A019	InitZone
A005	Status	A11A	GetZone
	PBStatus	A01B	SetZone
A006	KillIO	A01C	FreeMem
	PBKillIO	A11D	MaxMem
A007	GetVolInfo	A11E	NewPtr
	PBGetVInfo	A01F	DisposPtr
A008	Create	A020	SetPtrSize
	PBCreate	A021	GetPtrSize
A009	Delete	A122	NewHandle
	PBDelete	A023	DisposHandle
A00A	OpenRF	A024	SetHandleSize
	PBOpenRF	A025	GetHandleSize
A00B	Rename	A126	HandleZone
	PBRename	A027	ReallocHandle
A00C	GetFileInfo	A128	RecoverHandle
	PBGetInfo	A029	HLock
A00D	SetFileInfo	A02A	HUnlock
	PBSetFInfo	A02B	EmptyHandle
A00E	UnmountVol	A02C	InitApplZone
	PBUnmountVol	A02D	SetApplLimit
A00F	MountVol	A02E	BlockMove
	PBMountVol	A02F	PostEvent
A010	Allocate	A030	OSEventAvail
	PBAllocate	A031	GetOSEvent
A011	GetEOF	A032	FlushEvents
	PBGetEOF	A033	VInstall
A012	SetEOF	A034	VRemove
	PBSetEOF	A035	Offline
A013	FlushVol		PBOffline
	PBFlushVol	A036	MoreMasters

Trap word	Name
A038	WriteParam
A039	ReadDateTime
A03A	SetDateTime
A03B	Delay
A03C	CmpString
	EqualString
A03D	DrvrInstall
	(internal use only)
A03E	DrvrRemove
	(internal use only)
A03F	InitUtil
A040	ResrvMem
A041	SetFilLock
	PBSetFLock
A042	RstFilLock
	PBRstFLock
A043	SetFilType
	PBSetFVers
A044	SetFPos
	PBSetFPos
A045	FlushFile
	PBFlushFile
A146	GetTrapAddress
A047	SetTrapAddress
A148	PtrZone
A049	HPurge
A04A	HNoPurge
A04B	SetGrowZone
A04C	CompactMem
A04D	PurgeMem
A04E	AddDrive
	(internal use only)
A04F	RDrvrInstall
	(internal use only)
A850	InitCursor
A851	SetCursor
A852	HideCursor
A853	ShowCursor
A054	UprString
A855	ShieldCursor
A856	ObscureCursor
A057	SetAppBase
	SetApplBase
A858	BitAnd
A859	BitXor
A85A	BitNot
A85B	BitOr
A85C	BitShift
A85D	BitTst
A85E	BitSet
A85F	BitClr
A861	Random
A862	ForeColor
A863	BackColor
A864	ColorBit
A865	GetPixel
A866	StuffHex
A867	LongMul
A868	FixMul
A869	FixRatio
A86A	HiWord
A86B	LoWord
A86C	FixRound
A86D	InitPort
A86E	InitGraf
A86F	OpenPort
A870	LocalToGlobal
A871	GlobalToLocal
A872	GrafDevice
A873	SetPort
A874	GetPort
A875	SetPBits
	SetPortBits
A876	PortSize
A877	MovePortTo
A878	SetOrigin
A879	SetClip
A87A	GetClip
A87B	ClipRect
A87C	BackPat
A87D	ClosePort
A87E	AddPt
A87F	SubPt
A880	SetPt
A881	EqualPt
A882	StdText
A883	DrawChar
A884	DrawString
A885	DrawText
A886	TextWidth
A887	TextFont
A888	TextFace
A889	TextMode
A88A	TextSize
A88B	GetFontInfo
A88C	StringWidth
A88D	CharWidth
A88E	SpaceExtra
A890	StdLine
A891	LineTo
A892	Line
A893	MoveTo

Trap word	Name	Trap word	Name
A894	Move	A8C8	ErasePoly
A896	HidePen	A8C9	InvertPoly
A897	ShowPen	A8CA	FillPoly
A898	GetPenState	A8CB	OpenPoly
A899	SetPenState	A8CC	ClosePgon
A89A	GetPen		ClosePoly
A89B	PenSize	A8CD	KillPoly
A89C	PenMode	A8CE	OffsetPoly
A89D	PenPat	A8CF	PackBits
A89E	PenNormal	A8D0	UnpackBits
A8A0	StdRect	A8D1	StdRgn
A8A1	FrameRect	A8D2	FrameRgn
A8A2	PaintRect	A8D3	PaintRgn
A8A3	EraseRect	A8D4	EraseRgn
A8A4	InverRect	A8D5	InverRgn
	InvertRect		InvertRgn
A8A5	FillRect	A8D6	FillRgn
A8A6	EqualRect	A8D8	NewRgn
A8A7	SetRect	A8D9	DisposRgn
A8A8	OffsetRect		DisposeRgn
A8A9	InsetRect	A8DA	OpenRgn
A8AA	SectRect	A8DB	CloseRgn
A8AB	UnionRect	A8DC	CopyRgn
A8AC	Pt2Rect	A8DD	SetEmptyRgn
A8AD	PtInRect	A8DE	SetRecRgn
A8AE	EmptyRect	A8DF	SetRectRgn
A8AF	StdRRect		RectRgn
A8B0	FrameRoundRect	A8E0	OfsetRgn
A8B1	PaintRoundRect		OffsetRgn
A8B2	EraseRoundRect	A8E1	InsetRgn
A8B3	InverRoundRect	A8E2	EmptyRgn
	InvertRoundRect	A8E3	EqualRgn
A8B4	FillRoundRect	A8E4	SectRgn
A8B6	StdOval	A8E5	UnionRgn
A8B7	FrameOval	A8E6	DiffRgn
A8B8	PaintOval	A8E7	XorRgn
A8B9	EraseOval	A8E8	PtInRgn
A8BA	InvertOval	A8E9	RectInRgn
A8BB	FillOval	A8EA	SetStdProcs
A8BC	SlopeFromAngle	A8EB	StdBits
A8BD	StdArc	A8EC	CopyBits
A8BE	FrameArc	A8ED	StdTxMeas
A8BF	PaintArc	A8EE	StdGetPic
A8C0	EraseArc	A8EF	ScrollRect
A8C1	InvertArc	A8F0	StdPutPic
A8C2	FillArc	A8F1	StdComment
A8C3	PtToAngle	A8F2	PicComment
A8C4	AngleFromSlope	A8F3	OpenPicture
A8C5	StdPoly	A8F4	ClosePicture
A8C6	FramePoly	A8F5	KillPicture
A8C7	PaintPoly	A8F6	DrawPicture

Trap word	Name	Trap word	Name
A8F8	ScalePt	A929	ValidRgn
A8F9	MapPt	A92A	ValidRect
A8FA	MapRect	A92B	GrowWindow
A8FB	MapRgn	A92C	FindWindow
A8FC	MapPoly	A92D	CloseWindow
A8FE	InitFonts	A92E	SetWindowPic
A8FF	GetFName	A92F	GetWindowPic
	GetFontName	A930	InitMenus
A900	GetFNum	A931	NewMenu
A901	FMSwapFont	A932	DisposMenu
A902	RealFont		DisposeMenu
A903	SetFontLock	A933	AppendMenu
A904	DrawGrowIcon	A934	ClearMenuBar
A905	DragGrayRgn	A935	InsertMenu
A906	NewString	A936	DeleteMenu
A907	SetString	A937	DrawMenuBar
A908	ShowHide	A938	HiliteMenu
A909	CalcVis	A939	EnableItem
A90A	CalcVBehind	A93A	DisableItem
	CalcVisBehind	A93B	GetMenuBar
A90B	ClipAbove	A93C	SetMenuBar
A90C	PaintOne	A93D	MenuSelect
A90D	PaintBehind	A93E	MenuKey
A90E	SaveOld	A93F	GetItmIcon
A90F	DrawNew		GetItemIcon
A910	GetWMgrPort	A940	SetItmIcon
A911	CheckUpdate		SetItemIcon
A912	InitWindows	A941	GetItmStyle
A913	NewWindow		GetItemStyle
A914	DisposWindow	A942	SetItmStyle
	DisposeWindow		SetItemStyle
A915	ShowWindow	A943	GetItmMark
A916	HideWindow		GetItemMark
A917	GetWRefCon	A944	SetItmMark
A918	SetWRefCon		SetItemMark
A919	GetWTitle	A945	CheckItem
A91A	SetWTitle	A946	GetItem
A91B	MoveWindow	A947	SetItem
A91C	HiliteWindow	A948	CalcMenuSize
A91D	SizeWindow	A949	GetMHandle
A91E	TrackGoAway	A94A	SetMFlash
A91F	SelectWindow		SetMenuFlash
A920	BringToFront	A94B	PlotIcon
A921	SendBehind	A94C	FlashMenuBar
A922	BeginUpdate	A94D	AddResMenu
A923	EndUpdate	A94E	PinRect
A924	FrontWindow	A94F	DeltaPoint
A925	DragWindow	A950	CountMItems
A926	DragTheRgn	A951	InsertResMenu
A927	InvalRgn	A954	NewControl
A928	InvalRect		

