



A/UX Toolbox: Macintosh ROM Interface

Release 3.0

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for program coverage and details. In some countries the replacement period may be different; check with your authorized Apple dealer.

ALL IMPLIED WARRANTIES ON THE MEDIA AND MANUALS, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has tested the software and reviewed the documentation, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS, OR IMPLIED, WITH RESPECT TO SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, even if advised of the possibility of such damages. In particular, Apple shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering such programs or data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS, OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

 Apple Computer, Inc.

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or part, without written consent of Apple, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose.

The Apple logo is a registered trademark of Apple Computer, Inc. Use of the “keyboard” Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

© Apple Computer, Inc., 1992
20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010

APDA, Apple, the Apple logo, AppleLink, AppleShare, AppleTalk, A/UX, EtherTalk, LaserWriter, LocalTalk, Macintosh, MacTCP, MPW, MultiFinder, ProDOS, and SANE are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Finder, MacroMaker, MacX, QuickDraw, ResEdit, and TrueType are trademarks of Apple Computer, Inc.

Adobe, Adobe Illustrator, and PostScript are trademarks of Adobe Systems Incorporated, registered in the United States.

Electrocomp 2000 is a trademark of Image Graphics, Inc.

ITC Garamond and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation.

Motorola is a registered trademark of Motorola Corporation.

Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.

NFS is a trademark of Sun Microsystems, Inc.

NuBus is a trademark of Texas Instruments.

QuarkXPress is a registered trademark of Quark, Inc.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

VMS is a trademark of Digital Equipment Corporation.

Simultaneously published in the United States and Canada.

Mention of third-party products is for informational purposes only and constitutes neither an endorsement nor a recommendation. Apple assumes no responsibility with regard to the performance or use of these products.

Contents

Figures and Tables / xiii

About This Guide / xv

What's in this manual / xvi

Conventions used in this guide / xvii

Keys and key combinations / xvii

Terminology / xvii

The `Courier` font / xviii

Font styles / xix

A/UX command syntax / xix

Manual page reference notation / xx

For more information / xxi

1 About the A/UX Toolbox / 1-1

Overview / 1-2

New features in A/UX Release 3.0 / 1-3

A/UX Finder user interface / 1-3

Increased manager support / 1-4

Connectivity support / 1-5

Compatibility requirements / 1-6

Debugging under A/UX / 1-6

Contents of the A/UX Toolbox / 1-7

How the A/UX Toolbox works / 1-8

2	Using the A/UX Toolbox / 2-1
	Application development environments / 2-2
	Your application in the A/UX Finder environment / 2-4
	Using the <code>ui_setselect</code> call / 2-5
	Developing an A/UX Toolbox application / 2-6
	Developing the source code / 2-7
	Developing the resource file / 2-9
	Building and running the sample programs / 2-10
3	A/UX Toolbox Utilities and Extensions / 3-1
	Using the A/UX Toolbox utilities / 3-2
	A/UX Toolbox variables / 3-3
	Additional trap and routine / 3-4
	AUXDispatch trap / 3-4
	Using <code>select</code> to monitor A/UX I/O activity and Macintosh events / 3-6
	A/UX Toolbox environment variables / 3-6
	Making A/UX system calls / 3-7
	The MacsBug debugger under A/UX / 3-11
	The <code>dbx</code> debugger under A/UX / 3-13
4	Compatibility Guidelines / 4-1
	Introduction / 4-2
	Differences in execution environments / 4-2
	32-bit address violations / 4-3
	Privileged microprocessor instructions / 4-4
	Direct hardware access / 4-6
	Newline characters / 4-7
	File Manager / 4-9
	Memory Manager / 4-9
	International character support / 4-9
	Differences in C compilers / 4-10
	Differences in language conventions / 4-11

5 A/UX and Macintosh User Interface Toolbox Differences / 5-1

About the Macintosh interface library / 5-2

32-Bit QuickDraw with Color QuickDraw / 5-4

Alias Manager / 5-5

Apple Desktop Bus / 5-5

Apple Event Manager / 5-5

AppleTalk Manager / 5-5

Binary-Decimal Conversion Package / 5-6

Color Manager / 5-6

Color Picker Package / 5-6

Control Manager / 5-6

Data Access Manager / 5-7

Deferred Task Manager / 5-7

Desk (Accessory) Manager / 5-7

Desktop Manager / 5-7

Device Manager / 5-7

Dialog Manager / 5-8

Disk Driver / 5-9

Disk Initialization Package / 5-9

Edition Manager / 5-9

Event Manager, Operating System / 5-9

Event Manager, Toolbox / 5-10

File Manager / 5-11

Floating-Point Arithmetic and Transcendental Functions Packages / 5-11

Font Manager / 5-12

Gestalt Manager / 5-12

Graphics Devices Manager / 5-13

Help Manager / 5-14

International Utilities Package / 5-14

List Manager Package / 5-14

Memory Manager / 5-14

Menu Manager / 5-15

Notification Manager / 5-15

Package Manager / 5-15

Palette Manager / 5-15

Picture Utilities Package / 5-16

Power Manager / 5-16

PPC Toolbox / 5-16

Printing Manager / 5-16

Process Manager / 5-16

- Resource Manager / 5-17
- Scrap Manager / 5-18
- Script Manager / 5-18
- SCSI Manager / 5-18
- Segment Loader / 5-19
 - Finder information / 5-19
 - Segment Loader routines / 5-20
 - The jump table / 5-20
 - Alternate buffer support / 5-20
- Serial Driver / 5-20
- Shutdown Manager / 5-22
- Slot Manager / 5-22
- Sound Manager / 5-23
 - Support details / 5-24
 - The Raw Sound Driver / 5-25
- Standard File Package / 5-27
- System Error Handler / 5-27
- TextEdit / 5-27
- Time Manager / 5-28
- Utilities, Operating System / 5-28
 - Date and time operations / 5-29
 - Miscellaneous utilities / 5-29
- Utilities, Toolbox / 5-29
- Vertical Retrace Manager / 5-29
- Window Manager / 5-30
- Calls patched under A/UX / 5-31
- Calls not supported under A/UX / 5-34

6 File Systems and File Formats / 6-1

- File systems / 6-2
 - Overall file organization / 6-2
 - Pathnames and filenames / 6-3
 - File permissions / 6-4
 - Extended file attributes / 6-6
 - Text files / 6-6
 - Mounting and unmounting floppy disks / 6-7
- Storing files in the Macintosh OS and in the A/UX operating system / 6-8
 - Automatic conversion / 6-14

AppleSingle and AppleDouble format internals / 6-16
 AppleSingle format / 6-16
 AppleDouble format / 6-19
Filename conventions / 6-20

Appendix A Additional Reading / A-1

Information sources / A-2
Required references / A-4
Supplementary references / A-5

Appendix B Toolbox Contents / B-1

Appendix C Implementation Notes / C-1

The A/UX Finder and Toolbox applications / C-2
Running an A/UX Toolbox application / C-2
 User interface device driver / C-3
 Initialization routine / C-3
 A-line traps / C-4
 “Not in ROM” routines / C-6
 Macintosh global variables / C-6
 File type and creator / C-6
Converting between C and Pascal conventions / C-7
 Storing strings / C-8
 Ordering and storing parameters / C-8
 Passing small structures / C-9
 Returning function results / C-9
 Register conventions / C-10

Appendix D Low-Memory Global Variables / D-1

Appendix E Resource Compiler and Decompiler / E-1


About the resource compiler and decompiler / E-2
 Standard type declaration files / E-3
 Using rez and derez / E-4

- Structure of a resource description file / E-5
- Sample resource description file / E-6
- Resource description statements / E-7
 - Syntax notation / E-7
 - Special terms / E-8
 - change—change a resource’s vital information / E-9
 - data—specify raw data / E-10
 - delete—delete a resource / E-11
 - include—include resources from another file / E-12
 - read—read data as a resource / E-15
 - resource—specify resource data / E-16
 - type—declare resource type / E-20
 - Labels / E-32
- Preprocessor directives / E-39
 - Variable definitions / E-40
 - include directives / E-40
 - If-then-else processing / E-41
 - Print directive / E-42
- Resource description syntax / E-43
 - Numbers and literals / E-43
 - Expressions / E-44
 - Variables and functions / E-46
 - String values / E-46
 - Numeric values / E-47
 - Strings / E-49
 - Escape characters / E-50

Appendix F C Interface Library / F-1

- Interface library files / F-2
- Structures and calls by library / F-5
 - 32-Bit QuickDraw with Color QuickDraw / F-5
 - Color Picker / F-14
 - Common type definitions / F-15
 - Control Manager / F-15
 - Deferred Task Manager / F-17
 - Definitions for `AUXDispatch` / F-17
 - Definitions for ROM / F-17
 - Desk Manager / F-18
 - Device Manager / F-18

Dialog Manager /	F-19
Disk Driver /	F-21
Disk Initialization Package /	F-21
Event Manager, Operating System /	F-22
Event Manager, Toolbox /	F-22
File Manager /	F-23
Font Manager /	F-28
Gestalt Manager /	F-29
List Manager Package /	F-29
List of Macintosh traps /	F-30
Low-memory equates /	F-30
Memory Manager /	F-31
Menu Manager /	F-34
Notification Manager /	F-36
Package Manager /	F-36
Palette Manager /	F-38
Printing Manager /	F-39
Print traps /	F-40
Process Manager /	F-41
Resource Manager /	F-42
Scrap Manager /	F-45
Script Manager /	F-46
Segment Loader /	F-48
Serial Driver /	F-49
Shutdown Manager /	F-49
Slot Manager /	F-50
Sound Manager /	F-52
String conversion between Pascal and C /	F-54
System Error Handler /	F-54
TextEdit /	F-54
Time Manager /	F-56
Utilities, Operating System /	F-57
Utilities, Toolbox /	F-58
Vertical Retrace Manager /	F-60
Video Driver /	F-60
Window Manager /	F-61
Calls in alphabetical order /	F-63
Index /	In-1



Figures and Tables

Chapter 1 About the A/UX Toolbox / 1-1

Figure 1-1 Interactions among an application, the A/UX Toolbox, and the ROM code / 1-9

Chapter 2 Using the A/UX Toolbox / 2-1

Figure 2-1 Application development and execution environments / 2-2

Figure 2-2 Incorporating the A/UX Toolbox into development code / 2-8

Figure 2-3 Developing a resource file by using `rez` / 2-9

Chapter 4 Compatibility Guidelines / 4-1

Table 4-1 Privileged microprocessor instructions within the A/UX Toolbox / 4-5

Chapter 5 A/UX and Macintosh User Interface Toolbox Differences / 5-1

Table 5-1 Status of User Interface Toolbox and Macintosh OS libraries in the A/UX Toolbox / 5-2

Table 5-2 ROM calls patched under the A/UX Toolbox / 5-31

Table 5-3 ROM calls not supported under the A/UX Toolbox / 5-34

Chapter 6 File Systems and File Formats / 6-1

- Figure 6-1 Elements of a file in the native Macintosh OS environment / 6-9
- Figure 6-2 Typical contents of an AppleSingle file / 6-11
- Figure 6-3 Typical contents of a pair of AppleDouble files / 6-12
- Figure 6-4 Elements of Macintosh data and resource files in simple A/UX format / 6-13
- Figure 6-5 Formats for “file info” field entries / 6-19
- Table 6-1 A/UX permissions mapped to AppleShare privileges / 6-5
- Table 6-2 Automatic conversion of Macintosh files / 6-15
- Table 6-3 AppleSingle file header / 6-16

Appendix C Implementation Notes / C-1

- Figure C-1 A-line trap handling in A/UX / C-5

Appendix D Low-Memory Global Variables / D-1

- Table D-1 General global variables / D-2
- Table D-2 Window Manager globals / D-6
- Table D-3 TextEdit globals / D-6
- Table D-4 Resource Manager globals / D-7

Appendix E Resource Compiler and Decompiler / E-1

- Figure E-1 `rez` and `derez` / E-2
- Figure E-2 Creating a resource file / E-4
- Figure E-3 Padding of literals / E-44
- Figure E-4 Internal representation of a Pascal string / E-49
- Table E-1 Numeric constants / E-43
- Table E-2 Resource-description expression operators / E-45
- Table E-3 Resource compiler escape sequences / E-50
- Table E-4 Numeric escape sequences / E-51

Appendix F C Interface Library / F-1

- Table F-1 Interface library files / F-3

Font styles

Italics are used to indicate that a word or set of words is a placeholder for part of a command. For example,

```
cat filename
```

tells you that *filename* is a placeholder for the name of a file you wish to view. If you want to view the contents of a file named `Elvis`, type the word `Elvis` in place of *filename*. In other words, enter

```
cat Elvis
```

New terms appear in **boldface** where they are defined. Boldface is also used for steps in a series of instructions.

A/UX command syntax

A/UX commands follow a specific command syntax. A typical A/UX command gives the command name first, followed by options and arguments. For example, here is the syntax for the `wc` command:

```
wc [-l] [-w] [directory...]
```

In this example, `wc` is the command, `-l` and `-w` are options, *directory* is an argument, and the ellipses (...) indicate that more than one argument can be used. Note that each command element is separated by a space.

The following list gives more information about the elements of an A/UX command.

<i>Element</i>	<i>Description</i>
<i>command</i>	The command name.
<i>option</i>	A character or group of characters that modifies the command. Most options have the form <i>-option</i> , where <i>option</i> is a letter representing an option. Most commands have one or more options.
<i>argument</i>	A modification or specification of a command, usually a filename or symbols representing one or more filenames.
[]	Brackets used to enclose an optional item—that is, an item that is not essential for execution of the command.
...	Ellipses used to indicate that more than one argument may be entered.

For example, the `wc` command is used to count lines, words, and characters in a file. Thus, you can enter

```
wc -w Priscilla
```

In this command line, `-w` is the option that instructs the command to count all of the words in the file, and the argument `Priscilla` is the file to be searched.

Manual page reference notation

A/UX Command Reference, *A/UX Programmer's Reference*, *A/UX System Administrator's Reference*, *X11 Command Reference for A/UX*, and *X11 Programmer's Reference for A/UX* contain descriptions of commands, subroutines, and other related information. Such descriptions are known as *manual pages* (often shortened to *man pages*). Manual pages are organized within these references by section numbers. The standard A/UX cross-reference notation is

command(section)

where *command* is the name of the command, file, or other facility; *section* is the number of the section in which the item resides.

- Items followed by section numbers (1M) and (8) are described in *A/UX System Administrator's Reference*.
- Items followed by section numbers (1) and (6) are described in *A/UX Command Reference*.
- Items followed by section numbers (2), (3), (4), and (5) are described in *A/UX Programmer's Reference*.
- Items followed by section number (1X) are described in *X11 Command Reference for A/UX*.
- Items followed by section numbers (3X) and (3Xt) are described in *X11 Programmer's Reference for A/UX*.

For example,

```
cat(1)
```

refers to the command `cat`, which is described in Section 1 of *A/UX Command Reference*.

You can display manual pages on the screen by using the `man` command. For example, enter the command

```
man cat
```

to display the manual page for the `cat` command, including its description, syntax, options, and other pertinent information. To exit, press the `SPACE BAR` until you see a command prompt, or type `q` at any time to return immediately to your command prompt.

For more information

To find out where you need to go for more information about how to use A/UX, see *Road Map to A/UX*. This guide contains descriptions of each A/UX guide and ordering information for all the guides in the A/UX documentation suite.

About This Guide

This manual describes the A/UX Toolbox, which gives you access from within A/UX to the Macintosh User Interface Toolbox. The User Interface Toolbox is software in the Macintosh ROM that facilitates implementation of the standard Macintosh interface in applications. This manual also provides compatibility guidelines for programs intended to run under both the A/UX operating system and the standard Macintosh Operating System (OS).

This manual is intended for developers who are porting a Macintosh application to A/UX or developing a Macintosh-like application under A/UX. This guide assumes that

- You are an experienced C programmer.
- You are familiar with the standard Macintosh User Interface Toolbox and Operating System.
- You are familiar with the A/UX development environment.

If you need information on any of these subjects, please refer to the resources listed in Appendix A. For a detailed description of the User Interface Toolbox, see *Inside Macintosh*, Volumes I, IV, V, and VI. For a detailed description of the Macintosh OS, see *Inside Macintosh*, Volumes II, IV, V, and VI.

What's in this manual

Here is a description of the contents of this manual:

- Chapter 1, “About the A/UX Toolbox,” gives an overview of the A/UX Toolbox. The chapter discusses A/UX features, standards compliance, and the new features in Release 3.0.
- Chapter 2, “Using the A/UX Toolbox,” explains the role of the A/UX Toolbox in program development and execution and describes the sample programs provided with the A/UX Toolbox.
- Chapter 3, “A/UX Toolbox Utilities and Extensions,” describes the utilities and special features in the A/UX Toolbox that support program development.
- Chapter 4, “Compatibility Guidelines,” summarizes the compatibility guidelines you must be aware of to write code that runs under both the Macintosh OS and A/UX.
- Chapter 5, “A/UX and Macintosh User Interface Toolbox Differences,” describes in detail the differences between the facilities available in the User Interface Toolbox and Macintosh OS and those provided with the A/UX Toolbox.
- Chapter 6, “File Systems and File Formats,” describes the differences between the file systems in A/UX and those in the Macintosh Operating System, and how file-system functions are mapped between the two systems. In addition, the chapter describes the formats used for storing Macintosh files in A/UX, and the results of automatic conversion of files transferred between the two systems.
- Appendix A, “Additional Reading,” lists books and other information sources that are helpful.
- Appendix B, “Toolbox Contents,” lists directories and files that are part of the A/UX Toolbox or that are of special interest in application development.
- Appendix C, “Implementation Notes,” provides background information about implementation and compatibility issues.
- Appendix D, “Low-Memory Global Variables,” lists the Macintosh low-memory global variables that are supported in A/UX.
- Appendix E, “Resource Compiler and Decompiler,” describes the resource development tools that have been ported to the A/UX Toolbox from the Macintosh Programmer’s Workshop (MPW).

- Appendix F, “C Interface Library,” lists the functions, types, and parameters used by the A/UX Toolbox libraries.

Conventions used in this guide

A/UX guides follow specific conventions. For example, words that require special emphasis appear in specific fonts or font styles. The following sections describe the conventions used in all A/UX guides.

Keys and key combinations

Certain keys on the keyboard have special names. These modifier and character keys, often used in combination with other keys, perform various functions. In this guide, the names of these keys are in Initial Capital letters followed by SMALL CAPITAL letters.

The key names are

CAPS LOCK	DOWN ARROW (↓)	OPTION	SPACE BAR
COMMAND (⌘)	ENTER	RETURN	TAB
CONTROL	ESCAPE	RIGHT ARROW (→)	UP ARROW (↑)
DELETE	LEFT ARROW (←)	SHIFT	

Sometimes you will see two or more names joined by hyphens. The hyphens indicate that you use two or more keys together to perform a specific function. For example, Press COMMAND-K means “Hold down the COMMAND key and press the K key.”

Terminology

In A/UX guides, a certain term can represent a specific set of actions. For example, the word *enter* indicates that you type a series of characters on the command line and press the RETURN key. The instruction

Enter 1s

means “Type 1s and press the RETURN key.”

Here is a list of common terms and the corresponding actions you take.

<i>Term</i>	<i>Action</i>
Click	Press and then immediately release the mouse button.
Drag	Position the mouse pointer, press and hold down the mouse button while moving the mouse, and then release the mouse button.
Choose	Activate a command in a menu. To choose a command from a pull-down menu, click once on the menu title and, while holding down the mouse button, drag down until the command is highlighted. Then release the mouse button.
Select	Highlight a selectable object by positioning the mouse pointer on the object and clicking.
Type	Type an entry <i>without</i> pressing the RETURN key.
Enter	Type the series of characters indicated and press the RETURN key.

The Courier font

Throughout A/UX guides, words that you see on the screen or that you must type exactly as shown are in the `Courier` font. For example, suppose you see this instruction:

Type `date` on the command line and press Return.

The word `date` is in the `Courier` font to indicate that you must type it. Suppose you then read this explanation:

Once you press RETURN, you'll see something like this:

```
Tues Oct 17 17:04:00 PDT 1989
```

In this case, `Courier` is used to represent exactly what appears on the screen.

All A/UX manual page names are also shown in the `Courier` font. For example, the entry `ls(1)` indicates that `ls` is the name of a manual page in an A/UX reference manual. See “Manual Page Reference Notation” below for more information on A/UX command reference manuals.



1 About the A/UX Toolbox

Overview / 1-2

New features in A/UX Release 3.0 / 1-3

Compatibility requirements / 1-6

Debugging under A/UX / 1-6

Contents of the A/UX Toolbox / 1-7

How the A/UX Toolbox works / 1-8

This chapter gives a general overview of the functions of the A/UX Toolbox and specific information about the A/UX Toolbox in A/UX Release 3.0.

Overview

The A/UX Toolbox is a process environment that allows programs running under A/UX to make calls to the Macintosh User Interface Toolbox routines and to native Macintosh Operating System (OS) routines. With the A/UX Toolbox, you can run standard Macintosh applications unmodified as well as UNIX® programs that take advantage of the Macintosh interface. The A/UX Toolbox is included with the A/UX operating system. To use the A/UX Toolbox, you need only the standard A/UX distribution.

The A/UX Toolbox bridges the Macintosh and UNIX environments and gives you two kinds of code compatibility:

- You can execute Macintosh binary code (applications compiled in the Macintosh environment) under A/UX, within the current limitations of the A/UX Toolbox. (As the section “New Features in A/UX Release 3.0,” later in this chapter, makes clear, the new capabilities of the A/UX Toolbox remove many prior limitations.)
- Using the Macintosh Programmer’s Workshop (MPW), you can write common source code that can be separately built (compiled and linked) into executable code for both environments.

Both the User Interface Toolbox and the Macintosh OS are built into read-only memory (ROM) as well as into code modules in the System files. Because of differences between the UNIX operating system and the Macintosh OS, not all Macintosh ROM routines are available through the A/UX Toolbox. Release 3.0 has increased support for Macintosh ROM routines. Programs that are intended to run in both environments can use only the system routines common to both. Any Macintosh application that runs under Macintosh System 7 and does not access hardware directly or call routines not supported by the A/UX Toolbox can run under A/UX Release 3.0. Chapter 5, “A/UX and Macintosh User Interface Toolbox Differences,” gives details about all Macintosh managers and their support under the A/UX Toolbox; Table 5-1 in that chapter summarizes manager support. A/UX Release 3.0 also includes support for AppleTalk Phase II communications software running on both LocalTalk and Ethernet hardware.

The A/UX Toolbox supports some Macintosh device drivers, but not those that manipulate hardware directly.

New features in A/UX Release 3.0

A/UX Release 3.0 is a major enhancement that combines a standard UNIX operating system and programming environment with Macintosh System 7 capabilities, including the standard Macintosh Finder user interface. Macintosh applications with extensive multimedia capabilities, for example, can now run in a full UNIX environment. Macintosh applications can run together in the A/UX Toolbox environment, and users can transfer information between applications by using the Clipboard, the Scrapbook, editions, and AppleEvents, as in the Macintosh OS. For example, a user can run a graphics application to publish illustrations that can be subscribed to by a word-processing application. A user can run a computer-aided design (CAD) application and a spreadsheet application, and copy numbers developed for the design into the spreadsheet for use in cost calculations.

The complete features of A/UX Release 3.0 are listed in the general A/UX manuals. This section presents the major items of interest from a Toolbox developer viewpoint.

A/UX Finder user interface

A/UX Release 3.0 provides the user interface of Macintosh System 7, displaying both Macintosh and UNIX applications and directories as icons. The UNIX file permissions are shown in the icon display of files and folders. Icons are highlighted or dimmed according to the file permissions accorded the user who logged in. Macintosh floppy disks are accessible from the A/UX Finder. UNIX permissions can be modified from pull-down menus.

Users can open applications of both kinds by double-clicking, and can move files by dragging. Text files moved between the UNIX and Macintosh environments are automatically translated as needed. Chapter 6, "File Systems and File Formats," discusses the results of automatic conversion.

Increased manager support

The level of support for managers in Release 3.0 of the A/UX Toolbox has been generally extended to match the support provided by System 7. Several managers that were not supported or were partially supported are now fully supported, and several new managers have been added.

- *Alias Manager* The Alias Manager is fully supported. The Alias Manager allows you to create and use aliases in the Macintosh environment, which are similar to symbolic links in the UNIX environment.
- *Apple Event Manager* The Apple Event Manager is fully supported. The Apple Event Manager provides mechanisms for sending and receiving events between applications.
- *Data Access Manager* The Data Access Manager is fully supported. The Data Access Manager makes it easy for your application to communicate with data sources, such as databases.
- *Edition Manager* The Edition Manager is fully supported. The Edition Manager gives your application the ability to dynamically share data with other applications.
- *File Manager* The File Manager supports A/UX Toolbox access to the various UNIX file systems (Berkeley UNIX file system [UFS], System V file system [SVFS], and Network File System [NFS]) as well as those of the Macintosh OS (hierarchical file system [HFS], Macintosh file system [MFS], and AppleShare). The File Manager now implements file IDs and file specification (FSSpec) records. Multiple HFS partitions on a single disk are now supported.
- *Help Manager* The Help Manager is fully supported. The Help Manager lets you easily incorporate on-line assistance into your application.
- *Memory Manager* The Memory Manager is fully supported. The Memory Manager now allows you to use a portion of your hard disk as though it were chip-based RAM.
- *PPC Toolbox* The PPC Toolbox is fully supported. The PPC Toolbox allows your application to communicate directly with other applications.
- *Process Manager* The Process Manager is fully supported. The Process Manager allows your application to launch other applications.
- *Sound Manager* The Sound Manager is mostly supported. The Sound Manager now provides for sound input, in addition to the continuous sampled sound output and note and wavetable synthesizing available previously.

Connectivity support

A/UX Release 3.0 supports the following features:

- *AppleTalk 2.0 Phase II* Access by LocalTalk or EtherTalk is fully supported.
- *FileShare* Peer-to-peer networking through FileShare is fully supported.
- *NFS 4.0 enhancement* A/UX 3.0 implements NFS 4.0 with 4.1 enhancements. New features include automounting and directory export.
- *MacTCP* Macintosh network applications written to the Macintosh Transmission Control Protocol (MacTCP) programmer interface are supported.
- *AFP server (FileShare)* The AppleTalk Filing Protocol (AFP) built into Release 3.0 allows networked users access to shared volumes on the A/UX Finder desktop. The AFP server implements the facility known as FileShare of Macintosh System 7.
- *Macintosh Communications Toolbox* The Communications Toolbox is fully supported.
- *CD-ROM* The Apple CD-SC peripheral is partially supported. CD-ROM discs with either High Sierra or ISO 9660 formats can be used. Additionally, Macintosh HFS and UNIX file systems are supported. Audio CDs are not supported.

Two implementations of the X Window System are shipped with Release 3.0. Developers of X applications will find a favorable development environment in A/UX Release 3.0.

- X11 for A/UX is a standard implementation of the X Window System developed at the Massachusetts Institute of Technology, which utilizes Release 4 of X Window System Version 11. X11 provides a complete development environment. Users can switch between the X11 environment and the A/UX Finder environment. X11 is described in *X11 User's Guide for A/UX*.
- MacX, Apple's implementation of the X Window System, provides a Finder-like environment for X11, with such features as pull-down menus, dialog boxes, and windows in which users can run X applications. MacX also implements Release 4 of X. Users have a range of installation choices for the user environment displayed by MacX. At one end of the range, MacX allows the user to switch between either the standard X11 look or a Finder-like environment; at the other end, only a Finder-like environment is displayed; and in the middle are a variety of tradeoffs, in which portions of both interfaces can be used. Whatever kind of MacX display the user chooses, the user can switch between the MacX environment and the A/UX Finder environment. X11 is described in *MacX User's Guide*.

Compatibility requirements

For a Macintosh application binary to run in the A/UX Release 3.0 environment, it must meet certain requirements, which are briefly summarized here. Generally, any application, INIT, or CDEV that runs under System 7 will run under A/UX. For more information, see “Your Application in the A/UX Finder Environment” in Chapter 2.

- *32-bit clean* Macintosh applications must be 32-bit clean to run in the standard A/UX Toolbox environment (as they must be to run under 32-bit mode in System 7). A special 24-bit environment is also furnished that provides a 24-bit test environment for developers who are making their applications 32-bit clean. This environment supports the running of older applications that have not been converted and orphan applications that never will be. The 24-bit environment is accessed by a special login. For information on this special login, see *A/UX Essentials*.
- *Compatible with System 7* A Macintosh binary that does not run in a Macintosh OS multitasking environment (either System 7 or MultiFinder) will not run under A/UX Release 3.0.
- *No calls on unsupported traps or system calls* An application cannot make calls that are not supported under the A/UX Toolbox. See Chapter 5, “A/UX and Macintosh User Interface Toolbox Differences,” for detailed information on call support.
- *No direct access to hardware* An application cannot issue instructions for direct control of hardware (although video memory is directly accessible).

Debugging under A/UX

The `dbx`, `adb`, and `sdb` debuggers, used frequently in UNIX development environments, are delivered with A/UX 3.0. These debuggers require use of an additional terminal that can communicate with your computer over either a serial line or a network.

The MacsBug debugger provides a familiar Macintosh software debugging tool for use with A/UX. MacsBug (version 6.2 required) is available from the Apple Programmers and Developers Association (APDA).

Several new and enhanced application development tools are included in the A/UX Developer's Tools product, also available from APDA. A new, ANSI-compliant C compiler (c89) is included, as is a complete library of A/UX system calls that can be used from the MPW environment to assist in the development of hybrid applications.

Hybrid applications are programs that employ facilities from both the UNIX and Macintosh application models. There are two basic types of hybrid applications. The first type is a UNIX application that uses the A/UX Toolbox to provide an interface that has the Macintosh look and feel. Hybrid applications of this type are called *UNIX hybrid applications*. The A/UX CommandShell application is an example of a UNIX hybrid application. The second type of hybrid application is a Macintosh application that makes UNIX system calls. Hybrid applications of this type are called *Macintosh hybrid applications*.

Contents of the A/UX Toolbox

This section summarizes the types of files that are included in the A/UX Toolbox, the locations of these files, and where you can find more information about some of them.

The `/mac` directory contains Macintosh-specific material:

- | | |
|-----------------------|--|
| <code>/mac/sys</code> | This directory contains the System Folders used for startup, login, and users without a personal System Folder. (For more information on personal System Folders, see <i>A/UX Essentials</i> and <code>systemfolder(1M)</code> .) The System file provided with Release 3.0 of A/UX is almost identical in functionality to the System file provided with Macintosh System 7. |
| <code>/mac/bin</code> | This directory contains various executables, including a few utilities for use in developing and running applications with the A/UX Toolbox. See Chapter 3, "A/UX Toolbox Utilities and Extensions," for descriptions of <code>fcvnt</code> , <code>setfile</code> , <code>rez</code> , and <code>derez</code> . |
| <code>/mac/src</code> | This directory contains source code for sample applications, including <code>sample</code> , <code>qdsamp</code> , and the Sound Manager demo, <code>sndDemo</code> . The source material includes associated makefiles, which demonstrate how to compile and link an application; it also includes Macintosh system resource files for use with the sample programs. For additional information, see Chapter 2, "Using the A/UX Toolbox." |

`/mac/lib` This directory contains libraries in three subdirectories. The `rincludes` directory contains resource file material. The `sessiontypes` directory contains session-type information used at login. The `cmds` directory contains dialog scripts used to implement the Commando functions for UNIX commands.

Outside the `/mac` directory are certain other files that should be mentioned:

`/lib` This directory contains library routines used in the implementation of the A/UX Toolbox and in UNIX program development. An example of the first kind is the file `mac crt0.o`, and examples of the second kind are the files `libs.a`, `libposix.a`, and `termcap.a`.

`/usr/include/mac` This directory contains the C interface files that define the constants, types, and functions used by the A/UX Toolbox libraries. For additional information, see Appendix F, “C Interface Library.”

`/shlib` This directory contains the shared libraries `libc_s` and `libmac_s`. Shared libraries are discussed in *A/UX Programming Languages and Tools*, Volume 1.

Appendix B, “Toolbox Contents,” lists the full pathnames of all files pertaining to the A/UX Toolbox and briefly describes the function of each file.

How the A/UX Toolbox works

The primary function of the A/UX Toolbox is to allow applications developed for the Macintosh to be used within a UNIX environment. Most of the support code consists of routines built into the Macintosh ROM, supplemented by other routines loaded into memory as necessary.

When an A/UX Toolbox application issues a call to one of the ROM-based routines, the A/UX Toolbox intercepts the call and, as necessary, passes the call either to the ROM routine or to an alternate A/UX Toolbox support routine.

Figure 1-1 illustrates how the two elements of the A/UX Toolbox library interact with the application and the ROM code. For a more detailed description of how the A/UX Toolbox works, see Appendix C, “Implementation Notes.”

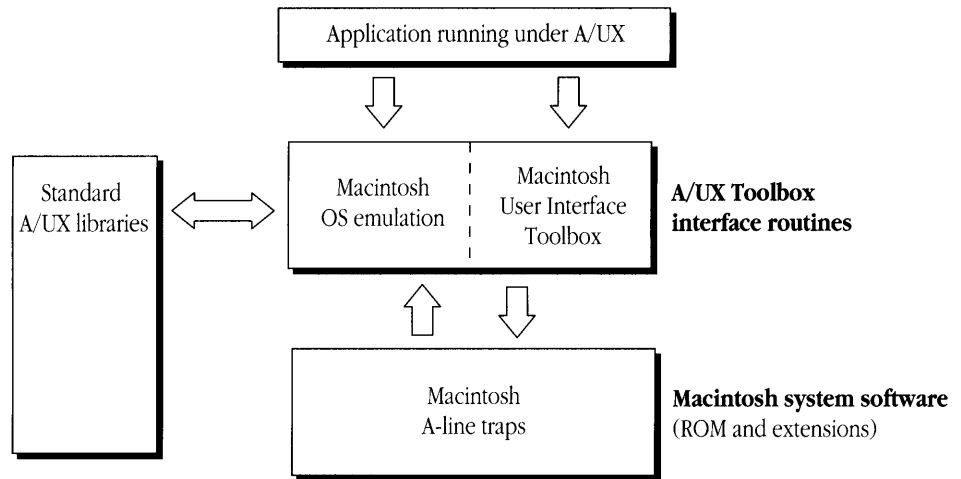


Figure 1-1 Interactions among an application, the A/UX Toolbox, and the ROM code



2 Using the A/UX Toolbox

Application development environments / 2-2

Your application in the A/UX Finder environment / 2-4

Developing an A/UX Toolbox application / 2-6

Building and running the sample programs / 2-10

This chapter describes A/UX Toolbox development environments and some tools for porting applications to A/UX, outlines the procedures for developing A/UX Toolbox applications, and describes the sample programs included with the A/UX Toolbox.

Application development environments

You can develop applications under either the Macintosh OS or A/UX. The A/UX Toolbox lets you run applications and tools under one environment that were developed under the other. Figure 2-1 summarizes the four possible application development and execution paths.