Trap word	Name	Trap word	Name
A955	DisposControl	A986	StopAlert
	DisposeControl	A987	NoteAlert
A956	KillControls	A988	CautionAlert
A957	ShowControl	A989	CouldAlert
A958	HideControl	A98A	FreeAlert
A959	MoveControl	A98B	ParamText
A95A	GetCRefCon	A98C	ErrorSound
A95B	SetCRefCon	A98D	GetDItem
A95C	SizeControl	A98E	SetDItem
A95D	HiliteControl	A98F	SetIText
A95E	GetCTitle	A990	GetIText
A95F	SetCTitle	A991	ModalDialog
A960	GetCtlValue	A992	DetachResource
A961	GetMinCtl	A993	SetResPurge
	GetCtlMin	A994	CurResFile
A962	GetMaxCtl	A995	InitResources
	GetCtlMax	A996	RsrcZoneInit
A963	SetCtlValue	A997	OpenResFile
A964	SetMinCtl	A998	UseResFile
	SetCtlMin	A999	UpdateResFile
A965	SetMaxCtl	A99A	CloseResFile
	SetCtlMax	A99B	SetResLoad
A966	TestControl	A99C	CountResources
A967	DragControl	A99D	GetIndResource
A968	TrackControl	A99E	CountTypes
A969	DrawControls	A99F	GetIndType
A96A	GetCtlAction	A9A0	GetResource
A96B	SetCtlAction	A9A1	GetNamedResource
A96C	FindControl	A9A2	LoadResource
A96E	Dequeue	A9A3	ReleaseResource
A96F	Enqueue	A9A4	HomeResFile
A970	GetNextEvent	A9A5	SizeRsrc
A971	EventAvail		SizeResource
A972	GetMouse	A9A6	GetResAttrs
A973	StillDown	A9A7	SetResAttrs
A974	Button	A9A8	GetResInfo
A975	TickCount	A9A9	SetResInfo
A976	GetKeys	A9AA	ChangedResource
A977	WaitMouseUp	A9AB	AddResource
A979	CouldDialog	A9AD	RmveResource
A97A	FreeDialog	A9AF	ResError
A97B	InitDialogs	A9B0	WriteResource
A97C	GetNewDialog	A9B1	CreateResFile
A97D	NewDialog	A9B2	SystemEvent
A97E	SelIText	A9B3	SystemClick
A97F	IsDialogEvent	A9B4	SystemTask
A980	DialogSelect	A9B5	SystemMenu
A981	DrawDialog	A9B6	OpenDeskAcc
A982	CloseDialog	A9B7	CloseDeskAcc
A983	DisposDialog	A9B8	GetPattern
A985	Alert	A9B9	GetCursor

Appendices

Trap word	Name
A9BA	GetString
A9BB	GetIcon
A9BC	GetPicture
A9BD	GetNewWindow
A9BE	GetNewControl
A9BF	GetRMenu
	GetMenu
A9C0	GetNewMBar
A9C1	UniqueID
A9C2	SysEdit
	SystemEdit
A9C6	Secs2Date
A9C7	Date2Secs
A9C8	SysBeep
A9C9	SysError
A9CB	TEGetText
A9CC	TEInit
A9CD	TEDispose
A9CE	TextBox
A9CF	TESetText
A9D0	TECalText
A9D1	TESetSelect
A9D2	TENew
A9D3	TEUpdate
A9D4	TEClick
A9D5	TECopy
A9D6	TECut
A9D7	TEDelete
A9D8	TEActivate
A9D9	TEDeactivate
A9DA	TEIdle
A9DB	TEPaste
A9DC	TEKey
A9DD	TEScroll
A9DE	TEInsert
A9DF	TESetJust
A9E0	Munger
A9E1	HandToHand
A9E2	PtrToXHand
A9E3	PtrToHand
A9E4	HandAndHand
A9E5	InitPack
A9E6	InitAllPacks
A9E7	Pack0
	(reserved for future use)
A9E8	Pack1
	(reserved for future use)
A9E9	Pack2
	DIBadMount (0)
	DILoad (2)
	DIUnload (4)
	DIFormat (6)
	DIVerify (8)
	DIZero (10)
A9EA	Pack3
	SFPutFile (1)
	SFGetFile (2)
	SFPPutFile (3)
	SFPGetFile (4)
A9EB	Pack4
	(synonym: FP68K)
A9EC	Pack5
	(synonym: Elems68K)
A9ED	Pack6
	IUDateString (0)
	IUTimeString (2)
	IUMetric (4)
	IUDGetIntl (6)
	IUSetIntl (8)
	IUMagString (10)
	IUMagIDString (12)
	IUDatePString (14)
	IUTimePString (16)
A9EE	Pack7
	NumToString (0)
	StringToNum (1)
A9EF	PtrAndHand
A9F0	LoadSeg
A9F1	UnloadSeg
A9F2	Launch
A9F3	Chain
A9F4	ExitToShell
A9F5	GetAppParms
A9F6	GetResFileAttrs
A9F7	SetResFileAttrs
A9F9	InfoScrap
A9FA	UnlodeScrap
	UnloadScrap
A9FB	LodeScrap
	LoadScrap
A9FC	ZeroScrap
A9FD	GetScrap
A9FE	PutScrap

APPENDIX D: GLOBAL VARIABLES

This appendix gives an alphabetical list of all system global variables described in *Inside Macintosh*, along with their locations in memory.

Name	Location	Contents
ABusVars	$2D8	Pointer to AppleTalk variables
ACount	$A9A	Stage number (0 through 3) of last alert (word)
ANumber	$A98	Resource ID of last alert (word)
ApFontID	$984	Font number of application font (word)
ApplLimit	$130	Application heap limit
ApplScratch	$A78	12-byte scratch area reserved for use by applications
ApplZone	$2AA	Address of application heap zone
AppParmHandle	$AEC	Handle to Finder information
BufPtr	$10C	Address of end of jump table
BufTgDate	$304	File tags buffer: date and time of last modification (long)
BufTgFBkNum	$302	File tags buffer: logical block number (word)
BufTgFFlg	$300	File tags buffer: flags (word: bit 1=1 if resource fork)
BufTgFNum	$2FC	File tags buffer: file number (long)
CaretTime	$2F4	Caret-blink interval in ticks (long)
CrsrThresh	$8EC	Mouse-scaling threshold (word)
CurActivate	$A64	Pointer to window to receive activate event
CurApName	$910	Name of current application (length byte followed by up to 31 characters)
CurApRefNum	$900	Reference number of current application's resource file (word)
CurDeactive	$A68	Pointer to window to receive deactivate event
CurJTOffset	$934	Offset to jump table from location pointed to by A5 (word)
CurMap	$A5A	Reference number of current resource file (word)
CurPageOption	$936	Sound/screen buffer configuration passed to Chain or Launch (word)
CurPitch	$280	Value of count in square-wave synthesizer buffer (word)
CurrentA5	$904	Address of boundary between application globals and application parameters
CurStackBase	$908	Address of base of stack; start of application globals
DABeeper	$A9C	Address of current sound procedure
DAStrings	$AA0	Handles to ParamText strings (16 bytes)

Name	Location	Contents
DefltStack	$322	Default space allotment for stack (long)
DefVCBPtr	$352	Pointer to default volume control block
DeskHook	$A6C	Address of procedure for painting desktop or responding to clicks on desktop
DeskPattern	$A3C	Pattern with which desktop is painted (8 bytes)
DlgFont	$AFA	Font number for dialogs and alerts (word)
DoubleTime	$2F0	Double-click interval in ticks (long)
DragHook	$9F6	Address of procedure to execute during TrackGoAway, DragWindow, GrowWindow, DragGrayRgn, TrackControl, and DragControl
DragPattern	$A34	Pattern of dragged region's outline (8 bytes)
DrvQHdr	$308	Drive queue header (10 bytes)
DSAlertRect	$3F8	Rectangle enclosing system error alert (8 bytes)
DSAlertTab	$2BA	Pointer to system error alert table in use
DSErrCode	$AF0	Current system error ID (word)
EventQueue	$14A	Event queue header (10 bytes)
ExtStsDT	$2BE	External/status interrupt vector table (16 bytes)
FCBSPtr	$34E	Pointer to file-control-block buffer
FinderName	$2E0	Name of the Finder (length byte followed by up to 15 characters)
FScaleDisable	$A63	Nonzero to disable font scaling (byte)
FSQHdr	$360	File I/O queue header (10 bytes)
GhostWindow	$A84	Pointer to window never to be considered frontmost
GrayRgn	$9EE	Handle to region drawn as desktop
GZRootHnd	$328	Handle to relocatable block not to be moved by grow zone function
HeapEnd	$114	Address of end of application heap zone
JFetch	$8F4	Jump vector for Fetch function
JIODone	$8FC	Jump vector for IODone function
JournalFlag	$8DE	Journaling mode (word)
JournalRef	$8E8	Reference number of journaling device driver (word)
JStash	$8F8	Jump vector for Stash function
KeyRepThresh	$190	Auto-key rate (word)
KeyThresh	$18E	Auto-key threshold (word)
Lo3Bytes	$31A	$00FFFFFF
Lvl1DT	$192	Level-1 secondary interrupt vector table (32 bytes)