		Execution environment	
		Macintosh	A/UX
Development environment	Macintosh	Develop, debug, and run program with Macintosh tools	Develop and debug program with Macintosh tools Transfer source code to A/UX environment Compile and link to run in A/UX environment
	A/UX	Develop and debug program with A/UX tools or (optional) Macintosh tools Transfer source code to Macintosh environment Compile and link to run in native Macintosh environment	Develop, debug, and run program with A/UX tools or (optional) Macintosh tools

Figure 2-1 Application development and execution environments

You can use whichever development environment best meets your needs. Generally speaking, a Macintosh application is usually developed on the Macintosh side and (if it is a well-behaved Macintosh application) can be executed under the A/UX Finder with few or no changes. A typical case for development on the A/UX side might be that of providing a Finder interface to an existing UNIX application, creating a hybrid application to run on both the A/UX side and the Macintosh side. Although such an application could be developed in either environment, as convenient, an experienced UNIX programmer might prefer to use the A/UX environment.

A/UX Release 3.0 supports two key phases of application development:

- *Porting an application from the Macintosh OS to the A/UX operating system and running it under A/UX* The section “Making A/UX System Calls,” in Chapter 3, outlines a strategy for using A/UX system calls in applications that will run optimally in both Macintosh and A/UX environments. Chapter 4, “Compatibility Guidelines,” and Chapter 5, “A/UX and Macintosh User Interface Toolbox Differences,” provide additional information to assist you in optimizing your applications.
- *Developing a UNIX application under A/UX that exploits the Macintosh user interface tools* The section “Developing an A/UX Toolbox Application,” later in this chapter, outlines the procedures for developing an A/UX Toolbox program. The section “Building and Running the Sample Programs,” later in this chapter, describes the sample programs and makefiles provided as examples. Chapter 3 describes the utilities that support these procedures.

Both Macintosh binary files ported to A/UX and A/UX Toolbox programs developed under A/UX must meet the A/UX Toolbox compatibility requirements. With Release 3.0, these requirements are likely to be met by most applications that meet the standards for current Macintosh OS applications. For details, see Chapter 4, “Compatibility Guidelines.”

Through the A/UX Finder environment, a user can access files in either the UNIX or the Macintosh file systems, and so can an application. You can transfer files between file systems either from within an application or by dragging icons on the desktop. Files transferred between the two file systems undergo certain changes that are generally transparent to users, but of interest to programmers. In addition, because of design differences pertaining to file handling in the two file systems (UNIX file permissions, Macintosh file structure, and so on), transferring a file results in automatic changes in information relating to that file in its new environment. These changes are described in Chapter 6, “File Systems and File Formats.”

Your application in the A/UX Finder environment

Applications must generally be 32-bit clean and must be compatible with System 7 to run in the A/UX Finder environment. (A special 24-bit login environment is provided for backward compatibility.) This section tells you how to ensure that an application is A/UX Finder-friendly. The information in this section extends the information in *Inside Macintosh*, Volume VI, which provides detailed information on System 7 compatibility.

Certain applications cannot run under A/UX because they violate A/UX requirements—for example, by doing direct hardware manipulations or by relying on Macintosh traps or functions that are not supported under A/UX. Information on these matters is given elsewhere in this manual, particularly in Chapters 4 and 5. This section is not concerned with these special requirements, but with how, in general, an application can function better in the A/UX Finder environment. The following strategies will help make your application more compatible with the A/UX Finder:

- **Use the `WaitNextEvent` routine rather than the `GetNextEvent` routine.**

The `GetNextEvent` routine is very unfriendly to the A/UX kernel scheduler. Use `WaitNextEvent`, with timeouts and mouse regions, if at all possible. This routine allows the kernel scheduler to put processes to sleep, improving the efficiency of CPU usage. `WaitNextEvent` also improves responsiveness to the user, because processes are penalized for accumulated CPU time.

- **Perform blocking operations only if unavoidable.**

The `ui_setselect` call is helpful in avoiding blocking. See the next section, “Using the `ui_setselect` Call,” for more information.

- **Set the `'SIZE'` resource higher than you would for System 7. See *Inside Macintosh*, Volume VI, for information on the `'SIZE'` resource.**

When setting the `'SIZE'` resource, allow slightly more memory than would be needed for running under System 7. Running Macintosh OS memory management under A/UX requires some additional overhead.

Using the `ui_setselect` call

If you are porting an application, such as a terminal emulator, that normally blocks on I/O by using `select(2)` for events, you can use the `ui_setselect` function to block in `WaitNextEvent`. This technique gives you a means to break out of `WaitNextEvent` before its timeout. (Use of `WaitNextEvent` allows other applications access to the CPU while you wait.) Effectively, the `ui_setselect` function gives you a way to post a UNIX event. Usage of `ui_setselect` is similar to that of `select(2N)`:

```
ui_setselect(nfds, readmask, writemask, exceptmask)
int nfds, readmask, writemask, exceptmask
```

The call is used before and after `WaitNextEvent`, as follows:

```
ui_setselect(nfds, readmask, writemask, exceptmask) /*set masks*/
WaitNextEvent (...);
ui_setselect(0, 0, 0, 0); /*clear select masks*/
select(nfds, readfds, writefds, exceptfds, 0) /*check I/O*/
```

The `ui_setselect` call causes `WaitNextEvent` to return a null event whenever a `select(2N)` call would succeed, that is, when a file descriptor becomes active. An example of when a `select` call would succeed is the point at which data becomes available to a read file descriptor. The method used with A/UX Release 1.1, which first called `select` and then called `GetNextEvent`, will not work properly with Release 3.0.

Whenever `WaitNextEvent` returns, you must call `select` with a timeout value of 0 to see if I/O is pending on any file descriptors. Using this mechanism you cannot tell why `WaitNextEvent` returned, so you need to call `select` to test if the reason is the driven by Macintosh or UNIX requests.

Calling `ui_setselect` to set the masks effectively adds another event type to the event mask for the `WaitNextEvent` call. Thus, calling this routine a second time to clear the masks prevents a potential problem. If, without the select masks cleared, the application enters a different event loop that does not handle `select` (by calling `ModalDialog`, for example) the event mask will still request an event. In such a case, a null event may be returned to indicate that `select` would return. For the `ModalDialog` example, this means that the update event for the dialog box would not be returned, because it has a lower precedence than the `select` physical event, and the contents of the dialog box would not be drawn.

The `ui_setselect` call is similar in function to `select(2N)`, as documented in *A/UX Programmer's Reference*, with the following exceptions:

- The `ui_setselect` call has no timeout argument.
- The masks for `ui_setselect` are integers; for `select`, they are pointers to integers.
- The `ui_setselect` call does not return the number of active file descriptors.
- Active file descriptors are not passed back in the descriptor masks.
- You can only use the first 32 file descriptors.

Developing an A/UX Toolbox application

This section summarizes the procedures for developing an A/UX Toolbox application under A/UX.

You must be familiar with the general Macintosh program development procedures before you can write a Macintosh-like application under A/UX. If you have never written a Macintosh application, see Appendix A, “Additional Reading,” for suggested references.

Development of an application that uses the Macintosh interface follows two parallel paths: development of source code and development of resources. This section briefly describes the tools provided for developing the two elements. Chapter 3 contains details on the special tools provided with the A/UX Toolbox for support of program development.

The sample programs provided with the A/UX Toolbox illustrate the procedures for compiling and building an A/UX Toolbox application, starting with separate files containing the source code and the uncompiled resources. See the directory `/mac/lib/examples` for the sample programs and the section “Building and Running the Sample Programs,” later in this chapter, for more information.

- △ **Important** Shared libraries are implemented in A/UX Release 3.0. Using shared library code for routines subject to change and development provides a convenient method of supporting future enhancements. You can update and ship the library code, and applications that call upon the shared library will automatically use the current code in the library without having to be recompiled. See *A/UX Programming Languages and Tools*, Volume 1, for more information. △

Developing the source code

You can use the standard A/UX C development environment for developing and debugging an A/UX Toolbox application. Additional enhanced development tools are included in the A/UX Developer's Tools product, available from APDA. (For information on the standard environment, see *A/UX Programming Languages and Tools*, Volumes 1 and 2. For information on the enhanced environment, see *A/UX Development Tools* and *A/UX c89 C*.)

Use the standard C libraries (shared or nonshared) as usual, and incorporate the special A/UX Toolbox components at each step:

■ Writing source code

Include the header file for each Macintosh library you use. (See Appendix F, "C Interface Library," for a list of the available header files.) Check each library's entry in Chapter 5 for any warnings about the A/UX implementation of that library.

Follow the general A/UX compatibility guidelines in Chapter 4 and the general Macintosh programming guidelines in *Inside Macintosh*.

■ Building the application

Adapt the sample makefiles in `/mac/src/examples` to compile and link your application as required by the A/UX Toolbox. The build procedure for an A/UX Toolbox application differs from that for a typical UNIX C program in that you must call in additional libraries and make provisions for Macintosh memory-use conventions. Your build procedure should include these additional steps:

- specifying the pathnames for the include files
- linking to the files that contain the A/UX Toolbox routines, the symbols for Macintosh global variables, and the initialization routine

As demonstrated in the sample makefiles, you can also define a constant to allow for selective compiling of common source code for different execution environments.

For more information on how an application is built and executed, see Appendix C, "Implementation Notes."

Figure 2-2 illustrates how to incorporate the A/UX Toolbox into an application. `appname.c` represents your source file. Use the standard makefile to compile it, using `cc(1)`, and link it, using `ld(1)`. The output is an executable Common Object File Format (COFF) object file.

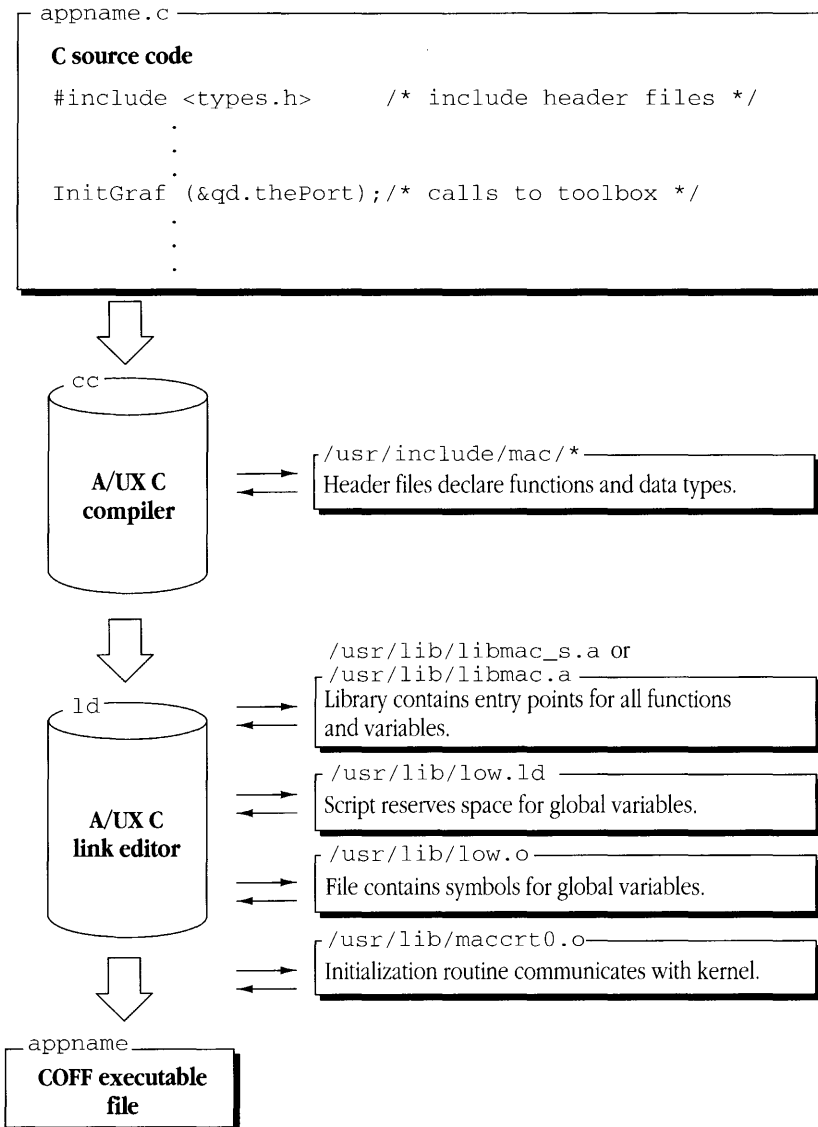


Figure 2-2 Incorporating the A/UX Toolbox into development code

Developing the resource file

The resource editor ResEdit, which allows you to manipulate resources graphically and to copy resources between applications, now runs under A/UX. (Version 2.1 of ResEdit is the preferred version as of this writing.) Using ResEdit is by far the most efficient way to create or manipulate resources on Macintosh systems, including those running A/UX.

Additional tools and utilities for development of resources under the Macintosh OS are available from third-party developers. MPW offers several tools, described in *Macintosh Programmer's Workshop 3.2 Reference*. If you develop your resources in the Macintosh OS, you can transfer the compiled resources into a file of appropriate format for A/UX.

For developers who want to do things the hard way, the A/UX Toolbox includes A/UX versions of the `rez(1)` and `derez(1)` tools, ported from MPW, for compiling and decompiling resources. Figure 2-3 illustrates this resource development path. You will probably want to save a source version of your resource files. You can build resources with ResEdit, then use `derez` on the file to obtain a source file. Should you ever need to, you can then use `rez` on your source file to re-create the resource file.

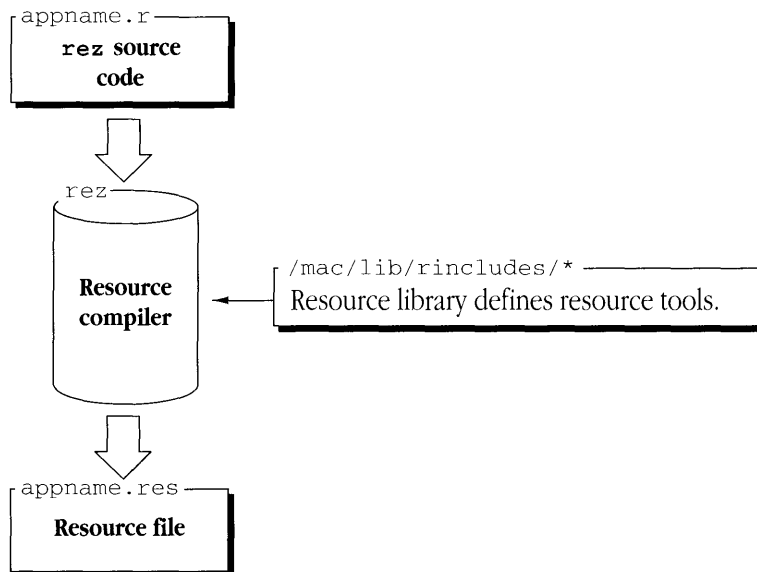


Figure 2-3 Developing a resource file by using `rez`

Appendix E, “Resource Compiler and Decompiler,” documents `rez` and `derez` in detail. The directory `/mac/lib/rincludes` contains the resource type definition files used by `rez` and `derez`.

Building and running the sample programs

Source code for two sample programs is in the directory `/mac/src/example` (provided that you have performed the complete A/UX installation). The directory contains a makefile, a C source file, and a resource file for each of these applications. (See *Inside Macintosh*, Volume I, for a detailed description of resource files and Macintosh application development procedures.)

- The first example is a program that demonstrates basic QuickDraw graphic operations. The relevant files are `qdsamp.c`, a C source file, and `qdsamp.r`, a resource file.
- The second example is a generic application that displays a fixed-sized window in which the user can enter and edit text. The relevant files are `sample.c`, a C source file, and `sample.r`, a resource file.

To build executable code, copy the source files and makefile to the directory you are working in, then enter the `make(1)` command with the name of the demonstration program as an argument. The executable code is put into an executable file, and the resources and header information are put into a resource file. You must explicitly build both files. To build `sample`, for example, enter this command:

```
make sample %sample
```

In this example, the executable file is `sample` and the resource information is in the file `%sample`.

To run one of the sample programs, enter the name of the executable file (`sample`, in this case) on a command line as you would for any other A/UX program, or double-click the program’s icon. The A/UX Toolbox automatically looks for the associated resource file, and uses the two files together so long as they are in the same directory.

The makefile provided with the sample programs illustrates the steps necessary to compile and link a Macintosh application under A/UX. Examine `makefile` in the `/mac/lib/examples` directory. When you are ready to build your own application, you can copy this makefile and adapt it for your program.

Another sample program resides in the `/mac/src/sndDemo` directory. The directory contains a makefile, C source file, resource file, C header file, and sound demo resource. When compiled, the program checks for sampled sound resources in its resource fork and in the System file. It places these resources under a “SampledSynth” menu.

3 A/UX Toolbox Utilities and Extensions

Using the A/UX Toolbox utilities / 3-2

A/UX Toolbox variables / 3-3

Additional trap and routine / 3-4

A/UX Toolbox environment variables / 3-6

Making A/UX system calls / 3-7

The MacsBug debugger under A/UX / 3-11

The dbx debugger under A/UX / 3-13

This chapter discusses some special features of the A/UX Toolbox that support program development in A/UX. The features fall into five broad categories:

- utility programs for controlling the execution environment, converting file formats and attributes, and compiling and decompiling Macintosh resource files
- variables defined in the A/UX Toolbox interface library that let you change how an application is executed
- an additional trap and routine for use in A/UX Toolbox applications
- environment variables for use during debugging
- debuggers that assist in finding problems with applications

Using the A/UX Toolbox utilities

The A/UX Toolbox utilities discussed in this chapter are all in the directory `/mac/bin`, which also contains many other useful utilities and programs. All of the A/UX Toolbox utilities have Commando interfaces that present the choices available when you are using the utilities. Descriptions of the files in the `/mac/bin` directory can be found in `/FILES`. Several of the facilities available in `/mac/bin` are particularly useful in a development environment:

<code>startmac</code> <code>startmac24</code>	<p>These two utilities handle the transition between the standard A/UX environment and an A/UX Toolbox application environment. The <code>startmac</code> utility provides the standard 32-bit A/UX Toolbox environment, while <code>startmac24</code> provides a 24-bit A/UX Toolbox environment. The <code>startmac</code> and <code>startmac24</code> utilities are described in Section 1 of <i>A/UX Command Reference</i>.</p> <p>The 24-bit environment is isolated from the rest of the system; it is provided for testing when you are converting 24-bit applications and for running obsolete 24-bit utilities and tools, such as orphaned compilers. Information on accessing the 24-bit environment is available in <i>A/UX Essentials</i>.</p>
<code>launch</code>	<p>This utility is available for launching a Macintosh binary from the command line of CommandShell. It provides special options and capabilities for a Macintosh binary launched within the A/UX Finder. This utility is not necessary for ordinary launching; you can launch applications by double-clicking their icons, by dragging a document onto their icons, or by opening an executable Macintosh binary. The utility is provided for convenience in launching from CommandShell and for use in application development. The <code>launch</code> utility functions only in the 32-bit environment. The <code>launch</code> utility is described in <i>A/UX Programmer's Reference</i>.</p>
<code>setfile</code>	<p>This utility sets file creator and type, and other attributes. The <code>setfile</code> utility is described in Section 1 of <i>A/UX Command Reference</i>.</p>

<code>changesize</code>	This utility changes the value of the file's 'SIZE' attribute. The 'SIZE' attribute is described in <i>Inside Macintosh</i> , Volume VI.
<code>fcnvt</code>	This utility converts files from and to six formats. The <code>fcnvt</code> utility is described in detail in Section 1 of <i>A/UX Command Reference</i> . For information on file formats, see Chapter 6, "File Systems and File Formats," in this manual.
<code>rez</code> <code>derez</code>	These two utilities are available for compiling (<code>rez</code>) and decompiling (<code>derez</code>) resources. Detailed information is in Appendix E, "Resource Compiler and Decompiler." These utilities are also described in Section 1 of <i>A/UX Command Reference</i> .

Further information on utilities is available from many sources, including on-line and printed manual pages and Commando dialog boxes.

A/UX Toolbox variables

The A/UX Toolbox interface library, in the file `/usr/lib/libmac.a`, defines two variables:

<code>dontForeground</code>	<p>This variable specifies whether or not the program runs only in the background. If it is set to 1, the program runs only in the background.</p> <p>To set <code>dontForeground</code> to 1, include this line in your program:</p> <pre>int dontForeground = 1;</pre>
<code>noCD</code>	<p>This variable sets the current directory. If it is set to 1 in a program, the current directory is the directory from which the user ran the program. Otherwise, the current directory is the directory in which the program resides.</p> <p>To set <code>noCD</code> to 1, include this line in your program:</p> <pre>int noCD = 1;</pre>

Additional trap and routine

The A/UX Toolbox interface library includes one additional trap and one additional routine for use in A/UX Toolbox applications: the `AUXDispatch` trap and the `select` routine. These are described in the sections that follow.

`AUXDispatch` trap

The `AUXDispatch` trap is a multipurpose call that supports some A/UX-specific extensions to the A/UX Toolbox. You should invoke this call only after using the `Gestalt` function to determine that the application is running under A/UX. Information on the `Gestalt` facility is available in *Inside Macintosh*, Volume VI.

The definitions for the `AUXDispatch` trap are in the header file `aux.h`, found in `/usr/include/mac`. (See “Definitions for `AUXDispatch`” in Appendix F.) The header file provides a syntax compatible with MPW C version 3.2. `AUXDispatch` uses this syntax:

```
AUXDispatch (selector, p)
    short selector ;
    char  *p ;
```

The function of `AUXDispatch` depends on the placeholder *selector*, which can be one of these values:

<code>AUX_HIGHEST</code>	Returns the highest available selector (for support of future releases, which may provide more selectors). With this selector, the pointer <i>p</i> is not used.
<code>AUX_GET_ERRNO</code>	Gets a pointer to <code>errno</code> , which is linked to your program through the standard C library. <code>AUXDispatch</code> puts the address of <code>errno</code> in the address that you specify with the pointer <i>p</i> .

AUX_GET_PRINTF	Gets a pointer to the <code>printf(3S)</code> routine, which is linked to your program through the standard C library. AUXDispatch puts the address of <code>printf</code> in the address that you specify with the pointer <i>p</i> .
AUX_GET_SIGNAL	Gets a pointer to the <code>signal(3)</code> routine, which is linked to your program through the standard C library. AUXDispatch puts the address of <code>signal</code> in the address that you specify with the pointer <i>p</i> .
AUX_GET_TIMEOUT	Returns a time period, in clock ticks, indicating when the next Macintosh device driver will need to obtain processor time through the <code>WaitNextEvent</code> routine. See <i>Inside Macintosh</i> , Volume VI, for a description of <code>WaitNextEvent</code> . With this selector, the pointer <i>p</i> is not used.
AUX_SET_SELRECT	Defines a rectangle that the user interface device driver will use to monitor mouse movements for the <code>select(2N)</code> system call. For an explanation of the <code>select</code> call, see “Using <code>select</code> to Monitor A/UX I/O Activity and Macintosh Events,” later in this chapter. With this selector, the pointer <i>p</i> points to the specified rectangle.
AUX_CHECK_KIDS	Checks for the existence of child processes, returning 1 if child processes exist for the specified process and 0 if not. With this selector, you specify the process to be checked for child processes by passing the pointer <i>p</i> to the process ID.
AUX_POST_MODIFIED	Posts an event, with modifiers. With this selector, you pass the pointer <i>p</i> to the event record.
AUX_FIND_EVENT	Searches the event queue for an event. With this selector, you pass the pointer <i>p</i> to a <code>FindEvent</code> structure (mask and pointer to an event record).

Using `select` to monitor A/UX I/O activity and Macintosh events

If you are writing an A/UX Toolbox application that will run only under A/UX, you can use the `select(2N)` system call to monitor not only standard A/UX I/O activity but also Macintosh events.

The `select` call examines a set of file descriptors that you specify through bit masks. The A/UX Toolbox provides a user interface device driver, `/dev/uinter0`, to handle communications between the A/UX Toolbox library and the kernel. The file descriptor `udevfd` is opened to `/dev/uinter0`. To include Macintosh events in the list of I/O activity to be monitored, include `udevfd` in the masks you pass to `select`.

You can use a combination of the `select` system call and the `AUXDispatch` call to expand the definition of a Macintosh event to include movement of the mouse outside a specified rectangle. (Ordinarily, mouse motion without the pressing or release of the mouse button is not an event.) First, issue the `AUXDispatch` call, using the `AUX_SET_SELRECT` selector and passing a pointer to the rectangle. `AUXDispatch` passes the rectangle to the user interface device driver. In subsequent `select` calls, include the `udevfd` descriptor in your masks. `select` will then wake up your program if there is a Macintosh event or other specified event pending, if the mouse moves out of the specified rectangle, or if the timer expires. Once `select` reports activity through the user interface device driver, you must call `GetNextEvent` to retrieve the event.

This sequence (`AUXDispatch` followed by `select`) is an alternative for A/UX Toolbox programs that cannot use the `waitNextEvent` trap (described in the section “Event Manager, Toolbox” in Chapter 5).

A/UX Toolbox environment variables

The A/UX Toolbox uses a number of environment variables to modify its actions under certain circumstances. Most of these variables are useful only during program development and debugging.

A/UX Toolbox environment variables are set and read like other environment variables. (For information on environment variables, see `environ(5)` in *A/UX Programmer's Reference*.) This section lists the environment variables used by the A/UX Toolbox and their functions.

TBCORE	If this variable is set to a nonzero value, the A/UX Toolbox causes a core dump if a fatal error occurs. If this variable is not set, the A/UX Toolbox displays a message and exits when a fatal error occurs. One example of the cause of a fatal error during development of A/UX Toolbox applications is attempting to execute an unimplemented A-line trap.
TBRAM	If this variable is set to a nonzero value, the ROM code is copied into a memory segment when a program is run. This variable lets you set a breakpoint in the ROM code for debugging.
TBSYSTEM	This variable contains the A/UX pathname of the directory that contains Macintosh system files. The default setting is <code>/mac/lib/SystemFiles</code> .
TBTRAP	If this variable is set to a nonzero value, the system writes debugging information to standard error every time an A-line trap is executed.
TBWARN	If this variable is set to a nonzero value, the system writes a warning message to standard error when certain error conditions are detected. These messages generally report that something unusual but not fatal has happened. Developers may want to set <code>TBWARN</code> in <code>.login</code> or <code>.profile</code> .

Making A/UX system calls

This section describes a strategy for building an application under the Macintosh OS by using A/UX system calls, resulting in an application that can be executed in both A/UX and Macintosh environments.

◆ **Note** The A/UX Developer's Tools product, available from APDA, already contains all of the A/UX system calls in MPW format. If you are using this product, you will not need to perform the first two steps of the following procedure. ◆

The strategy described in this section is intended for applications that need to perform functions available through the Macintosh OS or the User Interface Toolbox but not available under A/UX. You can write an application that uses the required function(s) when running under the Macintosh OS but uses alternative code, including A/UX system calls, when running under A/UX. Use Gestalt to determine under which system the application is running.

The basic procedure is to translate the A/UX system calls into assembly-language routines and to make those routines available to the compiler under the Macintosh OS. Specifically, you can use A/UX system calls in an application that will run under both environments by following these steps:

1 Determine the assembly-language sequence that is generated by the A/UX compiler when it encounters the system call you want to use.

- a. Write a program that uses the call. If you want to use `open(2)`, for example, you could start with this program:

```
main()
{
    int fd;
    fd = open("fred",2);
}
```

- b. Compile the program in the A/UX environment.
- c. Use the debugger `adb(1)` to disassemble the program. The `open` call, for example, results in this disassembled code:

```
open:
    mov.l    &0x5, &d0
    trap    &0x0
    bcc.b    noerror
    jmp     cerror%
noerror:
    rts
cerror%:
    mov.l    %d0, errno
    mov.l    &-1, %d0
    mov.l    %d0, %a0
    rts
```

2 In your Macintosh development environment, create an assembly-language routine that performs the same functions.

Give this routine a unique name. (The Macintosh OS equivalent to the `open` call, for example, might be `auxopen`.)

3 Insert the call conditionally into your application.

Use Gestalt to determine if A/UX is running; the following code segment shows you how. Gestalt will return the version of A/UX if it is currently running. (The result is placed into the lower word of the response parameter.) If A/UX is not running, Gestalt returns `gestaltUnknownErr`. If Gestalt is not running, the glue code returns an error. If you get an error, check the `HWCfgFlags` low-memory global, as shown in the following code segment.

◆ **Note** Checking that Gestalt is running is required only if you want your application to be backward-compatible with Macintosh systems running a Macintosh OS version prior to System 7. All systems running A/UX Release 2.0 or later releases implement the Gestalt facility. ◆

The following MPW code segment returns the version of A/UX currently running, or 0 if it is not running. This code relies on Gestalt glue code available in MPW version 3.2 and later versions.

```
/*
 *   getAUXVersion.c
 *
 *   Copyright © 1990 Apple Computer, Inc.
 *
 *   This file contains routines to test if an application
 *   is running on A/UX. If the Gestalt trap is available,
 *   it uses that; otherwise it falls back to HWCfgFlags,
 *   which will work on all A/UX systems.
 */
```

(continued) ➔


```

#include <Types.h>
#include <GestaltEqu.h>
#define HWCfgFlags 0xB22 /* Global used to check if A/UX
is running */

/*
 * getAUXVersion -- Checks for the presence of A/UX by
 * whatever means is appropriate. Returns the major
 * version number of A/UX (i.e., 0 if A/UX is not present,
 * 1 for any 1.x.x version 2 for any 2.x version, etc.
 * This code should work for all past, present and future
 * A/UX systems.
 */
short getAUXVersion ()
{
    long    auxversion;
    short   err;
    short   *flagptr;

    /*
     * This code assumes the Gestalt glue checks for the
     * presence of the _Gestalt trap and does something
     * intelligent if the trap is unavailable, i.e.
     * return unknown selector.
     */
    auxversion = 0;
    err = Gestalt (gestaltAUXVersion, &auxversion);
    /*
     * If gestaltUnknownErr or gestaltUndefSelectorErr
     * was returned, then either we weren't running on
     * A/UX, or the _Gestalt trap is unavailable so use
     * HWCfgFlags instead. All other errors are ignored
     * (implies A/UX not present).
     */
}

```

```

if (err == gestaltUnknownErr || err ==
    gestaltUndefSelectorErr) {flagptr = (short *)
    HWCfgFlags; /* Use HWCfgFlags */
    if (*flagptr & (1 << 9))
        auxversion = 0x100; /* Do Have A/UX; assume
                               version 1.x.x */
    }
/*
 * Now right shift auxversion by 8 bits to get major
 * version number
 */
auxversion >>= 8;
return ((short) auxversion);
}

```

Once you have the A/UX version, it is a simple matter to use the optimal call for the execution environment. For example:

```

#include </usr/include/sys/uio.h>
{
if auxversion >= 2
    then auxread(4, *tempbuff, 512)
    else FSread(4, 512, tempbuff, )
}

```

The MacsBug debugger under A/UX

The MacsBug debugger is available from APDA for use within the A/UX Toolbox environment. (Version 6.2 or a later version of MacsBug is required.) MacsBug comes with a reference manual. For APDA contact information, see “Information Sources” in Appendix A.

When used with the A/UX Toolbox, MacsBug does not underlie the entire system, as it does when used with the Macintosh OS. Pressing the hardware programmer switch when A/UX is running places you into the UNIX kernel debugger (if present).

To install MacsBug, place it in your System Folder. (By default, the System Folder is the directory `/mac/sys/System Folder`.) Placing MacsBug in this directory is the equivalent of placing it in the System Folder under the Macintosh OS. MacsBug will automatically install itself the next time you log in to A/UX.

MacsBug is invoked when the system encounters an exception error. You can force entry into MacsBug at any time by pressing `COMMAND-CONTROL-I`.

Once in MacsBug, you can use MacsBug commands to examine values, step through code, attempt recovery, and so forth, as with any debugger.

The combination keypress `COMMAND-CONTROL-E` exits the A/UX Toolbox and logs you out. This command can be useful for getting out of a hung system. It does a general tidying up of the system (closing open files, and so forth) before logging you out. If you press this key combination when MacsBug is not installed, a similar logout takes place, but it is not so tidy. (Open files may become corrupted, for example.)

When you are in MacsBug, many commands are available. Here are several:

<i>Command</i>	<i>Result</i>
<code>g</code>	Go; continues from current location.
<code>rs</code>	Restart; actually logs you out.
<code>rb</code>	Reboot; actually logs you out.
<code>es</code>	Exit to shell; exits current application. This is not a good way to exit. Low-memory globals are left in an indeterminate state, and after a while strange things start to happen to your other applications.
<code>dm curapname</code>	Displays current application name. The current application may not be what you think it is; it is worth trying this command before you exit the current application.
<code>help</code>	Lists the commands available.
<code>il</code>	Disassembles from the current instruction pointer.
<code>sc</code>	Stack crawl; shows a backtrace of calls to help discover what broke the current application.

Additional information on MacsBug can be found in *MacsBug 6.2 Reference and Debugging Guide* and in *Debugging Macintosh Software with MacsBug*. (See Appendix A for bibliographic information.)

The `dbx` debugger under A/UX

The `dbx` debugger is included for use within the A/UX Toolbox environment. This debugger, in conjunction with MacsBug, allows you to efficiently debug applications that make A/UX system calls.

Here are a few of the commands available with `dbx`:

<code>trace</code>	Prints tracing information when the program is executed. A number is associated with the command. You can use this number with the <code>delete</code> command to turn the tracing off.
<code>stop</code>	Stops execution when the given line is reached, procedure or function is called, variable is changed, or condition becomes true. Execution can be resumed with the <code>cont</code> command.
<code>status</code>	Prints the currently active <code>trace</code> and <code>stop</code> commands.
<code>delete</code>	Removes the traces or stops corresponding to the given numbers. The numbers associated with traces and stops are printed by the <code>status</code> command.
<code>cont</code>	Continues execution from where the process stopped. If a signal is specified, the process continues as though it had received the signal. Otherwise, the process continues as though it had not been stopped.
<code>step</code>	Executes one source line.
<code>next</code>	Executes up to the next source line. The difference between <code>next</code> and <code>step</code> is that if the next line contains a call to a procedure or function, the <code>step</code> command stops at the beginning of that block, whereas the <code>next</code> command does not.

Additional information on `dbx` can be found in *A/UX Programming Languages and Tools*, Volume 1.



4 Compatibility Guidelines

Introduction / 4-2

Differences in execution environments / 4-2

Differences in C compilers / 4-10

Differences in language conventions / 4-11

This chapter discusses various requirements for compatibility between application code that uses the Macintosh User Interface Toolbox and code that uses the A/UX Toolbox. This information, combined with the information available in *Inside Macintosh*, tells you what you need to consider when porting Macintosh applications to take full advantage of the A/UX environment.

Introduction

To run the same code in both the Macintosh OS and UNIX environments, you must make provisions for a number of compatibility issues:

- differences between the Macintosh OS and UNIX execution environments
- differences between the C compilers used in A/UX and those used in other Macintosh OS development environments
- differences between the C language typically used in A/UX and the Pascal language used by the Macintosh ROM routines

For details about the A/UX implementation of the Macintosh ROM code, see Chapter 5, “A/UX and Macintosh User Interface Toolbox Differences,” and Appendix C, “Implementation Notes.”

Differences in execution environments

The Macintosh OS was designed as a single-user system. Individual applications and the various libraries that support the Macintosh user interface can have much more control over the system than individual processes are allowed in UNIX.

In UNIX, the kernel arbitrates all access to hardware, including memory allocation. Only the kernel can use the hardware instructions of the MC680x0 microprocessor.

Like the current Macintosh OS in System 7, A/UX uses virtual memory. A/UX 3.0 allows control of virtual memory through the Memory control panel. While the A/UX virtual memory implementation is completely different than that used in System 7, this is transparent to the user and programmer.