Name	Location	Contents
Lvl2DT	$1B2	Level-2 secondary interrupt vector table (32 bytes)
MBarEnable	$A20	Unique menu ID for active desk accessory, when menu bar belongs to the accessory (word)
MBarHook	$A2C	Address of routine called by MenuSelect before menu is drawn
MemTop	$108	Address of end of RAM (on Macintosh XL, end of RAM available to applications)
MenuFlash	$A24	Count for duration of menu item blinking (word)
MenuHook	$A30	Address of routine called during MenuSelect
MenuList	$A1C	Handle to current menu list
MinStack	$31E	Minimum space allotment for stack (long)
MinusOne	$A06	$FFFFFFFF
OldContent	$9EA	Handle to saved content region
OldStructure	$9E6	Handle to saved structure region
OneOne	$A02	$00010001
PaintWhite	$9DC	Flag for whether to paint window white before update event (word)
PortBUse	$291	Current availability of serial port B (byte)
PrintErr	$944	Result code from last Printing Manager routine (word)
RAMBase	$2B2	Trap dispatch table's base address for routines in RAM
ResErr	$A60	Current value of ResError (word)
ResErrProc	$AF2	Address of resource error procedure
ResLoad	$A5E	Current SetResLoad state (word)
ResumeProc	$A8C	Address of resume procedure
RndSeed	$156	Random number seed (long)
ROMBase	$2AE	Base address of ROM
ROMFont0	$980	Handle to font record for system font
SaveUpdate	$9DA	Flag for whether to generate update events (word)
SaveVisRgn	$9F2	Handle to saved visRgn
SCCRd	$1D8	SCC read base address
SCCWr	$1DC	SCC write base address
ScrapCount	$968	Count changed by ZeroScrap (word)
ScrapHandle	$964	Handle to desk scrap in memory
ScrapName	$96C	Pointer to scrap file name (preceded by length byte)
ScrapSize	$960	Size in bytes of desk scrap (long)
ScrapState	$96A	Tells where desk scrap is (word)

Name	Location	Contents
Scratch8	$9FA	8-byte scratch area
Scratch20	$1E4	20-byte scratch area
ScrDmpEnb	$2F8	0 if GetNextEvent shouldn't process Command-Shift-number combinations (byte)
ScrHRes	$104	Pixels per inch horizontally (word)
ScrnBase	$824	Address of main screen buffer
ScrVRes	$102	Pixels per inch vertically (word)
SdVolume	$260	Current speaker volume (byte: low-order three bits only)
SEvtEnb	$15C	0 if SystemEvent should return FALSE (byte)
SFSaveDisk	$214	Negative of volume reference number used by Standard File Package (word)
SoundBase	$266	Pointer to free-form synthesizer buffer
SoundLevel	$27F	Amplitude in 740-byte buffer (byte)
SoundPtr	$262	Pointer to four-tone record
SPAlarm	$200	Alarm setting (long)
SPATalkA	$1F9	AppleTalk node ID hint for modem port (byte)
SPATalkB	$1FA	AppleTalk node ID hint for printer port (byte)
SPClikCaret	$209	Double-click and caret-blink times (byte)
SPConfig	$1FB	Use types for serial ports (byte)
SPFont	$204	Application font number minus 1 (word)
SPKbd	$206	Auto-key threshold and rate (byte)
SPMisc2	$20B	Mouse scaling, system startup disk, menu blink (byte)
SPPortA	$1FC	Modem port configuration (word)
SPPortB	$1FE	Printer port configuration (word)
SPPrint	$207	Printer connection (byte)
SPValid	$1F8	Validity status (byte)
SPVolCtl	$208	Speaker volume setting in parameter RAM (byte)
SysEvtMask	$144	System event mask (word)
SysMap	$A58	Reference number of system resource file (word)
SysMapHndl	$A54	Handle to map of system resource file
SysParam	$1F8	Low-memory copy of parameter RAM (20 bytes)
SysResName	$AD8	Name of system resource file (length byte followed by up to 19 characters)
SysZone	$2A6	Address of system heap zone
TEDoText	$A70	Address of TextEdit multi-purpose routine

Name	Location	Contents
TERecal	$A74	Address of routine to recalculate line starts for TextEdit
TEScrpHandle	$AB4	Handle to TextEdit scrap
TEScrpLength	$AB0	Size in bytes of TextEdit scrap (long)
TheMenu	$A26	Menu ID of currently highlighted menu (word)
TheZone	$118	Address of current heap zone
Ticks	$16A	Current number of ticks since system startup (long)
Time	$20C	Seconds since midnight, January 1, 1904 (long)
ToExtFS	$3F2	Pointer to external file system
ToolScratch	$9CE	8-byte scratch area
TopMapHndl	$A50	Handle to resource map of most recently opened resource file
UTableBase	$11C	Base address of unit table
VBLQueue	$160	Vertical retrace queue header (10 bytes)
VCBQHdr	$356	Volume-control-block queue header (10 bytes)
VIA	$1DA	VIA base address
WindowList	$9D6	Pointer to first window in window list; 0 if using events but not windows
WMgrPort	$9DE	Pointer to Window Manager port

GLOSSARY

access path: A description of the route that the File Manager follows to access a file; created when a file is opened.

access path buffer: Memory used by the File Manager to transfer data between an application and a file.

action procedure: A procedure, used by the Control Manager function TrackControl, that defines an action to be performed repeatedly for as long as the mouse button is held down.

activate event: An event generated by the Window Manager when a window changes from active to inactive or vice versa.

active control: A control that will respond to the user's actions with the mouse.

active window: The frontmost window on the desktop.

address mark: In a sector, information that's used internally by the Disk Driver, including information it uses to determine the position of the sector on the disk.

ALAP: See **AppleTalk Link Access Protocol.**

ALAP frame: A packet of data transmitted and received by ALAP.

ALAP protocol type: An identifier used to match particular kinds of packets with a particular protocol handler.

alert: A warning or report of an error, in the form of an alert box, sound from the Macintosh's speaker, or both.

alert box: A box that appears on the screen to give a warning or report an error during a Macintosh application.

alert template: A resource that contains information from which the Dialog Manager can create an alert.

alert window: The window in which an alert box is displayed.

alias: A different name for the same entity.

allocate: To reserve an area of memory for use.

allocation block: Volume space composed of an integral number of logical blocks.

amplitude: The maximum vertical distance of a periodic wave from the horizontal line about which the wave oscillates.

AppleTalk address: A socket's number and its node ID number.

AppleTalk Link Access Protocol (ALAP): The lowest-level protocol in the AppleTalk architecture, managing node-to-node delivery of frames on a single AppleTalk network.

AppleTalk Manager: An interface to a pair of RAM device drivers that enable programs to send and receive information via an AppleTalk network.

AppleTalk Transaction Protocol (ATP): An AppleTalk protocol that's a DDP client. It allows one ATP client to request another ATP client to perform some activity and report the activity's result as a response to the requesting socket with guaranteed delivery.

application font: The font your application will use unless you specify otherwise—Geneva, by default.

application heap: The portion of the heap available to the running application program and the Toolbox.

application heap limit: The boundary between the space available for the application heap and the space available for the stack.

application heap zone: The heap zone initially provided by the Memory Manager for use by the application program and the Toolbox; initially equivalent to the application heap, but may be subdivided into two or more independent heap zones.

application parameters: Thirty-two bytes of memory, located above the application globals, reserved for system use. The first application parameter is the address of the first QuickDraw global variable.

application space: Memory that's available for dynamic allocation by applications.

application window: A window created as the result of something done by the application, either directly or indirectly (as through the Dialog Manager).

ascent: The vertical distance from a font's base line to its ascent line.

ascent line: A horizontal line that coincides with the tops of the tallest characters in a font.

asynchronous communication: A method of data transmission where the receiving and sending devices don't share a common timer, and no timing data is transmitted.

asynchronous execution: After calling a routine asynchronously, an application is free to perform other tasks until the routine is completed.

at-least-once transaction: An ATP transaction in which the requested operation is performed at least once, and possibly several times.

ATP: See **AppleTalk Transaction Protocol.**

auto-key event: An event generated repeatedly when the user presses and holds down a character key on the keyboard or keypad.

auto-key rate: The rate at which a character key repeats after it's begun to do so.

auto-key threshold: The length of time a character key must be held down before it begins to repeat.

background procedure: A procedure passed to the Printing Manager to be run during idle times in the printing process.

base line: A horizontal line that coincides with the bottom of each character in a font, excluding descenders (such as the tail of a "p").

baud rate: The measure of the total number of bits sent over a transmission line per second.