This section lists the compatibility issues that result from the differences between the Macintosh OS and UNIX execution environments. This section augments the Macintosh programming guidelines provided in *Inside Macintosh*. To ensure that your code runs under both the Macintosh OS and A/UX, follow both the rules outlined here and the compatibility guidelines in *Inside Macintosh*, Volume VI.

Sometimes a program must perform differently depending on whether it is running on an MC68020-based Macintosh or an MC68030-based Macintosh and whether it is running under the native Macintosh OS or under A/UX. Use the Gestalt facility to determine which operating system is currently running. See the section “Making A/UX System Calls,” in Chapter 3, for an example using Gestalt.

32-bit address violations

A/UX uses all 32 bits of an address, but the Macintosh OS prior to System 7 used only the low-order 24 bits of an address. Historically, both the User Interface Toolbox and a number of application programs used the high-order 8 address bits for storing additional information. Present and future Macintosh OS applications must now be 32-bit clean in order to fully use the system capabilities. In the A/UX system, a Macintosh application must be 32-bit clean to run under the A/UX Finder. (A 24-bit environment is provided by means of a special login so that programmers making an application 32-bit clean can test in both environments. For information on how to log in to A/UX in 24-bit mode, see *A/UX Essentials*.)

To create 32-bit clean applications, use the Memory Manager in a safe manner. Adhering to the following guidelines will help you avoid many common problems as well as ensuring that your applications are 32-bit clean:

- Use Memory Manager operations for Memory Manager functions. Make no assumptions about the contents of Memory Manager structures. Do not set bits directly in these structures or manipulate them directly.
- In particular, never make your own handles; use `NewHandle`.
- Check every returned handle or pointer to ensure that it is not `NIL`. A `NIL` handle may indicate that a memory allocation failed or that a requested resource could not be found.
- Before using a handle marked purgeable, make sure that the handle is not empty.

As an aid to upgrading old applications, here is a list of (deplorable) practices once common in Macintosh applications that violate 32-bit address requirements:

- *Creating or using fake handles* A fake handle is one that was made not by the Memory Manager (the `NewHandle` function), but by the program (a pointer to a pointer). The Memory Manager has its own style of making handles, and a fake handle can cause trouble.
- *Use of direct access to the flag bits on relocatable blocks* The Macintosh OS used to store the flag bits, `Lock`, `Purge`, and `Resource`, in the high-order bits of a block's master pointer. The A/UX Toolbox stores these flags elsewhere. Setting high-order bits in the master pointer invalidates the address. If your application uses a `bset` instruction or moves bytes to change these flags, change your code to use the appropriate Memory Manager routines instead. See "The Memory Manager" in *Inside Macintosh*, Volumes II, IV, and VI.
- *Use of application-specific flags* Some applications use the high-order bits of addresses to store their own flags. This practice invalidates the address in A/UX (and in System 7).
- *Use of direct access to window and control variant codes* The Macintosh OS formerly stored the variant code for a window or control in the high-order bits of the handle to the definition procedure, which is located in the window or control record. Applications rarely access these codes, but custom definition procedures sometimes do. In A/UX, the variant codes are stored elsewhere. You can read them with the `GetWVariant` and `GetCVariant` calls. (For more information on these calls, see *Inside Macintosh*, Volume VI.)

Privileged microprocessor instructions

The A/UX Toolbox is run by an A/UX process in MC680x0 user mode. Therefore, privileged processor instructions are not available within an A/UX Toolbox application. However, commonly used privileged instructions are emulated by the A/UX kernel. This emulation provides approximately the same functionality for the A/UX environment, though the emulator executes far more slowly. Table 4-1 lists the status of all MC68020 and MC68030 privileged instructions.

Table 4-1 Privileged microprocessor instructions within the A/UX Toolbox

Instruction	Register and addressing modes supported
ANDI to SR	All
EORI to SR	All
FRESTORE	(An)+ (An) (d ₁₆ ,An) Only null and idle frames are supported.
FSAVE	-(An) (An) (d ₁₆ ,An) Only null and idle frames are supported.
MOVE from SR	Dn (An) (An)+ -(An) (d ₁₆ ,An) xxx.16 xxx.32
MOVE to SR	Dn (An) (An)+ -(An) (d ₁₆ ,An) xxx.16 xxx.32
MOVE USP	None
MOVEC	CACr supported on a per process basis; other control registers may be accessed, but no action is taken.
MOVES	None
ORI to SR	All
RESET	None
RTE	Only type 0 and type 2 fault frames are supported.
STOP	None
TAS	None

As shown in Table 4-1, the instructions that manipulate the status register are supported. A special exception handler in the kernel emulates these instructions so that they manipulate a virtual status register established for each process instead of the processor status register. The exception handler can accommodate all status-register instructions that are generated by calls to standard A/UX Toolbox routines. If you are writing assembly code, you can use the `ANDI`, `EORI`, and `ORI` instructions and the simple addressing modes of the `MOVE from SR` and `MOVE to SR` instructions. Table 4-1 lists the supported addressing modes.

You can use assembly-language routines to change any of the bits in the virtual status register, including the priority bits. When the priority is any value higher than 0, all A/UX signals are blocked.

Hardware processor instructions are available to device drivers and other software in the kernel.

Direct hardware access

In A/UX, only the kernel is allowed direct access to the hardware. Therefore, applications cannot bypass the A/UX Toolbox routines and manipulate hardware directly to perform custom functions or save execution time. This limitation has these implications:

- *Serial port access* You cannot access the serial port through the Serial Communications Controller (SCC) registers.
- *Disk drive access* Copy-protection schemes that use direct access to the disk drive controller chip do not work under A/UX.
- *Hardware exception vectors* The low-memory CPU exception vectors are not accessible from within an A/UX user process.
- *Macintosh global variables* Not all of the Macintosh low-memory global variables are valid in A/UX. In general, variables related to hardware are not supported. QuickDraw and Window Manager globals are accessible, because they are not hardware-specific. The screen is directly accessible by an application. Appendix D, “Low-Memory Global Variables,” lists the low-memory global variables supported in A/UX.

Because all input/output and processor-allocation functions are performed through the A/UX kernel, the A/UX Toolbox libraries themselves do not have as much control over the system as do their counterparts in the Macintosh environment.

The standard Macintosh environment provides the Vertical Retrace Manager to handle the scheduling and execution of tasks during the vertical retrace interrupt, and the Time Manager to schedule routines that require precise timing. (These managers are described in *Inside Macintosh*, Volume VI.) The A/UX Toolbox implementation of these managers is built on the A/UX signal mechanism. Depending on the activities of other processes, routines scheduled to be run by either of these managers might be delayed. Even if no other processes are active, the A/UX Time Manager provides coarser granularity than its Macintosh counterpart. For more information, see “Vertical Retrace Manager” and “Time Manager” in Chapter 5.

An application that demands more precise timing probably requires a custom A/UX device driver. See *Building A/UX Device Drivers* in the A/UX Device Drivers Kit, available through APDA, for information on writing device drivers.

You can use standard A/UX device drivers to manage external devices, but programs that use A/UX device drivers are not portable to the Macintosh OS. For a strategy to include A/UX system calls in applications that are intended to run under both the Macintosh OS and the A/UX operating system, see “Making A/UX System Calls” in Chapter 3.

Newline characters

The A/UX Toolbox supports the transfer of files between the Macintosh and A/UX environments. When a user or an application transfers an unformatted ASCII text file between the two environments, certain changes occur automatically. One change may mask a simple, but important, difference in conventions: how the newline character is defined.

In the Macintosh environment, lines are terminated with a return character, represented by the ASCII value 0x0D. In A/UX, lines are terminated with a line-feed character, represented by the ASCII value 0x0A.

This difference is often masked by automatic conversion, which changes newline characters when a file is moved between the environments. A text editor working with a text file in its own environment will find the appropriate newline character in the file, even if the file originated in the other environment. An A/UX utility processing a text file that originated on the Macintosh side of the system and that is now on the A/UX side will find the expected newline characters.

Automatic newline conversion is performed for Macintosh files identified as text files and for A/UX files that are determined to be text or shell script files. See “Text Files” and “Automatic Conversion,” in Chapter 6, for more detailed information on these topics.

This difference is also masked from the programmer by the C language’s newline character (`\n`), which is translated differently in the two environments.

When sending multiple-line strings to the Dialog Manager or any of the A/UX Toolbox managers that receive strings, remember that these managers require the Macintosh newline termination. As with the other cases described here, if the file is on the Macintosh side of the system, then the correct termination character will be present.

You should be aware of this issue, as you may encounter subtle difficulties involving the two newline conventions. Automatic conversion of newline characters occurs only for files known to be text files. A file that is actually a text file may be transferred between environments without being identified as a text file. A file that is not a text file, but which contains text, will not have newlines converted. For example, a resource file contains text and nontext matter. No conversion is done for resource files. A binary file transferred between the two environments has its original newline convention when running in the new environment, which may affect both the output of text and the processing of input text. An application intended for execution in both environments may need to determine its execution environment and select the desired newline character accordingly.

File Manager

The A/UX File Manager is fully supported for all volumes except “/”. The “/” volume is almost fully supported. For details, see “File Manager ” in Chapter 5. Chapter 6, “File Systems and File Formats,” provides details on what happens when files are transferred between the Macintosh and A/UX environments.

Memory Manager

The A/UX Memory Manager supports all access routines in the same way as does the Macintosh OS Memory Manager. The A/UX Memory Manager supports these routines within a virtual memory environment. Virtual memory has practical limits and performance limits. On a practical level, the amount of Macintosh virtual memory (specified by means of the Memory control panel) should be set no higher than twice the size of physical memory. In general, performance degrades to an unacceptable level unless all of the memory that is actively being used fits into physical memory.

International character support

The Script Manager is supported in its entirety; however, a caution applies to use of international character sets within the A/UX environment. In brief, if you have a Macintosh application that supports international character sets, the application should run appropriately under A/UX. However, if you attempt to process international character sets within the A/UX environment by using UNIX utilities, you are likely to encounter difficulties. The kernel itself is 8-bit clean, but this is not necessarily true for the hundreds of utilities and shell scripts furnished as part of A/UX (or any UNIX system), which were developed over the years by many different people. Such utilities and scripts may process characters as though implemented in 7 bits and, when processing text, may make assumptions that do not hold true for international character sets.

Differences in C compilers

This section lists the known differences between the A/UX C compilers and the MPW C Compiler. These differences affect you if you are writing source code that you plan to compile separately in the two environments. If you are using a different C compiler for Macintosh program development, consult your software vendor.

- *Zero-length-array warnings* The A/UX C compilers `cc` and `c89` generate warnings when they encounter zero-length arrays, which appear frequently in the A/UX Toolbox header files. The `c89` compiler is an ANSI-compliant C compiler contained in the A/UX Developer's Tools product, available from APDA.
- *Newline characters* The newline character is the return character (ASCII value 0x0D) in MPW C and the line-feed character (ASCII value 0x0A) in A/UX C.
- *Pascal function types* MPW C has an `extern pascal` function type that can be used for calling most of the ROM routines. To use these functions, an A/UX C program must use intermediate assembly-language “glue,” that is, routines that rework the C call into a form understandable by the ROM.

The A/UX Toolbox provides assembly-language transformation routines (glue routines) for most ROM calls in the files `/usr/lib/libmac.a` and `/usr/lib/libmac_s.a`. The second file, `libmac_s.a`, is a shared-library version of the first, and can be used on the `compile` or `link-edit` command line in exactly the same fashion as `libmac.a`. The shared-library version saves some space in an application binary, and has the advantage of always referring to the latest version of the file. Shared libraries are discussed in Chapter 7 of *A/UX Programming Languages and Tools*, Volume 1. See *A/UX Development Tools* for an explanation of how to call functions by using the Macintosh Toolbox library.

However, if you want to create your own definition functions or filter functions, you must generate your own assembly-language glue. The A/UX Developer's Tools product, available from APDA, provides most of the glue routines you may need. See Appendix C, “Implementation Notes,” for details about the requirements of the Pascal routines. See *A/UX Programming Languages and Tools*, Volume 1, for information on the A/UX assembler `as`.

- *Enumerated types* In MPW C, enumerated types can be 8, 16, or 32 bits long, depending on the range of possible values. In A/UX C, enumerated types are 32 bits long, unless packed in structures using bitfields. A/UX C does not treat an enumerated type as an integer in all cases; MPW C does.
- *Functions returning pointers* MPW C places the return value in register D0; A/UX C places the value in both A0 and D0.

Differences in language conventions

Most of the Macintosh ROM routines follow Pascal conventions for storing strings, passing structures, pushing parameters on the stack, and returning function results. These conventions differ from standard C conventions. (See Appendix C, “Implementation Notes,” for details about the differences between Pascal and C conventions.)

The A/UX Toolbox interface to the ROM routines includes conversion code that takes care of most of these incompatibilities. Since Release 1.1, A/UX has provided two versions of all routines that take parameters of type string or type point or that return values of type string. One version, spelled as the routine appears in *Inside Macintosh*, always uses Pascal-format strings and Pascal point-passing conventions. The second version, spelled in all lowercase letters, uses C-format strings and points. The lowercase version converts input parameters from C format to Pascal format before passing them to the ROM, and converts string return values back to C format.

An alphabetical list of all calls in the C interface libraries, “Calls in Alphabetical Order,” is provided in Appendix F. You can consult this list to determine whether an alternative version of a call is available, because lowercase and mixed-case versions of a call name sort together.

5 A/UX and Macintosh User Interface Toolbox Differences

About the Macintosh interface library / 5-2

Calls patched under A/UX / 5-31

Calls not supported under A/UX / 5-34

This chapter describes the differences between the A/UX Toolbox software libraries and those of the standard Macintosh User Interface Toolbox and Operating System. For each chapter in *Inside Macintosh* that describes a software library (typically called a “manager” of the feature it supports), this chapter contains a section describing the A/UX implementation of that library. The sections in this chapter appear in alphabetical order, not *Inside Macintosh* order. This chapter also contains an alphabetical listing of calls not supported under A/UX.

For a general description of how each library works, see the corresponding chapter in *Inside Macintosh*. For a detailed list of the constants, types, and functions used by each library, see Appendix F, “C Interface Library.”

About the Macintosh interface library

Most of the Macintosh User Interface Toolbox libraries, such as the Menu Manager and the Window Manager, work the same way in the A/UX Toolbox as they work in the standard Macintosh User Interface Toolbox. Some A/UX Toolbox libraries are different from their Macintosh OS counterparts because they replace parts of the Macintosh OS. This chapter provides detailed discussions of the differences between the two implementations. Appendix C, “Implementation Notes,” describes some additional implementation details.

Some of the standard Macintosh OS libraries, such as the SCSI Driver, are not implemented in the A/UX Toolbox. Refer to the A/UX Device Drivers Kit, available from APDA, for documentation and examples of source code for all A/UX device drivers.

Table 5-1 summarizes the status of the various ROM libraries at the time A/UX Release 3.0 was distributed.

Table 5-1 Status of User Interface Toolbox and Macintosh OS libraries in the A/UX Toolbox

ROM library	Supported?
32-Bit QuickDraw with Color QuickDraw	Yes
Alias Manager	Yes
Apple Desktop Bus	No
Apple Event Manager	Yes
AppleTalk Manager	Yes
Binary-Decimal Conversion Package	Yes
Color Manager	Yes
Color Picker Package	Yes
Control Manager	Yes
Data Access Manager	Yes
Deferred Task Manager	Yes
Desk Manager	Not needed
Desktop Manager	Yes
Device Manager	Yes
Dialog Manager	Yes

(continued) ➡

Table 5-1 Status of User Interface Toolbox and Macintosh OS libraries
in the A/UX Toolbox (*continued*)

ROM library	Supported?
Disk Driver	Yes
Disk Initialization Package	Yes
Edition Manager	Yes
Event Manager, Operating System	Partially
Event Manager, Toolbox	Yes*
File Manager	Mostly
Floating-Point Arithmetic and Transcendental Functions Packages	Yes*
Font Manager	Yes
Gestalt Manager	Yes
Graphics Devices Manager	Yes
Help Manager	Yes
International Utilities Package	Yes
List Manager Package	Yes
Memory Manager	Yes
Menu Manager	Yes
Notification Manager	Yes
Package Manager	Yes
Palette Manager	Yes
Picture Utilities Package	Yes
Power Manager	Not needed
PPC Toolbox	Yes
Printing Manager	Yes
Process Manager	Yes
Resource Manager	Yes
Scrap Manager	Yes
Script Manager	Yes
SCSI Manager	No
Segment Loader	Partially

(continued) ➡

Table 5-1 Status of User Interface Toolbox and Macintosh OS libraries in the A/UX Toolbox (*continued*)

ROM library	Supported?
Serial Driver	Yes
Shutdown Manager	Yes
Slot Manager	Partially
Sound Manager	Mostly
Standard File Package	Yes
Startup Manager	Not needed
System Error Handler	Yes*
TextEdit	Yes
Time Manager	Yes*
Utilities, Operating System	Partially
Utilities, Toolbox	Yes
Vertical Retrace Manager	Partially
Window Manager	Yes

* All calls are implemented in the A/UX Toolbox, but functionality is not identical to that in the Macintosh User Interface Toolbox. See the discussions later in this chapter for details.

The C interfaces to the standard Macintosh libraries are defined in a set of header files shipped in the directory `/usr/include/mac`. Include the header file for each library you use in your C program to declare the definitions, types, and functions provided by the library. Appendix F, “C Interface Library,” contains a list of calls available through the libraries, including the header to include for each call. Table F-1 lists the header filenames together with their library titles.

32-Bit QuickDraw with Color QuickDraw

The A/UX Toolbox 32-Bit QuickDraw is identical to the Macintosh OS 32-Bit QuickDraw. Color QuickDraw is included in 32-Bit QuickDraw.

See *Inside Macintosh*, Volumes I, V, and VI, for a description of QuickDraw and Color QuickDraw, supplemented by the APDA document on 32-Bit QuickDraw. See “32-Bit QuickDraw With Color QuickDraw” in Appendix F for the A/UX C interface.

Alias Manager

The A/UX Alias Manager is identical to the Macintosh OS Alias Manager. See *Inside Macintosh*, Volume VI, for a description of the Alias Manager.

Apple Desktop Bus

The A/UX Toolbox does not support the Macintosh OS Apple Desktop Bus. Source code for the UNIX ADB device driver can be found in the A/UX Device Drivers Kit, available from APDA.

Apple Event Manager

The A/UX Apple Event Manager is identical to the Macintosh OS Apple Event Manager. See *Inside Macintosh*, Volume VI, for a description of the Apple Event Manager.

AppleTalk Manager

All AppleTalk calls and protocols are supported under A/UX 3.0.

AppleTalk printing operations are available either through direct AppleTalk calls in Macintosh binary programs or through calls to the Printing Manager under A/UX. See “Printing Manager” or “Print Traps” in Appendix F for the A/UX C interface.

Other AppleTalk calls are available (at the program level) as UNIX system calls and Macintosh binary calls. The equivalent A/UX C header files are in the library `/usr/include/at`.

See *Inside Macintosh*, Volumes II, IV, V, and VI, for a description of the Macintosh OS AppleTalk Manager. See *A/UX Network Applications Programming* for a description of the A/UX AppleTalk Manager.

Binary-Decimal Conversion Package

The A/UX Toolbox Binary-Decimal Conversion Package is identical to the Macintosh OS Binary-Decimal Conversion Package.

See *Inside Macintosh*, Volumes I and IV, for a description of the package. See “Package Manager” in Appendix F for the A/UX C interface to the Binary-Decimal Conversion Package.

Color Manager

The A/UX Color Manager is identical to the Macintosh OS Color Manager. See *Inside Macintosh*, Volume V, for a description of the Color Manager. See “32-Bit QuickDraw With Color QuickDraw” in Appendix F for the A/UX C interface to the Color Manager.

Color Picker Package

The A/UX Color Picker Package is identical to the Macintosh OS Color Picker Package. See *Inside Macintosh*, Volume VI, for a description of the Color Picker Package. See “Color Picker” in Appendix F for the A/UX C interface.

Control Manager

The A/UX Toolbox Control Manager is almost identical to the Macintosh OS Control Manager. The difference is that in A/UX a control’s variant code is not stored in the `ctrlDefProc` field of the control record. To retrieve the variant code, use the Control Manager call `GetCVariant`, described in *Inside Macintosh*, Volume V.

See *Inside Macintosh*, Volumes I, IV, and V, for a description of the Control Manager. See “Control Manager” in Appendix F for the A/UX C interface.

Data Access Manager

The A/UX Data Access Manager is identical to the Macintosh OS Data Access Manager. See *Inside Macintosh*, Volume VI, for a description of the Data Access Manager.

Deferred Task Manager

The Deferred Task Manager is identical to the Macintosh OS Deferred Task Manager. See *Inside Macintosh*, Volume VI, for a description of the Deferred Task Manager. See “Deferred Task Manager” in Appendix F for the A/UX C interface.

Desk (Accessory) Manager

The Desk Manager is identical to the Macintosh OS Desk Manager. Usually, though not always, when a desk accessory is opened, the Process Manager launches the desk accessory in its own partition, and otherwise treats it as a small application.

See *Inside Macintosh*, Volume VI, for a description of the Process Manager. See “Desk Manager” in Appendix F for the A/UX C interface.

Desktop Manager

The A/UX Desktop Manager is identical to the Macintosh OS Desktop Manager. See *Inside Macintosh*, Volume VI, for a description of the Desktop Manager.

Device Manager

The A/UX Device Manager is identical to the Macintosh OS Device Manager, but A/UX places the same restrictions on device drivers as on applications. Device drivers outside the kernel cannot manipulate hardware directly. Therefore, desk accessories are supported, but most custom NuBus™ card drivers are not. A/UX currently supports custom video drivers, the AppleTalk drivers, and AppleTalk-based printer drivers.

If your application needs to control hardware directly, you must use an A/UX device driver. (For information on writing an A/UX device driver, see *Building A/UX Device Drivers*.) You can then write a Macintosh device driver that uses A/UX system calls, such as `open(2)` and `ioctl(2)`, to access the A/UX device driver that you have installed in the kernel. A program that uses an A/UX device driver is not portable to the Macintosh OS. See Chapter 3, “A/UX Toolbox Utilities and Extensions,” for a strategy for including A/UX system calls in applications that are intended to run under both the Macintosh OS and the A/UX operating system.

See *Inside Macintosh*, Volumes II, IV, and V, for a description of the Device Manager. See “Device Manager” in Appendix F for the A/UX C interface. For an example of driver calls, see “Disk Driver” in Appendix F.

Dialog Manager

The A/UX Dialog Manager is identical to the Macintosh OS Dialog Manager.

Because the System Error Handler cannot resume processing after an error, it ignores the `resumeProc` variable passed to it by the `InitDialogs` routine.

When using the Dialog Manager under A/UX, remember to make provisions for these common compatibility problems:

- *Newline character* Individual lines in a multiple-line message passed to the Dialog Manager must be separated by return characters (`\r`) when using the `cc` (or `c89`) C compiler, though not with the MPW C compiler.
- `ProcPtr` *parameters* Any procedure passed as a parameter to a Dialog Manager routine must use Pascal calling conventions. See Appendix C, “Implementation Notes,” for a description of the Pascal conventions.

See *Inside Macintosh*, Volumes I, IV, V, and VI, for a description of the Dialog Manager and the facilities it offers. See “Dialog Manager” in Appendix F for the A/UX C interface.

Disk Driver

The Disk Driver is supported. See *Inside Macintosh*, Volumes II, IV, and V, for a description of the Disk Driver. See “Disk Driver” in Appendix F for the A/UX C interface.

Disk Initialization Package

The Disk Initialization Package is supported. See *Inside Macintosh*, Volumes II and IV, for a description of the Disk Initialization Package. The Disk Initialization Package is accessed through the Package Manager. See “Package Manager” in Appendix F for the A/UX C interface.

Edition Manager

The A/UX Edition Manager is identical to the Macintosh OS Edition Manager. See *Inside Macintosh*, Volume VI, for a description of the Edition Manager.

Event Manager, Operating System

The A/UX Toolbox supports most of the standard Macintosh OS Event Manager routines. Because the A/UX kernel maintains the event queue, the A/UX Toolbox version of the manager performs differently than the Macintosh OS version. See *Inside Macintosh*, Volumes II and IV, for a description of the Macintosh OS Event Manager. See “Event Manager, Operating System” in Appendix F for the A/UX C interface. Also, see “AUXDispatch Trap,” in Chapter 3, for related information.

The global variable `EventQueue` always contains the header of an empty queue. Therefore, an application cannot look directly at the actual queue and must depend on Event Manager routines for manipulating the queue. You can use the `AUXDispatch` call `AUX_FIND_EVENT` to search the event queue for an event.

In the Macintosh OS, all events are put into the queue through the `PostEvent` routine. In A/UX, mouse and keyboard events are processed through the kernel, and the system never calls `PostEvent`; the `PPostEvent` trap is not supported. An application cannot depend on a patch to `PostEvent` to alert it to mouse and keyboard events. The `AUXDispatch` call `AUX_POST_MODIFIED` is the equivalent A/UX call.

Event Manager, Toolbox

The A/UX Toolbox supports all of the routines in the Macintosh OS Toolbox Event Manager, but some of the functions perform differently under the A/UX operating system than under the Macintosh OS. See *Inside Macintosh*, Volumes VI, I, IV, and V, for a description of the Toolbox Event Manager. See “Event Manager, Toolbox” in Appendix F for the A/UX C interface.

The A/UX Toolbox supports the `WaitNextEvent` call, which allows the system to run more efficiently in the multitasking environment of the A/UX Finder. Use the `WaitNextEvent` call instead of the `GetNextEvent` call. For more information on the `WaitNextEvent` call, see *Inside Macintosh*, Volume VI, and “Using the `ui_setselect` Call” in Chapter 2.

When using the Toolbox Event Manager, make provisions for these differences between the A/UX operating system and the Macintosh OS:

- Because the global variables `KeyThresh` and `KeyRepThresh` are ignored, an application cannot change key-repeat characteristics.
- Journaling is not supported, because events are handled differently by each operating system. However, journals recorded under the Macintosh Operating System (for example, macros made with MacroMaker) can be played under the A/UX operating system.

File Manager

The A/UX Toolbox File Manager supports access to Macintosh OS volumes and UNIX file systems. UNIX file systems are supported as external file systems in much the same way that AppleShare file systems are supported. UNIX external file systems support almost all File Manager calls, including file IDs, file specification (FSSpec) records, and foreign privilege buffers. In addition, A/UX supports the `_GetForeignPrivs` and `_SetForeignPrivs` traps, returning the `ioForeignPrivBuffer` to store information on UNIX permissions, so this information can be backed up and restored by Macintosh backup utilities.

Files can be accessed across the boundary between the Macintosh file environment and the A/UX file environment. Chapter 6, “File Systems and File Formats,” provides information on how file structure and contents change when files move across the boundary. Such changes include file permissions, file formats, and line-termination codes.

The underlying support for the File Manager is provided by the Berkeley UNIX file system (UFS), an implementation of the file system used by the BSD (Berkeley Software Distribution) 4.2 operating system. In addition to being faster than the System V file system (which is still available), the new file system allows filenames of up to 255 characters. The maximum length of an HFS name is 32 characters; longer names brought into the Macintosh OS environment are truncated.

For information on the Macintosh OS File Manager, see *Inside Macintosh*, Volumes IV, V, and VI. See “File Manager” in Appendix F for the A/UX C interface.

Floating-Point Arithmetic and Transcendental Functions Packages

C programmers rarely, if ever, explicitly call the routines in the Floating-Point Arithmetic and Transcendental Functions Packages. These packages support the Standard Apple Numeric Environment (SANE).

Most Macintosh C compilers use SANE. Mathematical functions in the standard C library are routed through the SANE packages. When a Macintosh binary file that uses SANE is ported to A/UX, the SANE routines are already in place in the code.

The A/UX C compiler `cc` uses the standard A/UX floating-point routines. The SANE packages are not available to programs compiled under A/UX.

See *Inside Macintosh*, Volumes II and V, for a description of the Floating-Point Arithmetic and Transcendental Functions Packages. See “Package Manager” in Appendix F for the A/UX C interface to these packages.

Font Manager

The A/UX Toolbox Font Manager in Release 3.0 supports TrueType fonts. See *Inside Macintosh*, Volumes I, IV, V, and VI, for a description of the Font Manager. See “Font Manager” in Appendix F for the A/UX C interface.

Gestalt Manager

The A/UX Toolbox Gestalt Manager provides full support for the Macintosh OS Gestalt facility. The following environmental selector is available; it can be used to determine if your application is running under A/UX and, if so, which version:

```
gestaltAUXVersion = 'a/ux'
```

Calling Gestalt with this selector returns the version number, with implied decimal points. If you are not running under A/UX, Gestalt returns a result code of `gestaltUnknownErr`, value `-5550`.

To determine if Gestalt itself is available, use the `TrapAvailable` function. The Gestalt trap address is `$A1AD`. For example, use the following routine:

```
/* determine if Gestalt Manager is available by calling
   TrapAvailable */
unsigned char GestaltAvailable( void )
{
    return( TrapAvailable( _Gestalt ) );
}
```

Here is a typical implementation of `TrapAvailable`:

```
Boolean TrapAvailable(tNumber, tType)
    short      tNumber;
    TrapType   tType;
{
    if ( ( tType == ToolTrap ) &&
        ( gMac.machineType > envMachUnknown ) &&
        ( gMac.machineType < envMacII ) ) { /*it's a 512KE, */
        tNumber = tNumber & 0x03FF;          /* Plus, or SE */
        if ( tNumber > 0x01FF )             /* which means the */
            tNumber = _Unimplemented;      /* tool traps only */
        }                                   /* go to 0x01FF */
    return NGetTrapAddress(tNumber, tType) !=
        GetTrapAddress(_Unimplemented);
}
```

◆ **Note** Checking that Gestalt is available is required only if you want your application to be backward-compatible with Macintosh systems running a Macintosh OS version prior to System 7. All systems running A/UX Release 2.0 or later releases implement the Gestalt facility. ◆

See *Inside Macintosh*, Volume VI, for a description of the Gestalt Manager.

Graphics Devices Manager

The A/UX Graphics Devices Manager is identical to the Macintosh OS Graphics Devices Manager. See *Inside Macintosh*, Volume VI, for a description of the Graphics Devices Manager.

Help Manager

The A/UX Help Manager is identical to the Macintosh OS Help Manager. See *Inside Macintosh*, Volume VI, for a description of the Help Manager.

International Utilities Package

The A/UX Toolbox fully supports the Macintosh OS International Utilities Package. See *Inside Macintosh*, Volumes I, V, and VI, for a description of the International Utilities Package. See “Package Manager” in Appendix F for the A/UX C interface to the International Utilities Package.

List Manager Package

The A/UX Toolbox fully supports the Macintosh OS List Manager Package. See *Inside Macintosh*, Volume IV, for a description of the List Manager Package. See “List Manager Package” in Appendix F for the A/UX C interface.

Memory Manager

The A/UX Toolbox fully supports the Macintosh OS Memory Manager, including virtual memory. While the implementation of virtual memory differs between the Macintosh OS and A/UX operating system, this is transparent to both the user and the programmer.

The Memory Manager is 32-bit clean and expects to serve applications that are 32-bit clean. A/UX Release 3.0 offers a special 24-bit environment in which older applications can be run. The special environment takes care of memory addressing. From within the 24-bit environment, there is limited access to the standard 32-bit environment.

See “The Memory Manager” in *Inside Macintosh*, Volume II, which is intended to be read in conjunction with related chapters in Volumes IV and VI. See “Memory Manager” in Appendix F for the A/UX C interface.

Menu Manager

The A/UX Toolbox Menu Manager is identical to the Macintosh OS Menu Manager.

If your application uses custom menu definition functions, you must provide assembly-language routines to transform the parameters into Pascal format for compatibility with the ROM. (For information on transforming parameters into Pascal format, see Chapter 4 and Appendix C.)

See *Inside Macintosh*, Volumes I, IV, and V, for a description of the Menu Manager. See “Menu Manager” in Appendix F for the A/UX C interface.

Notification Manager

The A/UX Toolbox Notification Manager is identical to the Macintosh OS Notification Manager. See *Inside Macintosh*, Volume VI, for a description of the Notification Manager. See “Notification Manager” in Appendix F for the A/UX C interface.

Package Manager

The A/UX Toolbox supports both Macintosh OS Package Manager routines. The A/UX Package Manager supports interfaces to the Standard File, Floating-Point Arithmetic, Transcendental Functions, International Utilities, Disk Initialization, and Binary-Decimal Conversion Packages. The List Manager Package, available directly as a separate library, is also available through the Package Manager for historical reasons.

See *Inside Macintosh*, Volumes I and IV, for a description of the Package Manager. See “Package Manager” and “List Manager Package” in Appendix F for the A/UX C interface.

Palette Manager

The A/UX Toolbox Palette Manager is identical to the Macintosh OS Palette Manager. See *Inside Macintosh*, Volume VI, for a description of the Palette Manager. See “Palette Manager” in Appendix F for the A/UX C interface.

Picture Utilities Package

The A/UX Picture Utilities Package is identical to the Macintosh OS Picture Utilities Package. See *Inside Macintosh*, Volume VI, for a description of the Picture Utilities Package.

Power Manager

The Power Manager is not implemented in A/UX Release 3.0. Macintosh portable computers are the only systems that use this manager, and as of this writing A/UX is not supported on portable computers.

PPC Toolbox

The A/UX Program-to-Program Communications (PPC) Toolbox is identical to the Macintosh OS PPC Toolbox. See *Inside Macintosh*, Volume VI, for a description of the PPC Toolbox.

Printing Manager

The A/UX Toolbox Printing Manager is identical to the Macintosh OS Printing Manager. A/UX Release 3.0 supports AppleTalk-based printer drivers (for LocalTalk or Ethernet) and serial printer drivers.

See *Inside Macintosh*, Volumes II and V, for a description of the Printing Manager. See “Printing Manager” and “Print Traps” in Appendix F for the A/UX C interface.

Process Manager

The A/UX Process Manager is identical to the Macintosh OS Process Manager. See *Inside Macintosh*, Volume VI, for a description of the Process Manager. See “Process Manager” in Appendix F for the A/UX C interface.

Resource Manager

The A/UX Toolbox Resource Manager is almost identical to the standard Macintosh OS Resource Manager. The differences between the two result primarily from differences between file systems. All Resource Manager calls documented in *Inside Macintosh* are implemented in the A/UX Toolbox.

See *Inside Macintosh*, Volumes I, IV, V, and VI, for a description of the Resource Manager. See “Resource Manager” in Appendix F for the A/UX C interface. See *ResEdit Reference* (for version 2.1) for information on editing resources. Additional information on the resource compiler, `rez`, and resource decompiler, `derез`, is given in Chapter 3 and in Appendix E.

When using the Resource Manager, you must make provisions for these differences between the environments:

- *Resource files* A/UX files in AppleDouble format (described in Chapter 6) have their resources and data stored in separate files. Be careful to keep both files together when copying, renaming, or otherwise manipulating AppleDouble files with UNIX commands such as `mv`. See Chapter 2 and Chapter 6 for descriptions of Macintosh file formats in A/UX.
- *Write permission* Your application might not have write permission in the directory containing the System Folder (typically `/mac/lib/System Files`) or the application.
- *Case-sensitive filenames* Unlike the Macintosh OS, A/UX differentiates between uppercase and lowercase characters in filenames. Be careful with the filenames in `OpenResFile` and `CreateResFile`.
- *Search paths* The System 7 Macintosh OS File Manager uses FSSpec records to unambiguously establish the location of files. When creating resource files, you are encouraged to use the `FSpCreateResFile` procedure whenever possible. The previous version of the Macintosh OS File Manager checks a number of search paths if it cannot find a file in the specified directory. Because of this feature, the `CreateResFile` routine can introduce some subtle inconsistencies in search paths when creating resource files. Searching of alternative paths is not supported under A/UX. Programs that are intended to run in both environments should follow the strategies recommended in *Macintosh Technical Note #101*, even though those strategies are not needed in A/UX. Technical notes are available through APDA.