Binary-Decimal Conversion Package: A Macintosh package for converting integers to decimal strings and vice versa.

bit image: A collection of bits in memory that have a rectilinear representation. The screen is a visible bit image.

bit map: A set of bits that represent the position and state of a corresponding set of items; in QuickDraw, a pointer to a bit image, the row width of that image, and its boundary rectangle.

block: A group regarded as a unit; usually refers to data or memory in which data is stored. See **allocation block** and **memory block**.

block contents: The area that's available for use in a memory block.

block device: A device that reads and writes blocks of bytes at a time. It can read or write any accessible block on demand.

block header: The internal "housekeeping" information maintained by the Memory Manager at the beginning of each block in a heap zone.

block map: Same as **volume allocation block map**.

boundary rectangle: A rectangle, defined as part of a QuickDraw bit map, that encloses the active area of the bit image and imposes a coordinate system on it. Its top left corner is always aligned around the first bit in the bit image.

break: The condition resulting when a device maintains its transmission line in the space state for at least one frame.

bridge: An intelligent link between two or more AppleTalk networks.

broadcast service: An ALAP service in which a frame is sent to all nodes on an AppleTalk network.

bundle: A resource that maps local IDs of resources to their actual resource IDs; used to provide mappings for file references and icon lists needed by the Finder.

button: A standard Macintosh control that causes some immediate or continuous action when clicked or pressed with the mouse. See also **radio button**.

caret: A generic term meaning a symbol that indicates where something should be inserted in text. The specific symbol used is a vertical bar (|).

caret-blink time: The interval between blinks of the caret that marks an insertion point.

character code: An integer representing the character that a key or combination of keys on the keyboard or keypad stands for.

character device: A device that reads or writes a stream of characters, one at a time. It can neither skip characters nor go back to a previous character.

character image: An arrangement of bits that defines a character in a font.

character key: A key that generates a keyboard event when pressed; any key except Shift, Caps Lock, Command, or Option.

character offset: The horizontal separation between a character rectangle and a font rectangle.

character origin: The point on a base line used as a reference location for drawing a character.

character position: An index into an array containing text, starting at 0 for the first character.

character rectangle: A rectangle enclosing an entire character image. Its sides are defined by the image width and the font height.

character style: A set of stylistic variations, such as bold, italic, and underline. The empty set indicates plain text (no stylistic variations).

character width: The distance to move the pen from one character's origin to the next character's origin.

check box: A standard Macintosh control that displays a setting, either checked (on) or unchecked (off). Clicking inside a check box reverses its setting.

clipping: Limiting drawing to within the bounds of a particular area.

clipping region: Same as clipRgn.

clipRgn: The region to which an application limits drawing in a grafPort.

clock chip: A special chip in which are stored parameter RAM and the current setting for the date and time. This chip is powered by a battery when the system is off, thus preserving the information.

close routine: The part of a device driver's code that implements Device Manager Close calls.

closed driver: A device driver that cannot be read from or written to.

closed file: A file without an access path. Closed files cannot be read from or written to.

compaction: The process of moving allocated blocks within a heap zone in order to collect the free space into a single block.

completion routine: Any application-defined code to be executed when an asynchronous call to a routine is completed.

content region: The area of a window that the application draws in.

control: An object in a window on the Macintosh screen with which the user, using the mouse, can cause instant action with visible results or change settings to modify a future action.

control definition function: A function called by the Control Manager when it needs to perform type-dependent operations on a particular type of control, such as drawing the control.

control definition ID: A number passed to control-creation routines to indicate the type of control. It consists of the control definition function's resource ID and a variation code.

control information: Information transmitted by an application to a device driver. It may select modes of operation, start or stop processes, enable buffers, choose protocols, and so on.

control list: A list of all the controls associated with a given window.

Control Manager: The part of the Toolbox that provides routines for creating and manipulating controls (such as buttons, check boxes, and scroll bars).

control record: The internal representation of a control, where the Control Manager stores all the information it needs for its operations on that control.

control routine: The part of a device driver's code that implements Device Manager Control and KillIO calls.

control template: A resource that contains information from which the Control Manager can create a control.

coordinate plane: A two-dimensional grid. In QuickDraw, the grid coordinates are integers ranging from –32767 to 32767, and all grid lines are infinitely thin.

current heap zone: The heap zone currently under attention, to which most Memory Manager operations implicitly apply.

current resource file: The last resource file opened, unless you specify otherwise with a Resource Manager routine.

cursor: A 16-by-16 bit image that appears on the screen and is controlled by the mouse; called the "pointer" in Macintosh user manuals.

cursor level: A value, initialized by InitCursor, that keeps track of the number of times the cursor has been hidden.

data bits: Data communications bits that encode transmitted characters.

data buffer: Heap space containing information to be written to a file or device driver from an application, or read from a file or device driver to an application.

data fork: The part of a file that contains data accessed via the File Manager.

data mark: In a sector, information that primarily contains data from an application.

datagram: A packet of data transmitted by DDP.

Datagram Delivery Protocol (DDP): An AppleTalk protocol that's an ALAP client, managing socket-to-socket delivery of datagrams over AppleTalk internets.

date/time record: An alternate representation of the date and time (which is stored on the clock chip in seconds since midnight, January 1, 1904).

DDP: See **Datagram Delivery Protocol.**

default button: In an alert box or modal dialog, the button whose effect will occur if the user presses Return or Enter. In an alert box, it's boldly outlined; in a modal dialog, it's boldly outlined or the OK button.

default volume: A volume that will receive I/O during a File Manager routine call, whenever no other volume is specified.

dereference: To refer to a block by its master pointer instead of its handle.

descent: The vertical distance from a font's base line to its descent line.

descent line: A horizontal line that coincides with the bottoms of the characters in a font that extend furthest below the base line.

desk accessory: A "mini-application", implemented as a device driver, that can be run at the same time as a Macintosh application.

Desk Manager: The part of the Toolbox that supports the use of desk accessories from an application.

desk scrap: The place where data is stored when it's cut (or copied) and pasted among applications and desk accessories.

desktop: The screen as a surface for doing work on the Macintosh.

Desktop file: A resource file in which the Finder stores the version data, bundle, icons, and file references for each application on the volume.

destination rectangle: In TextEdit, the rectangle in which the text is drawn.

device: A part of the Macintosh, or a piece of external equipment, that can transfer information into or out of the Macintosh.

device control entry: A 40-byte relocatable block of heap space that tells the Device Manager the location of a driver's routines, the location of a driver's I/O queue, and other information.

device driver: A program that controls the exchange of information between an application and a device.

device driver event: An event generated by one of the Macintosh's device drivers.

Device Manager: The part of the Operating System that supports device I/O.

dial: A control with a moving indicator that displays a quantitative setting or value. Depending on the type of dial, the user may be able to change the setting by dragging the indicator with the mouse.

dialog: Same as **dialog box.**

dialog box: A box that a Macintosh application displays to request information it needs to complete a command, or to report that it's waiting for a process to complete.

Dialog Manager: The part of the Toolbox that provides routines for implementing dialogs and alerts.

dialog record: The internal representation of a dialog, where the Dialog Manager stores all the information it needs for its operations on that dialog.

dialog template: A resource that contains information from which the Dialog Manager can create a dialog.

dialog window: The window in which a dialog box is displayed.

dimmed: Drawn in gray rather than black

disabled: A disabled menu item or menu is one that cannot be chosen; the menu item or menu title appears dimmed. A disabled item in a dialog or alert box has no effect when clicked.

Disk Driver: The device driver that controls data storage and retrieval on 3 1/2-inch disks.

Disk Initialization Package: A Macintosh package for initializing and naming new disks; called by the Standard File Package.

disk-inserted event: An event generated when the user inserts a disk in a disk drive or takes any other action that requires a volume to be mounted.

display rectangle: A rectangle that determines where an item is displayed within a dialog or alert box.

document window: The standard Macintosh window for presenting a document.

double-click time: The greatest interval between a mouse-up and mouse-down event that would qualify two mouse clicks as a double-click.

draft printing: Printing a document immediately as it's drawn in the printing grafPort.

drag region: A region in a window frame. Dragging inside this region moves the window to a new location and makes it the active window unless the Command key was down.

drive number: A number used to identify a disk drive. The internal drive is number 1, the external drive is number 2, and any additional drives will have larger numbers.

drive queue: A list of disk drives connected to the Macintosh.

driver name: A sequence of up to 255 printing characters used to refer to an open device driver. Driver names always begin with a period (.).

driver I/O queue: A queue containing the parameter blocks of all I/O requests for one device driver.

driver reference number: A number from –1 to –32 that uniquely identifies an individual device driver.

edit record: A complete editing environment in TextEdit, which includes the text to be edited, the grafPort and rectangle in which to display the text, the arrangement of the text within the rectangle, and other editing and display information.

empty handle: A handle that points to a NIL master pointer, signifying that the underlying relocatable block has been purged.

empty shape: A shape that contains no bits, such as one defined by only a single point.

end-of-file: See **logical end-of-file** or **physical end-of-file**.

entity name: An identifier for an entity, of the form object:type@zone.

event: A notification to an application of some occurrence that the application may want to respond to.

event code: An integer representing a particular type of event.

Event Manager: See **Toolbox Event Manager** or **Operating System Event Manager**.

event mask: A parameter passed to an Event Manager routine to specify which types of events the routine should apply to.

event message: A field of an event record containing information specific to the particular type of event.

event queue: The Operating System Event Manager's list of pending events.

event record: The internal representation of an event, through which your program learns all pertinent information about that event.

exactly-once transaction: An ATP transaction in which the requested operation is performed only once.

exception: An error or abnormal condition detected by the processor in the course of program execution; includes interrupts and traps.

exception vector: One of 64 vectors in low memory that point to the routines that are to get control in the event of an exception.

external reference: A reference to a routine or variable defined in a separate compilation or assembly.

file: A named, ordered sequence of bytes; a principal means by which data is stored and transmitted on the Macintosh.

file control block: A fixed-length data structure, contained in the file-control-block buffer, where information about an access path is stored.

file-control-block buffer: A nonrelocatable block in the system heap that contains one file control block for each access path.

file directory: The part of a volume that contains descriptions and locations of all the files on the volume.

file I/O queue: A queue containing parameter blocks for all I/O requests to the File Manager.