- ◆ **Note** Checking search paths is an issue only if you want your application to be backward-compatible with Macintosh systems running a Macintosh OS version prior to System 7. ◆

In the absence of the default search paths, an application must explicitly set the default directory when opening a resource file in the “blessed” folder, usually the System Folder. An application must first determine the working-directory reference number of the desired directory, and then set the default directory with the File Manager function `SetVol`. See *Macintosh Technical Notes #67* and *#77* and *Inside Macintosh*, Volumes IV and VI.

Scrap Manager

The A/UX Toolbox Scrap Manager is almost identical to the Macintosh OS Scrap Manager.

The only difference between the two Scrap Managers is the way in which they store material cut to the Clipboard. The A/UX Toolbox Scrap Manager maintains a `.clipboard` file in your home directory when you execute an A/UX Toolbox application. The contents of the scrap are written into this file when an application exits, allowing you to cut and paste between applications.

See *Inside Macintosh*, Volumes I and IV, for a description of the Scrap Manager. See “Scrap Manager” in Appendix F for the A/UX C interface.

Script Manager

The A/UX Toolbox Script Manager is identical to the Macintosh OS Script Manager. See *Inside Macintosh*, Volumes V and VI, for a description of the Script Manager. See “Script Manager” in Appendix F for the A/UX C interface.

SCSI Manager

The A/UX Toolbox does not currently support the Macintosh OS SCSI Manager functions. A call to a SCSI Manager routine returns an `unimplemented trap message`.

For an application that is intended to run only under A/UX, you can write an A/UX device driver. For more information, see *Building A/UX Device Drivers* in the A/UX Device Drivers Kit, available from APDA. A program that uses an A/UX device driver is not directly portable to the Macintosh OS. However, by using the Gestalt facility, you can create binary-compatible code that runs in both environments.

Segment Loader

Applications in the standard Macintosh development environment are written in segments, which are loaded individually as needed so that memory is used efficiently. Segments are not used in the A/UX environment, but the Segment Loader has been implemented to support Macintosh binary applications launched under A/UX.

See *Inside Macintosh*, Volumes II, IV, and VI, for a description of the Segment Loader. See “Segment Loader” in Appendix F for the A/UX C interface.

An application may or may not contain Segment Loader calls, depending on its format and intended running environment:

- Standard Macintosh binary files launched under A/UX are loaded by the Segment Loader in the normal fashion.
- Applications ported to A/UX from Macintosh sources or written to run in both environments can include calls to Segment Loader routines.
- Applications written to run exclusively under A/UX need not use Segment Loader calls.

Finder information

The format of the file information passed to an application by the A/UX Finder follows Macintosh OS conventions.

When an application is started under A/UX, the application’s Finder information is in one of these states:

- An application developed for A/UX shows no documents selected.
- An application developed for A/UX shows one or several documents selected.
- A Macintosh application has a Finder document list based on the parameters in the `launch(1)` command line.

Segment Loader routines

This section lists the Segment Loader routines that are different in the A/UX Toolbox than in the User Interface Toolbox. The Segment Loader routines not listed here are implemented as described in *Inside Macintosh*.

<code>UnloadSeg</code>	Performs normally for Macintosh binary applications that are launched; stubbed out for native A/UX applications.
<code>Chain</code>	Not used in multitasking environments; <code>Launch</code> is used instead.
<code>LoadSeg</code>	Performs normally for Macintosh binary applications that are launched; stubbed out for native A/UX applications.

The jump table

The jump table works as described in *Inside Macintosh* for Macintosh applications that are launched under A/UX; it is not implemented for native A/UX and UNIX applications.

Alternate buffer support

`CurPageOption` is always set to 0, meaning that there is no alternate screen or sound buffer.

Serial Driver

The A/UX Toolbox partially supports the Macintosh OS Serial Driver. The five exceptions to full support are as follows:

- `ASYNC` calls are not supported.
- Three baud rates are not supported: 3600, 7200, and 57600. These rates are mapped to 2400, 4800, and 19200, respectively. This constraint affects control calls 8 (`SerReset`) and 13 (`baudRate`). The baud rate 38400 is supported.
- Control call 9 (`SerSetBuf`) has no effect. When called, it just returns.
- Event-message posting is not supported. This limitation affects control calls 10 and 14 (`SerHShake`). If the `evts` field in the `SerShk` record is nonzero, an error is returned.
- Status call 8 (`SerStatus`) will always return 0 in the `rdPend` and `wrPend` fields.

Eight ioctl calls have been added to the A/UX Serial Driver to support the Serial Driver running under A/UX:

```
ioctl(fd, TCRESET, 0);
```

This ioctl causes a reset of the serial line denoted by the file descriptor, *fd*.

```
ioctl(fd, TCGETSTAT, &serstat);
```

This ioctl returns status information for the serial line denoted by *fd* into the structure *serstat*. The variables *ser_frame*, *ser_ovrun*, and *ser_parity* represent the error counts that have been tallied since the last call to *TCGETSTAT*. (These fields are set to 0 when the call completes.) The variable *ser_cts* indicates the current status of the CTS (Clear to Send) signal; *TRUE* indicates CTS ON (high). The variable *ser_inflow* is *TRUE* if input is currently blocked because of flow control. The variable *ser_outflow* is *TRUE* if output is blocked because of flow control. Here is the data structure:

```
struct sererr {
    unsigned long ser_frame;    /* framing errors */
    unsigned long ser_ovrun;    /* overrun errors */
    unsigned long ser_parity;  /* parity errors */
    unsigned long ser_cts;     /* cts signal */
    unsigned long ser_inflow;   /* input flow control */
    unsigned long ser_outflow; /* output flow control */
};
```

```
ioctl(fd, TCSETDTR, 0);
```

This ioctl turns on the DTR (Data Terminal Ready) line (drives it high) for the serial line denoted by *fd*.

```
ioctl(fd, TCCLRDRTR, 0);
```

This ioctl turns off the DTR line (drives it low) for the serial line denoted by *fd*.

```
ioctl(fd, TCSBRKM, 0);
```

This ioctl sets break mode (starts a line-break signal) for the serial line denoted by *fd*.

```
ioctl(fd, TCCBRKM, 0);
```

This ioctl clears break mode (terminates a line-break signal) for the serial line denoted by *fd*.

```
ioctl (fd, TCSETSTP, &chr);
```

This ioctl sets the stop character for flow control for the serial line denoted by *fd*. *chr* points to a byte containing the new stop character.

```
ioctl (fd, TCSETSTA, &chr);
```

This ioctl sets the start character for flow control for the serial line denoted by *fd*. *chr* points to a byte containing the new start character.

The developer of a driver for a serial card must support these eight calls in the driver's ioctl routine if the A/UX Serial Driver is to work properly. See *Building A/UX Device Drivers* for additional details. See “Serial Driver” in Appendix F for the A/UX C interface.

Shutdown Manager

The A/UX Toolbox supports all Macintosh OS Shutdown Manager routines. The shutdown queue is traversed and executed at logout.

See *Inside Macintosh*, Volume V, for a description of the Shutdown Manager. See “Shutdown Manager” in Appendix F for the A/UX C interface.

Slot Manager

The A/UX Toolbox Slot Manager partially supports the Macintosh OS Slot Manager. Details follow, keyed to the summary in “Slot Manager” in *Inside Macintosh*, Volume V.

- All principal routines are supported.
- Specialized routines are supported except for one routine:
SDeleteSRTRec
- Advanced routines are supported except for the following routines:

InitSDeclMgr

SPrimaryInit

SExec

InitsRsrcTable

InitPRAMRecs

SGetDriver

SFindsInfoRecPtr

- Assembly-language routine selectors are supported except for the following routines:

sDisposePtr

InitSDeclMgr

sPrimaryInit

sExec

InitPRAMRecs

InitsRsrcTable

sGetDriver

sFindsInfoRecPtr

sDeleteSRTRec

See *Inside Macintosh*, Volumes V and VI, for a description of the Slot Manager. See “Slot Manager” in Appendix F for the A/UX C interface.

Sound Manager

The A/UX Toolbox partially supports the Macintosh OS Sound Manager. Operating with the virtual memory environment of A/UX, the Sound Manager can process files of any desired length. A raw sound driver is also available for use outside the A/UX Toolbox—for example, in shell scripts.

The exceptions to full support for the Sound Manager are as follows:

- The A/UX Toolbox supports only one sampled channel, instead of two.
- Some commands are currently available only on computers equipped with the Apple Sound Chip (ASC). The following A/UX-capable systems use the ASC: the Macintosh SE/30 and the Macintosh II family of computers, including the Macintosh II, IIx, IIcx, IICI, IISI, and IIfx.

- For the `_SndAddModifier` trap, no `addMod` command is supported for synthesizer modules, because synthesizer modules must go in the kernel. Current synthesizers are supported for `noteSynth`, `waveSynth`, and `sampledSynth`. Modules for new synthesizers would need to be ported to the kernel.

The ability to call user routines at interrupt level has been emulated by means of a circular buffering scheme, avoiding the anticipated problems with security, page faults, and context switching.

If the system has too heavy a load of other activities, sound production is affected. The process slows, and the sound begins to have gaps or sputtering. This situation can occur under either the Macintosh Operating System or the A/UX operating system.

The folder `/mac/src/sndDemo` contains demonstration and sample programs that use the Sound Manager.

See *Inside Macintosh*, Volume VI, for a description of the Sound Manager. See “Sound Manager” in Appendix F for the A/UX C interface.

Support details

Here are the details on differences in trap support:

- Sound channel commands are supported with one exception:

<code>SndAddModifier</code>	not supported for synthesizer modules
-----------------------------	---------------------------------------
- Sound recording commands are supported with two exceptions:

<code>SPBSignInDevice</code>	not supported
<code>SPBSignOutDevice</code>	not supported
- Commands sent normally only by the Sound Manager are partially supported:

<code>reinitCmd</code>	supported for note, wave, and sampled synthesizers
<code>timbreCmd</code>	supported for systems that have the ASC
<code>waveTableCmd</code>	supported for systems that have the ASC

- Initialization options for `SndNewChannel`, sampled synthesizer only, are partially supported:

<code>initChanLeft</code>	ignored; defaults to mono
<code>initChanRight</code>	ignored
<code>initStereo</code>	not supported; defaults to mono
- Synthesizer resource IDs with `SndNewChannel` are partially supported:

<code>squareWaveSynth</code>	supported for systems that have the ASC
<code>waveTableSynth</code>	supported for systems that have the ASC
- Initialization options for `SndNewChannel`, wave-table synthesizer only, are supported:

<code>initChan0</code>	supported for wave-table synthesizers only
<code>initChan1</code>	supported for wave-table synthesizers only
<code>initChan2</code>	supported for wave-table synthesizers only
<code>initChan3</code>	supported for wave-table synthesizers only

The Raw Sound Driver

You can use the Raw Sound Driver under A/UX (for use in shell scripts and so forth) without calling upon the Sound Manager. The Raw Sound Driver is available as `/dev/snd/raw`

To use the Raw Sound Driver, prepare a file of sampled sound resources and send it (by means of `cat`, for instance) to the device. For example, to send a file called `sndFile`, use the following command line:

```
cat sndFile > /dev/snd/raw
```

Sending a character to `/dev/snd/reset` resets the synthesizer driver in the kernel, resetting both the Sound Manager and the Raw Sound Driver. Here is a reset example that uses the `echo` command:

```
'x' > /dev/snd/reset
```


The sampling rate of the Raw Sound Driver is 22 KHz by default. To change the rate, use an `ioctl` to send a structure to the Raw Sound Driver. (See the example later in this section.) The structure, named `rawSndCtl`, contains a field called `sampleRate`, which contains a 4-byte value interpreted as a fixed-point binary number with an implied binary point between the upper and lower words. The value is a multiplier used to reduce the 22KHz maximum rate. The value of `$00010000`, meaning \$1.0000, preserves the default rate. Here is an example that sets the rate to 7KHz. Calculate the multiplier:

$$7K/22K = .318 \text{ (decimal value)}$$

Multiply by `$00001.0000` to adjust for the binary point and convert to hexadecimal (`$00001.0000 = 65536`):

$$65536 * .318 = 20852 = \$0000.5222$$

In practice, the value need not be calculated so precisely.

Here is an example C routine that places `$00005222` in the `rawSndCtl` structure and sets the driver with an `ioctl(2)` call:

```
#include <mac/sm.h>
#include <sys/types.h>
#include <sys/ssioctl.h>
#include <sys/sys/sm_aux.h>
#include <sys/file.h>
#define SAMPLERATE 0x5222 /* (7k/22k) * 65536 */
main()
{
    int snd_fd;
    struct rawSndCtl rawSndInfo;
    if ((snd_fd = open("/dev/snd/raw",O_WRONLY)) < 0) {
        printf("open failed\n");
        exit(1);
    }
    rawSndInfo.sampleRate = SAMPLERATE;
    rawSndInfo.flags = 0;
    if (ioctl(snd_fd, SND_RAW_CTL, &rawSndInfo) < 0)
        printf("ioctl failed\n");
    close(snd_fd);
}
```

Standard File Package

The A/UX Toolbox Standard File Package is identical to the Macintosh OS Standard File Package. See *Inside Macintosh*, Volumes I, IV, and VI, for a description of the package. See “Package Manager” in Appendix F for the A/UX C interface to the Standard File Package.

System Error Handler

The A/UX Toolbox supports the single Macintosh OS System Error Handler routine, `SysError`.

When the system issues the `SysError` call, and MacsBug is *not* installed, the System Error Handler writes a brief error message to the program’s `stderr` file and terminates the program, with an exit status of 1. The error message contains the error number and the location from which `SysError` was called.

See *Inside Macintosh*, Volumes II, IV, and V, for a description of the System Error Handler. See “System Error Handler” in Appendix F for the A/UX C interface.

TextEdit

The A/UX Toolbox TextEdit facility is identical to the Macintosh OS TextEdit facility. You can set up low-level routines to perform tasks such as customized word-breaking, but you must provide assembly-language routines to handle the interface between TextEdit and your custom routines. The TextEdit interface is based on registers. This interface follows neither Pascal nor C conventions, and it varies from call to call.

See *Inside Macintosh*, Volumes I, IV, V, and VI, for a description of TextEdit. See “TextEdit” in Appendix F for the A/UX C interface.

Time Manager

The A/UX Toolbox provides a less accurate implementation of the standard Macintosh Time Manager. The A/UX Toolbox Time Manager uses the A/UX `setitimer(2)` system call. Because of the A/UX kernel's processor-allocation strategies, response from the Time Manager may be delayed an arbitrary amount of time, depending on other system activity. Even when it is operating without interference, the A/UX Time Manager provides accuracy to only one-sixtieth of a second.

When you use the Time Manager in an application, you must observe these limitations:

- You must not make calls to the A/UX C library routines `alarm(2)`, `setitimer(2)`, and `sleep(2)`.
- You must not use `signal(2)` to change the status of the `SIGALRM` signal.

See *A/UX Programmer's Reference* for more information on `setitimer(2)`. See *Inside Macintosh*, Volume VI, for a description of the Time Manager. See "Time Manager" in Appendix F for the A/UX C interface.

Utilities, Operating System

The A/UX Toolbox contains some of the Operating System utilities:

- Routines that manipulate pointers and handles and compare strings are fully functional.
- Routines that read the date and time behave differently. (See the next section, "Date and Time Operations.")
- Routines that manipulate parameter RAM are fully functional.
- The queuing and trap vector routines are fully functional.
- The miscellaneous utility `Delay` is fully functional.
- The former Operating System utility `SysBeep` is now a Sound Manager routine.
- The `StripAddress` routine always returns the pointer unchanged in the 32-bit environment; the routine functions in the 24-bit environment.

See *Inside Macintosh*, Volumes II, IV, and V, for a description of the Operating System Utilities. See "Utilities, Operating System" in Appendix F for the A/UX C interface.

Date and time operations

To find out the correct date and time, use the `ReadDateTime` utility. The global variable `Time` is set when a program starts running, and it is not updated. Therefore, `GetDateTime` always returns the time at which the program started running.

Because you must be logged in to A/UX as `root` to set the system clock through either the `date(1)` command or the `stime(2)` command, you cannot change the clock setting by using A/UX Toolbox calls. The `SetDateTime` utility returns the error `clkWrEr`.

Miscellaneous utilities

Because the `Restart` routine results in a privileged 680x0 instruction not available to programs running at the user level, it is not supported in A/UX. Instead, use the Shutdown Manager routines. (See “Shutdown Manager,” earlier in this chapter.)

`Delay` is fully functional.

`SetUpA5` and `RestoreA5` are dummy routines that return with no action.

The `StripAddress` routine always returns the pointer unchanged in the 32-bit environment; the routine functions in the 24-bit environment.

Utilities, Toolbox

All Macintosh OS Toolbox Utilities routines are implemented in the A/UX Toolbox. See *Inside Macintosh*, Volumes I and IV, for a description of the Toolbox Utilities. See “Utilities, Toolbox” in Appendix F for the A/UX C interface.

Vertical Retrace Manager

All of the Vertical Retrace Manager routines described in *Inside Macintosh*, Volumes II and V, are implemented in A/UX, with the exception of `DOVBLTask`. You are encouraged to use the Time Manager instead of the Vertical Retrace Manager in your applications.

The A/UX Vertical Retrace Manager routines are implemented by means of the A/UX `setitimer(2)` system call. Because of changes in the Macintosh II ROM that allow for multiple video options, tasks scheduled by the Vertical Retrace Manager are not necessarily run during the vertical retrace. Like Time Manager routines, Vertical Retrace Manager routines under A/UX may be delayed an arbitrary length of time, depending on other system activity.