File Manager: The part of the Operating System that supports file I/O.

file name: A sequence of up to 255 printing characters, excluding colons (:), that identifies a file.

file number: A unique number assigned to a file, which the File Manager uses to distinguish it from other files on the volume. A file number specifies the file's entry in a file directory.

file reference: A resource that provides the Finder with file and icon information about an application.

file tags: Information associated with each logical block, designed to allow reconstruction of files on a volume whose directory or other file-access information has been destroyed.

file tags buffer: A location in memory where file tags are read from and written to.

file type: A four-character sequence, specified when a file is created, that identifies the type of file.

Finder information: Information that the Finder provides to an application upon starting it up, telling it which documents to open or print.

fixed-point number: A signed 32-bit quantity containing an integer part in the high-order word and a fractional part in the low-order word.

fixed-width font: A font whose characters all have the same width.

Floating-Point Arithmetic Package: A Macintosh package that supports extended-precision arithmetic according to IEEE Standard 754.

font: The complete set of characters of one typeface.

font characterization table: A table of parameters in a device driver that specifies how best to adapt fonts to that device.

font height: The vertical distance from a font's ascent line to its descent line.

Font Manager: The part of the Toolbox that supports the use of various character fonts for QuickDraw when it draws text.

font number: The number by which you identify a font to QuickDraw or the Font Manager.

font record: A data structure that contains all the information describing a font.

font rectangle: The smallest rectangle enclosing all the character images in a font, if the images were all superimposed over the same character origin.

font size: The size of a font in points; equivalent to the distance between the ascent line of one line of text and the ascent line of the next line of single-spaced text.

fork: One of the two parts of a file; see **data fork** and **resource fork**.

four-tone record: A data structure describing the tones produced by a four-tone synthesizer.

four-tone synthesizer: The part of the Sound Driver used to make simple harmonic tones, with up to four "voices" producing sound simultaneously.

frame: The time elapsed from the start bit to the last stop bit during serial communication.

frame check sequence: A 16-bit value generated by the AppleTalk hardware, used by the receiving node to detect transmission errors.

frame header: Information at the beginning of a packet.

frame pointer: A pointer to the end of the local variables within a routine's stack frame, held in an address register and manipulated with the LINK and UNLK instructions.

frame trailer: Information at the end of an ALAP frame.

framed shape: A shape that's drawn outlined and hollow.

framing error: The condition resulting when a device doesn't receive a stop bit when expected.

free block: A memory block containing space available for allocation.

free-form synthesizer: The part of the Sound Driver used to make complex music and speech.

frequency: The number of cycles per second (also called hertz) at which a wave oscillates.

full-duplex communication: A method of data transmission where two devices transmit data simultaneously.

global coordinate system: The coordinate system based on the top left corner of the bit image being at (0,0).

go-away region: A region in a window frame. Clicking inside this region of the active window makes the window close or disappear.

grafPort: A complete drawing environment, including such elements as a bit map, a subset of it in which to draw, a character font, patterns for drawing and erasing, and other pen characteristics.

grow image: The image pulled around when the user drags inside the grow region; whatever is appropriate to show that the window's size will change.

grow region: A window region, usually within the content region, where dragging changes the size of an active window.

grow zone function: A function supplied by the application program to help the Memory Manager create free space within a heap zone.

handle: A pointer to a master pointer, which designates a relocatable block in the heap by double indirection.

hardware overrun error: The condition that occurs when the SCC's buffer becomes full.

heap: The area of memory in which space is dynamically allocated and released on demand, using the Memory Manager.

heap zone: An area of memory initialized by the Memory Manager for heap allocation.

highlight: To display an object on the screen in a distinctive visual way, such as inverting it.

horizontal blanking interval: The time between the display of the rightmost pixel on one line and the leftmost pixel on the next line.

hotSpot: The point in a cursor that's aligned with the mouse location.

icon: A 32-by-32 bit image that graphically represents an object, concept, or message.

icon list: A resource consisting of a list of icons.

icon number: A digit from 1 to 255 to which the Menu Manager adds 256 to get the resource ID of an icon associated with a menu item.

image width: The width of a character image.

inactive control: A control that won't respond to the user's actions with the mouse. An inactive control is highlighted in some special way, such as dimmed.

inactive window: Any window that isn't the frontmost window on the desktop.

indicator: The moving part of a dial that displays its current setting.

input driver: A device driver that receives serial data via a serial port and transfers it to an application.

insertion point: An empty selection range; the character position where text will be inserted (usually marked with a blinking caret).

interface routine: A routine called from Pascal whose purpose is to trap to a certain Toolbox or Operating System routine.

International Utilities Package: A Macintosh package that gives you access to country-dependent information such as the formats for numbers, currency, dates, and times.

internet: An interconnected group of AppleTalk networks.

internet address: The AppleTalk address and network number of a socket.

interrupt: An exception that's signaled to the processor by a device, to notify the processor of a change in condition of the device, such as the completion of an I/O request.

interrupt handler: A routine that services interrupts.

interrupt priority level: A number identifying the importance of the interrupt. It indicates which device is interrupting, and which interrupt handler should be executed.

interrupt vector: A pointer to an interrupt handler.

invert: To highlight by changing white pixels to black and vice versa.

invisible control: A control that's not drawn in its window.

invisible window: A window that's not drawn in its plane on the desktop.

I/O queue: See **driver I/O queue** or **file I/O queue**.

I/O request: A request for input from or output to a file or device driver; caused by calling a File Manager or Device Manager routine asynchronously.

item: In dialog and alert boxes, a control, icon, picture, or piece of text, each displayed inside its own display rectangle. See also **menu item**.

item list: A list of information about all the items in a dialog or alert box.

item number: The index, starting from 1, of an item in an item list.

IWM: "Integrated Woz Machine"; the custom chip that controls the 3 1/2-inch disk drives.

job dialog: A dialog that sets information about one printing job; associated with the Print command.

journal code: A code passed by a Toolbox Event Manager routine in its Control call to the journaling device driver, to designate which routine is making the Control call.

journaling mechanism: A mechanism that allows you to feed the Toolbox Event Manager events from some source other than the user.

jump table: A table that contains one entry for every routine in an application and is the means by which the loading and unloading of segments is implemented.

justification: The horizontal placement of lines of text relative to the edges of the rectangle in which the text is drawn.

kern: To draw part of a character so that it overlaps an adjacent character.

key code: An integer representing a key on the keyboard or keypad, without reference to the character that the key stands for.

key-down event: An event generated when the user presses a character key on the keyboard or keypad.

key-up event: An event generated when the user releases a character key on the keyboard or keypad.

keyboard configuration: A resource that defines a particular keyboard layout by associating a character code with each key or combination of keys on the keyboard or keypad.

keyboard equivalent: The combination of the Command key and another key, used to invoke a menu item from the keyboard.

keyboard event: An event generated when the user presses, releases, or holds down a character key on the keyboard or keypad; any key-down, key-up, or auto-key event.

leading: The amount of blank vertical space between the descent line of one line of text and the ascent line of the next line of single-spaced text.

ligature: A character that combines two letters.

list separator: The character that separates numbers, as when a list of numbers is entered by the user.

local coordinate system: The coordinate system local to a grafPort, imposed by the boundary rectangle defined in its bit map.

local ID: A number that refers to an icon list or file reference in an application's resource file and is mapped to an actual resource ID by a bundle.

location table: An array of words (one for each character in a font) that specifies the location of each character's image in the font's bit image.

lock: To temporarily prevent a relocatable block from being moved during heap compaction.

lock bit: A bit in the master pointer to a relocatable block that indicates whether the block is currently locked.

locked file: A file whose data cannot be changed.

locked volume: A volume whose data cannot be changed. Volumes can be locked by either a software flag or a hardware setting.

logical block: Volume space composed of 512 consecutive bytes of standard information and an additional number of bytes of information specific to the Disk Driver.

logical end-of-file: The position of one byte past the last byte in a file; equal to the actual number of bytes in the file.

logical size: The number of bytes in a memory block's contents.

magnitude: The vertical distance between any given point on a wave and the horizontal line about which the wave oscillates.

main event loop: In a standard Macintosh application program, a loop that repeatedly calls the Toolbox Event Manager to get events and then responds to them as appropriate.

main segment: The segment containing the main program.

mark: The position of the next byte in a file that will be read or written.

mark state: The state of a transmission line indicating a binary 1.

master directory block: Part of the data structure of a volume; contains the volume information and the volume allocation block map.

master pointer: A single pointer to a relocatable block, maintained by the Memory Manager and updated whenever the block is moved, purged, or reallocated. All handles to a relocatable block refer to it by double indirection through the master pointer.

memory block: An area of contiguous memory within a heap zone.

Memory Manager: The part of the Operating System that dynamically allocates and releases memory space in the heap.

menu: A list of menu items that appears when the user points to a menu title in the menu bar and presses the mouse button. Dragging through the menu and releasing over an enabled menu item chooses that item.

menu bar: The horizontal strip at the top of the Macintosh screen that contains the menu titles of all menus in the menu list.

menu definition procedure: A procedure called by the Menu Manager when it needs to perform type-dependent operations on a particular type of menu, such as drawing the menu.

menu ID: A number in the menu record that identifies the menu.

menu item: A choice in a menu, usually a command to the current application.

menu item number: The index, starting from 1, of a menu item in a menu.

menu list: A list containing menu handles for all menus in the menu bar, along with information on the position of each menu.

Menu Manager: The part of the Toolbox that deals with setting up menus and letting the user choose from them.

menu record: The internal representation of a menu, where the Menu Manager stores all the information it needs for its operations on that menu.

menu title: A word or phrase in the menu bar that designates one menu.

missing symbol: A character to be drawn in case of a request to draw a character that's missing from a particular font.

modal dialog: A dialog that requires the user to respond before doing any other work on the desktop.

modeless dialog: A dialog that allows the user to work elsewhere on the desktop before responding.

modifier key: A key (Shift, Caps Lock, Option, or Command) that generates no keyboard events of its own, but changes the meaning of other keys or mouse actions.

mounted volume: A volume that previously was inserted into a disk drive and had descriptive information read from it by the File Manager.

mouse-down event: An event generated when the user presses the mouse button.

mouse scaling: A feature that causes the cursor to move twice as far during a mouse stroke than it would have otherwise, provided the change in the cursor's position exceeds the mouse-scaling threshold within one tick after the mouse is moved.

mouse-scaling threshold: A number of pixels which, if exceeded by the sum of the horizontal and vertical changes in the cursor position during one tick of mouse movement, causes mouse scaling to occur (if that feature is turned on); normally six pixels.

mouse-up event: An event generated when the user releases the mouse button.

Name-Binding Protocol (NBP): An AppleTalk protocol that's a DDP client, used to convert entity names to their internet socket addresses.

name lookup: An NBP operation that allows clients to obtain the internet addresses of entities from their names.

names directory: The union of all name tables in an internet.

names information socket: The socket in a node used to implement NBP (always socket number 2).

names table: A list of each entity's name and internet address in a node.

NBP: See **Name-Binding Protocol.**

NBP tuple: An entity name and an internet address.

network event: An event generated by the AppleTalk Manager.

network number: An identifier for an AppleTalk network.

network-visible entity: A named socket client on an internet.

newline character: Any character, but usually Return (ASCII code $0D), that indicates the end of a sequence of bytes.

newline mode: A mode of reading data where the end of the data is indicated by a newline character (and not by a specific byte count).

node: A device that's attached to and communicates via an AppleTalk network.

node ID: A number, dynamically assigned, that identifies a node.

nonbreaking space: The character with ASCII code $CA; drawn as a space the same width as a digit, but interpreted as a nonblank character for the purposes of word wraparound and selection.

nonrelocatable block: A block whose location in the heap is fixed and can't be moved during heap compaction.

null event: An event reported when there are no other events to report.

off-line volume: A mounted volume with all but 94 bytes of its descriptive information released.

offset/width table: An array of words that specifies the character offsets and character widths of all characters in a font.

on-line volume: A mounted volume with its volume buffer and descriptive information contained in memory.

open driver: A driver that can be read from and written to.

open file: A file with an access path. Open files can be read from and written to.

open permission: Information about a file that indicates whether the file can be read from, written to, or both.

open routine: The part of a device driver's code that implements Device Manager Open calls.

Operating System: The lowest-level software in the Macintosh. It does basic tasks such as I/O, memory management, and interrupt handling.

Operating System Event Manager: The part of the Operating System that reports hardware-related events such as mouse-button presses and keystrokes.

Operating System Utilities: Operating System routines that perform miscellaneous tasks such as getting the date and time, finding out the user's preferred speaker volume and other preferences, and doing simple string comparison.

output driver: A device driver that receives data via a serial port and transfers it to an application.

overrun error: See **hardware overrun error** and **software overrun error**.

package: A set of routines and data types that's stored as a resource and brought into memory only when needed.

Package Manager: The part of the Toolbox that lets you access Macintosh RAM-based packages.

page rectangle: The rectangle marking the boundaries of a printed page image. The boundary rectangle, portRect, and clipRgn of the printing grafPort are set to this rectangle.

palette: A collection of small symbols, usually enclosed in rectangles, that represent operations and can be selected by the user.

pane: An independently scrollable area of a window, for showing a different part of the same document.

panel: An area of a window that shows a different interpretation of the same part of a document.

paper rectangle: The rectangle marking the boundaries of the physical sheet of paper on which a page is printed.

parameter block: A data structure used to transfer information between applications and certain Operating System routines.

parameter RAM: In the clock chip, 20 bytes where settings such as those made with the Control Panel desk accessory are preserved.

parity bit: A data communications bit used to verify that data bits received by a device match the data bits transmitted by another device.

parity error: The condition resulting when the parity bit received by a device isn't what was expected.

part code: An integer between 1 and 253 that stands for a particular part of a control (possibly the entire control).

path reference number: A number that uniquely identifies an individual access path; assigned when the access path is created.

pattern: An 8-by-8 bit image, used to define a repeating design (such as stripes) or tone (such as gray).

pattern transfer mode: One of eight transfer modes for drawing lines or shapes with a pattern.

period: The time elapsed during one complete cycle of a wave.

phase: Some fraction of a wave cycle (measured from a fixed point on the wave).

physical end-of-file: The position of one byte past the last allocation block of a file; equal to 1 more than the maximum number of bytes the file can contain.

physical size: The actual number of bytes a memory block occupies within its heap zone.

picture: A saved sequence of QuickDraw drawing commands (and, optionally, picture comments) that you can play back later with a single procedure call; also, the image resulting from these commands.

picture comments: Data stored in the definition of a picture that doesn't affect the picture's appearance but may be used to provide additional information about the picture when it's played back.

picture frame: A rectangle, defined as part of a picture, that surrounds the picture and gives a frame of reference for scaling when the picture is played back.

pixel: The visual representation of a bit on the screen (white if the bit is 0, black if it's 1).

plane: The front-to-back position of a window on the desktop.

point: The intersection of a horizontal grid line and a vertical grid line on the coordinate plane, defined by a horizontal and a vertical coordinate; also, a typographical term meaning approximately 1/72 inch.

polygon: A sequence of connected lines, defined by QuickDraw line-drawing commands.

port: See **grafPort**.

portBits: The bit map of a grafPort.

portRect: A rectangle, defined as part of a grafPort, that encloses a subset of the bit map for use by the grafPort.

post: To place an event in the event queue for later processing.

prime routine: The part of a device driver's code that implements Device Manager Read and Write calls.

print record: A record containing all the information needed by the Printing Manager to perform a particular printing job.

Printer Driver: The device driver for the currently installed printer.

printer resource file: A file containing all the resources needed to run the Printing Manager with a particular printer.

printing grafPort: A special grafPort customized for printing instead of drawing on the screen.

Printing Manager: The routines and data types that enable applications to communicate with the Printer Driver to print on any variety of printer via the same interface.

processor priority: Bits 8-10 of the MC68000's status register, indicating which interrupts will be processed and which will be ignored.

proportional font: A font whose characters all have character widths that are proportional to their image width.

protocol: A well-defined set of communications rules.

protocol handler: A software process in a node that recognizes different kinds of frames by their ALAP type and services them.

protocol handler table: A list of the protocol handlers for a node.

purge: To remove a relocatable block from the heap, leaving its master pointer allocated but set to NIL.

purge bit: A bit in the master pointer to a relocatable block that indicates whether the block is currently purgeable.

purge warning procedure: A procedure associated with a particular heap zone that's called whenever a block is purged from that zone.

purgeable block: A relocatable block that can be purged from the heap.

queue: A list of identically structured entries linked together by pointers.

QuickDraw: The part of the Toolbox that performs all graphic operations on the Macintosh screen.

radio button: A standard Macintosh control that displays a setting, either on or off, and is part of a group in which only one button can be on at a time.

RAM: The Macintosh's random access memory, which contains exception vectors, buffers used by hardware devices, the system and application heaps, the stack, and other information used by applications.

read/write permission: Information associated with an access path that indicates whether the file can be read from, written to, both read from and written to, or whatever the file's open permission allows.

reallocate: To allocate new space in the heap for a purged block, updating its master pointer to point to its new location.

reference number: A number greater than 0, returned by the Resource Manager when a resource file is opened, by which you can refer to that file. In Resource Manager routines that expect a reference number, 0 represents the system resource file.

reference value: In a window record or control record, a 32-bit field that an application program may store into and access for any purpose.

region: An arbitrary area or set of areas on the QuickDraw coordinate plane. The outline of a region should be one or more closed loops.

register-based routine: A Toolbox or Operating System routine that receives its parameters and returns its results, if any, in registers.

relative handle: A handle to a relocatable block expressed as the offset of its master pointer within the heap zone, rather than as the absolute memory address of the master pointer.

release: To free an allocated area of memory, making it available for reuse.

release timer: A timer for determining when an exactly-once response buffer can be released.

relocatable block: A block that can be moved within the heap during compaction.

resource: Data or code stored in a resource file and managed by the Resource Manager.

resource attribute: One of several characteristics, specified by bits in a resource reference, that determine how the resource should be dealt with.

resource data: In a resource file, the data that comprises a resource.

resource file: The resource fork of a file.

resource fork: The part of a file that contains data used by an application (such as menus, fonts, and icons). The resource fork of an application file also contains the application code itself.

resource header: At the beginning of a resource file, data that gives the offsets to and lengths of the resource data and resource map.

resource ID: A number that, together with the resource type, identifies a resource in a resource file. Every resource has an ID number.

Resource Manager: The part of the Toolbox that reads and writes resources.

resource map: In a resource file, data that is read into memory when the file is opened and that, given a resource specification, leads to the corresponding resource data.

resource name: A string that, together with the resource type, identifies a resource in a resource file. A resource may or may not have a name.

resource reference: In a resource map, an entry that identifies a resource and contains either an offset to its resource data in the resource file or a handle to the data if it's already been read into memory.

resource specification: A resource type and either a resource ID or a resource name.

resource type: The type of a resource in a resource file, designated by a sequence of four characters (such as 'MENU' for a menu).

response BDS: A data structure used to pass response information to the ATP module.

result code: An integer indicating whether a routine completed its task successfully or was prevented by some error condition (or other special condition, such as reaching the end of a file).

resume procedure: A procedure within an application that allows the application to recover from system errors.

retry count: The maximum number of retransmissions for an NBP or ATP packet.

retry interval: The time between retransmissions of a packet by NBP or ATP.

ROM: The Macintosh's permanent read-only memory, which contains the routines for the Toolbox and Operating System, and the various system traps.

routine selector: An integer that's pushed onto the stack before the _PackN macro is invoked, to identify which routine to execute. (N is the resource ID of a package; all macros for calling routines in the package expand to invoke _PackN.)

routing table: A table in a bridge that contains routing information.

Routing Table Maintenance Protocol (RTMP): An AppleTalk protocol that's used internally by AppleTalk to maintain tables for routing datagrams through an internet.

row width: The number of bytes in each row of a bit image.

RTMP: See **Routing Table Maintenance Protocol.**

RTMP socket: The socket in a node used to implement RTMP.

RTMP stub: The RTMP code in a nonbridge node.

scaling factor: A value, given as a fraction, that specifies the amount a character should be stretched or shrunk before it's drawn.

SCC: See **Serial Communications Controller**.

scrap: A place where cut or copied data is stored.

scrap file: The file containing the desk scrap (usually named "Clipboard File").

Scrap Manager: The part of the Toolbox that enables cutting and pasting between applications, desk accessories, or an application and a desk accessory.

screen buffer: A block of memory from which the video display reads the information to be displayed.

sector: Disk space composed of 512 consecutive bytes of standard information and 12 bytes of file tags.

segment: One of several parts into which the code of an application may be divided. Not all segments need to be in memory at the same time.

Segment Loader: The part of the Operating System that loads the code of an application into memory, either as a single unit or divided into dynamically loaded segments.

selection range: The series of characters (inversely highlighted), or the character position (marked with a blinking caret), at which the next editing operation will occur.

sequence number: A number from 0 to 7, assigned to an ATP response datagram to indicate its ordering within the response.

Serial Communications Controller (SCC): The chip that handles serial I/O through the modem and printer ports.

serial data: Data communicated over a single-path communication line, one bit at a time.

Serial Driver: A device driver that controls communication, via serial ports, between applications and serial peripheral devices.

signature: A four-character sequence that uniquely identifies an application to the Finder.

socket: A logical entity within the node of a network.

socket client: A software process in a node that owns a socket.

socket listener: The portion of a socket client that receives and services datagrams addressed to that socket.

socket number: An identifier for a socket.

socket table: A listing of all the socket listeners for each active socket in a node.

software overrun error: The condition that occurs when an input driver's buffer becomes full.

solid shape: A shape that's filled in with any pattern.

sound buffer: A block of memory from which the sound generator reads the information to create an audio waveform.

Sound Driver: The device driver that controls sound generation in an application.

sound procedure: A procedure associated with an alert that will emit one of up to four sounds from the Macintosh's speaker. Its integer parameter ranges from 0 to 3 and specifies which sound.

source transfer mode: One of eight transfer modes for drawing text or transferring any bit image between two bit maps.

space state: The state of a transmission line indicating a binary 0.

spool printing: Writing a representation of a document's printed image to disk or to memory, and then printing it (as opposed to immediate draft printing).

square-wave synthesizer: The part of the Sound Driver used to produce less harmonic sounds than the four-tone synthesizer, such as beeps.

stack: The area of memory in which space is allocated and released in LIFO (last-in-first-out) order.

stack-based routine: A Toolbox or Operating System routine that receives its parameters and returns its results, if any, on the stack.

stack frame: The area of the stack used by a routine for its parameters, return address, local variables, and temporary storage.

stage: Every alert has four stages, corresponding to consecutive occurrences of the alert, and a different response may be specified for each stage.

Standard File Package: A Macintosh package for presenting the standard user interface when a file is to be saved or opened.

start bit: A serial data communications bit that signals that the next bits transmitted are data bits.

status information: Information transmitted to an application by a device driver. It may indicate the current mode of operation, the readiness of the device, the occurrence of errors, and so on.

status routine: The part of a device driver's code that implements Device Manager Status calls.

stop bit: A serial data communications bit that signals the end of data bits.

structure region: An entire window; its complete "structure".

style: See **character style**.

style dialog: A dialog that sets options affecting the page dimensions; associated with the Page Setup command.

synchronous execution: After calling a routine synchronously, an application cannot continue execution until the routine is completed.

synthesizer: See **free-form, four-tone,** or **square-wave synthesizer**.

synthesizer buffer: A description of the sound to be generated by a synthesizer.

system error alert: An alert box displayed by the System Error Handler.

system error alert table: A resource that determines the appearance and function of system error alerts.

System Error Handler: The part of the Operating System that assumes control when a fatal system error occurs.

system error ID: An ID number that appears in a system error alert to identify the error.

system event mask: A global event mask that controls which types of events get posted into the event queue.

system font: The font that the system uses (in menus, for example). Its name is Chicago.

system font size: The size of text drawn by the system in the system font; 12 points.

system heap: The portion of the heap reserved for use by the Operating System.

system heap zone: The heap zone provided by the Memory Manager for use by the Operating System; equivalent to the system heap.

system resource: A resource in the system resource file.

system resource file: A resource file containing standard resources, accessed if a requested resource wasn't found in any of the other resource files that were searched.

system startup information: Certain configurable system parameters that are stored in the first two logical blocks of a volume and read in at system startup.

system window: A window in which a desk accessory is displayed.

TextEdit: The part of the Toolbox that supports the basic text entry and editing capabilities of a standard Macintosh application.

TextEdit scrap: The place where certain TextEdit routines store the characters most recently cut or copied from text.

thousands separator: The character that separates every three digits to the left of the decimal point.

thumb: The Control Manager's term for the scroll box (the indicator of a scroll bar).

tick: A sixtieth of a second.

Toolbox: Same as **User Interface Toolbox**.

Toolbox Event Manager: The part of the Toolbox that allows your application program to monitor the user's actions with the mouse, keyboard, and keypad.

Toolbox Utilities: The part of the Toolbox that performs generally useful operations such as fixed-point arithmetic, string manipulation, and logical operations on bits.

track: Disk space composed of 8 to 12 consecutive sectors. A track corresponds to one ring of constant radius around the disk.

transaction: A request-response communication between two ATP clients. See **transaction request** and **transaction response**.

transaction ID: An identifier assigned to a transaction.

transaction request: The initial part of a transaction in which one socket client asks another to perform an operation and return a response.

transaction response: The concluding part of a transaction in which one socket client returns requested information or simply confirms that a requested operation was performed.

Transcendental Functions Package: A Macintosh package that contains trigonometric, logarithmic, exponential, and financial functions, as well as a random number generator.

transfer mode: A specification of which Boolean operation QuickDraw should perform when drawing or when transferring a bit image from one bit map to another.

trap dispatch table: A table in RAM containing the addresses of all Toolbox and Operating System routines in encoded form.

trap dispatcher: The part of the Operating System that examines a trap word to determine what operation it stands for, looks up the address of the corresponding routine in the trap dispatch table, and jumps to the routine.

trap macro: A macro that assembles into a trap word, used for calling a Toolbox or Operating System routine from assembly language.

trap number: The identifying number of a Toolbox or Operating System routine; an index into the trap dispatch table.

trap word: An unimplemented instruction representing a call to a Toolbox or Operating System routine.

unimplemented instruction: An instruction word that doesn't correspond to any valid machine-language instruction but instead causes a trap.

unit number: The number of each device driver's entry in the unit table.

unit table: A 128-byte nonrelocatable block containing a handle to the device control entry for each device driver.

unlock: To allow a relocatable block to be moved during heap compaction.

unmounted volume: A volume that hasn't been inserted into a disk drive and had descriptive information read from it, or a volume that previously was mounted and has since had the memory used by it released.

unpurgeable block: A relocatable block that can't be purged from the heap.

update event: An event generated by the Window Manager when a window's contents need to be redrawn.

update region: A window region consisting of all areas of the content region that have to be redrawn.

user bytes: Four bytes in an ATP header provided for use by ATP's clients.

User Interface Toolbox: The software in the Macintosh ROM that helps you implement the standard Macintosh user interface in your application.

validity status: A number stored in parameter RAM designating whether the last attempt to write there was successful. (The number is $A8 if so.)

variation code: The part of a window or control definition ID that distinguishes closely related types of windows or controls.

VBL task: A task performed during the vertical retrace interrupt.

vector table: A table of interrupt vectors in low memory.

version data: In an application's resource file, a resource that has the application's signature as its resource type; typically a string that gives the name, version number, and date of the application.

version number: A number from 0 to 255 used to distinguish between files with the same name.

Versatile Interface Adapter (VIA): The chip that handles most of the Macintosh's I/O and interrupts.

vertical blanking interrupt: See **vertical retrace interrupt.**

vertical blanking interval: The time between the display of the last pixel on the bottom line of the screen and the first one on the top line.

vertical retrace interrupt: An interrupt generated 60 times a second by the Macintosh video circuitry while the beam of the display tube returns from the bottom of the screen to the top; also known as vertical blanking interrupt.

Vertical Retrace Manager: The part of the Operating System that schedules and executes tasks during the vertical retrace interrupt.

vertical retrace queue: A list of the tasks to be executed during the vertical retrace interrupt.

VIA: See **Versatile Interface Adapter.**

view rectangle: In TextEdit, the rectangle in which the text is visible.

visible control: A control that's drawn in its window (but may be completely overlapped by another window or other object on the screen).

visible window: A window that's drawn in its plane on the desktop (but may be completely overlapped by another window or object on the screen).

visRgn: The region of a grafPort, manipulated by the Window Manager, that's actually visible on the screen.

volume: A piece of storage medium formatted to contain files; usually a disk or part of a disk. A 400K-byte 3 1/2-inch Macintosh disk is one volume.

volume allocation block map: A list of 12-bit entries, one for each allocation block, that indicate whether the block is currently allocated to a file, whether it's free for use, or which block is next in the file. Block maps exist both on volumes and in memory.

volume attributes: Information contained on volumes and in memory indicating whether the volume is locked, whether it's busy (in memory only), and whether the volume control block matches the volume information (in memory only).

volume buffer: Memory used initially to load the master directory block, and used thereafter for reading from files that are opened without an access path buffer.

volume control block: A nonrelocatable block that contains volume-specific information, including the volume information from the master directory block.

volume-control-block queue: A list of the volume control blocks for all mounted volumes.

volume index: A number identifying a mounted volume listed in the volume-control-block queue. The first volume in the queue has an index of 1, and so on.

volume information: Volume-specific information contained on a volume, including the volume name and the number of files on the volume.

volume name: A sequence of up to 27 printing characters that identifies a volume; followed by a colon (:) in File Manager routine calls, to distinguish it from a file name.

volume reference number: A unique number assigned to a volume as it's mounted, used to refer to the volume.

waveform: The physical shape of a wave.

waveform description: A sequence of bytes describing a waveform.

wavelength: The horizontal extent of one complete cycle of a wave.

window: An object on the desktop that presents information, such as a document or a message.

window class: In a window record, an indication of whether a window is a system window, a dialog or alert window, or a window created directly by the application.

window definition function: A function called by the Window Manager when it needs to perform certain type-dependent operations on a particular type of window, such as drawing the window frame.

window definition ID: A number passed to window-creation routines to indicate the type of window. It consists of the window definition function's resource ID and a variation code.

window frame: The structure region of a window minus its content region.

window list: A list of all windows ordered according to their front-to-back positions on the desktop.

Window Manager: The part of the Toolbox that provides routines for creating and manipulating windows.

Window Manager port: A grafPort that has the entire screen as its portRect and is used by the Window Manager to draw window frames.

window record: The internal representation of a window, where the Window Manager stores all the information it needs for its operations on that window.

window template: A resource that contains information from which the Window Manager can create a window.

word: In TextEdit, any series of printing characters, excluding spaces (ASCII code $20) but including nonbreaking spaces (ASCII code $CA).

word wraparound: Keeping words from being split between lines when text is drawn.

write data structure: A data structure used to pass information to the ALAP or DDP modules.

zone: An arbitrary subset of AppleTalk networks in an internet. See also **heap zone**.

zone header: The internal "housekeeping" information maintained by the Memory Manager at the beginning of each heap zone.

zone pointer: A pointer to a zone record.

zone record: A data structure representing a heap zone.

zone trailer: A minimum-size free block marking the end of a heap zone.

INDEX

Index

Index

Inside Macintosh

Welcome to the world of programming for the Macintosh®. No other personal computer has been as enthusiastically received by the programming community, as the large—and growing—body of Macintosh software attests. *Inside Macintosh* provides the guidelines and technical information that you'll need to develop Macintosh programs, but many other resources can help speed and simplify your software development efforts.

Development Languages

You won't have to look far to find a development language that suits your specific requirements. A growing family of Macintosh languages will serve your development needs whether your expertise is in Pascal, C, Assembler, FORTH, FORTRAN, COBOL, BASIC, Lisp, Modula-2, or one of many others. And the information in *Inside Macintosh* can be applied to any of the Macintosh languages.

The Certified Developer Program

If your primary business is developing software products for commercial markets, we strongly suggest that you investigate the Apple Certified Developer Program. This program helps developers produce and bring Macintosh products to market by providing them with support programs, services, and information. Among them are

- **Technical Support:** Apple's Developer Technical Support Group offers fast answers by way of AppleLink® or MCI electronic mail.
- **Macintosh Technical Notes:** This is a bimonthly package of supplemental technical information.
- **AppleLink:** Through this electronic service, you can get answers to your technical questions and current information on Apple and third-party products and programs.
- **Certified Developer Mailings:** These monthly mailings keep you informed about Apple's products, development tools, and technical and company directions.
- **The Information Exchange:** This information, available in printed and HyperCard® stack form, lists company-sponsored programs and services available to you and your company.
- ***Outside Apple*:** This monthly newsletter informs you of developer-oriented Apple groups, programs, and events.

You must meet certain criteria to get Certified Developer status. You can get an information package and application by writing to

Developer Programs
Apple Computer, Inc.
20525 Mariani Avenue, M/S 51-W
Cupertino, CA 95014

APDA

The Apple Programmer's and Developer's Association, APDA™, provides technical documentation and products for all programmers and developers who work on Apple equipment. It provides material that is unavailable elsewhere (including preliminary documentation of new Apple products). APDA also sells compilers and other tools from both Apple and third-party sources. For information on joining, write to

APDA
290 SW 43rd Street
Renton, WA 98055
(206) 251-6548

Technical Notes

Published bimonthly by Developer Technical Support, these notes answer frequently asked questions through examples and sample code and provide updates, additions, and corrections to the *Inside Macintosh* books. They are available through the Certified Developer Program, APDA, and major electronic information services.

Apple® Inside Macintosh Volume III

The Official Publication from Apple Computer, Inc.

> $19.95 FPT
USA

Written by the people at Apple Computer, *Inside Macintosh* is the definitive source of information for programmers writing application programs, desk accessories, device drivers, and other software for any of the computers in the Apple Macintosh® family. It includes:

- Guidelines for designing a user interface that conforms to the Macintosh standard.
- Descriptions of more than 1,200 ROM- and disk-based routines.
- A description of the Macintosh hardware.

Inside Macintosh is your guide to creating software for the Macintosh, whatever programming language you use. It describes the Pascal interfaces to the routines and, wherever applicable, gives special information for programming in assembly language. (If you're using a high-level language other than Pascal, your development system documentation should tell you how to apply the information in *Inside Macintosh*.) A typical chapter describes a related set of routines, such as the Window Manager, and provides:

- Key concepts and background information.
- Hints on which routines you need to learn about and how they fit into your program.
- A detailed description of each routine.

Inside Macintosh consists of six volumes. This volume, Volume III, contains:

- A discussion of your program's interface with the Macintosh Finder.™
- A description of the Macintosh 128K and 512K computers.
- Summaries of all the Managers and other software described in volumes I, II, and III.

Volume I contains important introductory material and describes the QuickDraw graphics package and important Managers such as the Resource, Font, and Menu Managers. Volume II complements Volume I in describing the Managers that perform such basic routines as file and device I/O, memory management, and interrupt handling. Volume IV discusses the changes introduced by the Macintosh 512K Enhanced and Macintosh Plus computers, including the Hierarchical File System and the SCSI port. Volume V discusses the changes introduced by the Macintosh SE and Macintosh II computers, including color, NuBus™ slots, and the Apple Desktop Bus.™ *Inside Macintosh X-Ref* provides a single index to *Inside Macintosh* and other Macintosh technical books.

About the cover: This design represents a new look for the original edition of *Inside Macintosh, Volume III,* and the other books in the Apple Technical Library. The contents have not been changed.

Printed in U.S.A.

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010
TLX 171-576

Addison-Wesley Publishing Company, Inc.

51995
9 780201 177336

ISBN 0-201-17733-1