When you use the Vertical Retrace Manager, you must observe these limitations:

- You cannot make calls to the A/UX C library routines `alarm(2)`, `setitimer(2)`, and `sleep(2)`.
- You cannot use `signal(2)` to change the status of the `SIGALRM` signal.
- You cannot use the `DoVBLTask` function.

See *A/UX Programmer's Reference* for more information on the A/UX calls used by the Vertical Retrace Manager. See “Vertical Retrace Manager” in Appendix F for the A/UX C interface.

Window Manager

The A/UX Toolbox Window Manager is almost identical to the Macintosh OS Window Manager. The difference is that in A/UX the window's variant code is not stored in the `windowDefProc` field of the window record. To get the variant code, use the Window Manager call `GetWVariant`, described in *Inside Macintosh*, Volume V.

See *Inside Macintosh*, Volumes I, IV, V, and VI, for a description of the Window Manager. See “Window Manager” in Appendix F for the A/UX C interface.

Calls patched under A/UX

Table 5-2 lists alphabetically the calls that are patched under A/UX. Implementation notes or cross-references are provided where available.

Table 5-2 ROM calls patched under the A/UX Toolbox

ROM call	Notes
AttachVBL	
Button	
CompactMem	
Debugger	
DebugStr	
Delay	
DTInstall	
EmptyHandle	
EnQueue	
FlushEvents	
FreeMem	
GetHandleSize	
GetKeys	
GetOSEvent	
GetPtrSize	
GetZone	
HGetState	
HLock	
HNoPurge	
HPurge	
HSetRBit	
HSetState	
HUnlock	
InitApplZone	
InitUtil	

(continued) ➡

Table 5-2 ROM calls patched under the A/UX Toolbox (*continued*)

ROM call	Notes
InitZone	
InsTime	
InsXTime	
LoadSeg	Performs normally for Macintosh binary applications that are launched; stubbed out for native A/UX applications.
MaxApplZone	
MaxBlock	
MaxMem	
MoreMasters	
Pack12	
PostEvent	Applications cannot depend on patches to PostEvent to alert it to mouse and keyboard events. The AUXDispatch call AUX_POST_MODIFIED is the equivalent A/UX call to PPostEvent.
PowerOff	
PrimeTime	
PtrZone	
PurgeMem	
PurgeSpace	
ReadDateTime	
ReadXPRam	
RmvTime	
ScriptUtil	
ScrnBitMap	
SCSIDispatch	No calls are supported except for the scsiStat selector of _SCSIDispatch.
SerHShake	Event-message posting is not supported. If the evts field in the SerShk record is nonzero, an error is returned.
SerReset	3600, 7200, and 57600 baud rates are not supported.
SerSetBuf	Has no effect. When called, it just returns.
SerStatus	Always returns 0 in the rdPend and wrPend fields.

(continued) ➡

Table 5-2 ROM calls patched under the A/UX Toolbox (*continued*)


ROM call	Notes
SetAplLimit	
SetDateTime	
SetHandleSize	
SetPtrSize	
SetZone	
ShutDown	
SlotManager	
SlotVInstall	
SlotVRemove	
SndAddModifier	Not supported for synthesizer modules.
SndControl	
SndDisposeChannel	
SndDoCommand	
SndDoImmediate	
SndNewChannel	
SndPlay	
StripAddress	Always returns the pointer unchanged in the 32-bit environment; the routine functions in the 24-bit environment.
SwapMMUMode	
SysBeep	
UnloadSeg	Performs normally for Macintosh binary applications that are launched; stubbed out for native A/UX applications.
VInstall	
VRemove	
WriteParam	
WriteXPram	

Calls not supported under A/UX

Table 5-3 lists alphabetically the calls that are not supported under A/UX.

Table 5-3 ROM calls not supported under the A/UX Toolbox

ADBop	IdleState	SetOSDefault
ADBReInit	IdleUpdate	SIntInstall
AddReference	Pack10	SIntRemove
Chain	PMgrOP	Sleep
CountADBs	PPostEvent	SlpQInstall
DoVBLTask	RmveReference	SlpQRemove
FInitQueue	SCSIDispatch	SPBSignInDevice
GetADBInfo	SerialPo	SPBSignOutDevice
GetIndADB	SetADBInf	



6 File Systems and File Formats

File systems / 6-2

Storing files in the Macintosh OS and in the A/UX operating system / 6-8

AppleSingle and AppleDouble format internals / 6-16

Filename conventions / 6-20

This chapter describes how the file systems in the A/UX operating system and the file systems in the Macintosh environment differ. Users and applications can access files either from the A/UX environment or from the A/UX Toolbox, as convenient, and can transfer files between the two environments without any special requirements. This chapter also describes the formats used for storing Macintosh files in the A/UX environment, and the automatic conversion that occurs when files are transferred between the two environments.

File systems

The design of file systems in the A/UX operating system differs from that of file systems in the Macintosh OS, but file-system functions are mapped between the two environments so that files can be transferred between them or accessed from either environment by A/UX Toolbox programs.

The term *file system*, as used in this chapter, refers to general design and implementation. In the context of the UNIX operating system, the term *file system* is used for a subset of the general file-handling design. When the UNIX definition is meant, that is specifically stated.

These general file systems offer high-level functionality. For A/UX, each file system mounted under the root hierarchy provides high-level UNIX operations such as open, create, and delete, regardless of the underlying physical implementation (System V file system [SVFS], Berkeley UNIX file system [UFS], Network File System [NFS], and so on). The Macintosh OS file system provides equivalent functionality for files in volumes under its control. The discussion in this chapter is concerned with the high-level view, except for description of the format of Macintosh files and the consequences of that structure for file operations.

Overall file organization

The A/UX kernel (or any UNIX kernel) represents external storage to applications as a single, hierarchical volume having the root level designated by “slash” (/), the root directory. The one volume can contain multiple file systems. A file system, in this technical sense, is a combination of routines for manipulating files together with associated data structures; it provides support for high-level calls dealing with files (open, create, delete, and so on) that are under the domain of the file system. An A/UX file (or any UNIX file) is seen by the A/UX system as a stream of bytes. Any further structure within a file is created and maintained by applications that need to have such a structure.

In the UNIX design, subordinate file systems can be added to or removed from the one volume only by means of formal mount and unmount operations.

The Macintosh File Manager represents external storage to applications as a collection of volumes, each having an associated file system and driver. Each volume is associated with a physical device. The file system interprets high-level operations into low-level driver calls; the driver handles device-dependent requirements. Each volume contains an independent file-system hierarchy, the root of which is represented by the volume name. Applications call on the File Manager by means of A-line traps to manipulate the volumes and the files within them. Each file within the volume has a defined structure, consisting of two “forks,” a data fork and a resource fork, and a third element (a quasi-fork) containing the Finder information. In the Macintosh design, volumes are independent. The user can add or remove volumes (floppy disks, for instance) as desired. The system keeps track, in a general way, of these volumes.

Both designs organize files in a tree structure. Files are grouped into directories (in the A/UX environment) or folders (in the Macintosh environment). Directories and folders are functionally equivalent. A directory or folder can hold other directories or folders as well as files.

Pathnames and filenames

In both file-system designs, the location of any file within the complete file tree can be specified by a pathname. The pathname lists the sequence of directories or folders in hierarchical order and ends with the name of the file. (The pathname for a directory or folder ends with the name of the directory or folder.) In the A/UX environment, the full pathname starts with the root volume (/); in the Macintosh environment, the full pathname starts with a volume name.

Pathnames require a special character as a delimiter between directory or folder names and the filename. The A/UX pathname uses the slash (/) as a delimiter and the Macintosh pathname uses the colon (:). Here is an example of each type:

```
A/UX:          /users/fred/memos/tripmems
Macintosh:     fred's stuff:memos:trip memos
```

The restrictions for an A/UX filename depend on the type of file system in use on the physical volume where the file resides. The file system may be UFS, SVFS, or NFS; all are supported under the A/UX operating system.

A Macintosh filename consists of any sequence of 1 to 32 eight-bit characters, excluding colons (the pathname delimiter).

In System V, a filename consists of 1 to 14 seven-bit characters, excluding slashes (the pathname delimiter). With UFS, which is now the default file system created by the A/UX Installer, A/UX filenames can consist of 1 to 255 eight-bit characters (slashes excluded).

When comparing filenames, the A/UX file system distinguishes between the uppercase and lowercase versions of a character; the Macintosh file system does not distinguish between uppercase and lowercase characters. This fact poses a problem for programs, such as development tools and utilities, that assume a case-insensitive file-system environment. Although a space character can be used in an A/UX filename, practical considerations suggest that a space should not be used. For example, suppose that a user saves a text file under the name `my report` and attempts to access it under A/UX by using the `vi` editor. When the user enters

```
vi my report
```

the editor will not locate the file. The editor will create two new files, called `my` and `report`, or will access a file of either name, if present. To access `my report` under A/UX, the user must quote the filename or the space, as follows:

```
vi "my report"
```

or

```
vi my\ report
```

Because the space character is used as a practical delimiter between filenames by the shell programs that provide user interface throughout the A/UX (or any UNIX) system, blanks should not be used in filenames. However, Macintosh filenames routinely use spaces.

File permissions

Because UNIX is a multi-user system, every A/UX file has an associated set of file permissions. There are three categories of user: owner, group, and other. For each category, there are three types of permissions: read, write, and execute.

The Macintosh file system has no set of file permissions for user files; privileges are set for folders and volumes. (System files have restrictions on access.) The AppleShare access privilege structure was developed for use of files in a multi-user environment. AppleShare privileges are in three categories: “See Folders,” “See Files,” and “Make Changes.” AppleShare privileges apply only to folders (directories) and not to individual files.

Both these permission structures are present and independently settable for folders on the “/” volume. The UNIX permissions are the controlling permissions; the AppleShare privileges provide a secondary constraint on what may be done with a folder.

When A/UX files are viewed through A/UX Toolbox traps designed specifically for AppleShare (such as `_GetDirAccess`) folders will appear to have access privileges set. In brief, UNIX (A/UX) permissions and AppleShare privileges are mapped as shown in Table 6-1. Effectively, write permission equates to Make Changes, while the combination of read and execute permissions equate to the combination of See Files and See Folders.

◆ **Note** When AppleShare access privileges are set within a file system, the UNIX file permissions are not affected. Nor, when UNIX permissions are set, are the AppleShare access privileges affected. Table 6-1 shows only how the privileges and permissions appear when files are viewed *between* file systems. ◆

Table 6-1 A/UX permissions mapped to AppleShare privileges

A/UX Permissions			AppleShare Privileges	
Read	Write	Execute	See Folders and See Files	Make Changes
No	No	No	No	No
No	No	Yes	No	No
No	Yes	No	No	Yes
No	Yes	Yes	No	Yes
Yes	No	No	No	No
Yes	No	Yes	Yes	No
Yes	Yes	No	No	Yes
Yes	Yes	Yes	Yes	Yes

Extended file attributes

In the A/UX file system, a file has associated with it a general type—regular, directory, character or block special, or FIFO (First In, First Out)—but no special repository of information about the file. The Macintosh OS file system provides each file with a set of extended attributes used by the Finder and other system tools. These attributes include the file type, which among other things tells whether or not the file is executable; the file creator name; the screen location and icon ID, which the Finder uses to display the file icon; and the comment field, which contains text displayed when the user requests file information.

In order to accommodate the Macintosh environment's needs for such attributes, A/UX uses special file formats when storing a file of Macintosh OS origin. These formats preserve the extended file attributes.

The details of these file formats are given later in this chapter. The options available are as follows:

- Place all of the attribute information at various specified locations within the one new file, which also contains the file's data. (AppleSingle format.)
- Create two files, one containing the attribute information in specified locations, together with other information. The second file contains the data. (AppleDouble format.)
- Use additional special-purpose formats, one of which (the “triple” file) creates a special file to hold the attribute information.

The advantages of each of these formats are discussed in the section “Storing Files in the Macintosh OS and in the A/UX Operating System,” later in this chapter.

Text files

A text file created by a Macintosh application running under the A/UX Toolbox has these attributes:

- Each line is terminated by a return character (ASCII value 0x0D).
- The file's data is accompanied by a set of Finder information that includes the file's type and creator. The file type is 'TEXT' and the creator varies with the application.

A text file created by an A/UX program, such as `vi(1)`, has these attributes:

- Each line is terminated by a line-feed character (ASCII value 0x0A).
- The text file has no stored type or creator information (it is not in AppleDouble format). When such a file enters the A/UX Finder environment, it receives a file type and creator based on the rules described in the section “Automatic Conversion,” later in this chapter. A/UX text files are assigned file type 'TEXT' and creator 'tefi'.

Line-termination characters are translated on the fly when a file is moved between the two environments. Macintosh text-file return characters become line feeds, and A/UX text-file line feeds become return characters. Text editors and other programs that handle text find the expected line-termination character, depending on the environment in which the file is read and not on the actual termination character used in the file. When an application running under the A/UX Toolbox reads a text file, the lines will be terminated by return characters. When that application stores the file, the line-termination character used depends on the environment in which the file is placed. If the file goes into the A/UX environment, the termination characters will be line feeds. This behavior is essentially transparent to the user.

Mounting and unmounting floppy disks

Under A/UX (or any UNIX system), the file system can recognize a UNIX file structure on a floppy disk and grant appropriate access only if the disk is mounted. The mounting process provides, among other things, a specific location in the file tree for the files on the disk. To remove that file structure, the disk is unmounted. Mounting and unmounting are system operations, separate from physically inserting and removing the floppy disk. Mounted file structures do not appear as separate icons on the desktop.

Under the Macintosh file system, users are accustomed to inserting and removing floppy disks as needed. To the file system, each physical device (such as a floppy disk) is a separate volume and can be dealt with as an independent volume. If a disk has a recognizable file structure, then it is accessible without a formal mount operation, and removing the floppy disk does not require an unmount operation. (The floppy disk is implicitly unmounted when it is ejected.) Macintosh volumes appear as icons on the desktop.

A Macintosh OS file system cannot be mounted under the A/UX file tree. A Macintosh application cannot use the A/UX Toolbox file system to mount (or unmount) a Macintosh file volume (such as a floppy disk) as part of the A/UX file tree. As described in “Automatic Conversion,” later in this chapter, an individual Macintosh file can be placed under A/UX, after which the file becomes some variety of A/UX file.

An application may, by means of the appropriate A/UX system calls, mount, access, and unmount floppy disks under the A/UX file system by using A/UX file-handling methods. More typically, an application running under the A/UX Toolbox may deal with any number of floppy disks as Macintosh volumes in the usual way.

In short, when an application is running under the A/UX Toolbox, all floppy disk files that it “sees” (can access) are on Macintosh volumes. It does not “see” any A/UX files on floppy disks except by access through the A/UX file system, that is, unless those files have been mounted.

Storing files in the Macintosh OS and in the A/UX operating system

Under the Macintosh OS, a file consists of two forks: a data fork and a resource fork. In general, the data fork contains user data, such as the text in a word-processing document, and the resource fork contains resources used by the application. Resources include commonly used structures such as dialog boxes and icons, as well as the body of an application’s code. (See *Inside Macintosh*, Volume I, for a description of resources.)

Although a file can contain two forks, one of the two forks may be empty. A file that holds a document created by an application, for example, often contains only a data fork, with an empty resource fork. Similarly, a file that holds an executable application may contain only resources, with an empty data fork. Figure 6-1 illustrates the elements of a file under the Macintosh OS. Text in brackets in the figure represents elements that may be absent from the file.

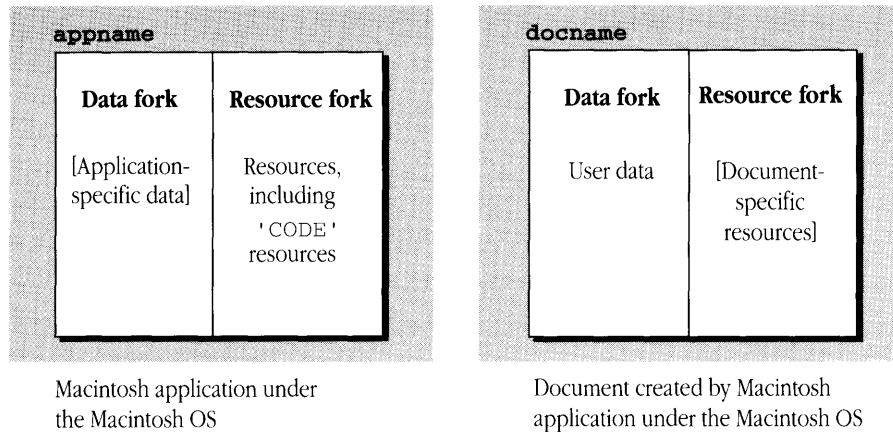


Figure 6-1 Elements of a file in the native Macintosh OS environment

The Macintosh OS file system also stores extended-file-attribute information in a separate record in the directory. See *Inside Macintosh*, Volume IV, for a description of the hierarchical file system (HFS) file-directory information. For a description of the obsolete Macintosh flat file system (MFS) and its file-directory information, see *Inside Macintosh*, Volume II.

The A/UX file structure makes no distinction between data and resources, and the A/UX directory structure makes no provision for the Macintosh file-attribute information. Apple has developed two standard file formats that you can use to store Macintosh-style files in A/UX:

- *AppleSingle format* All contents and file information are kept in a single file.
- *AppleDouble format* The contents of the data fork are stored in one file, known as the *data file*; resources and file-attribute information are stored in a separate file, known as the *header file*. The header file has the same name as the data file, except that the header file is prefixed with a percent sign (%).

The AppleDouble format is a good choice for text data and data to be shared with UNIX utilities, because the data fork is available as an isolated file. When moving an AppleDouble file pair with UNIX utilities, remember to move both files.

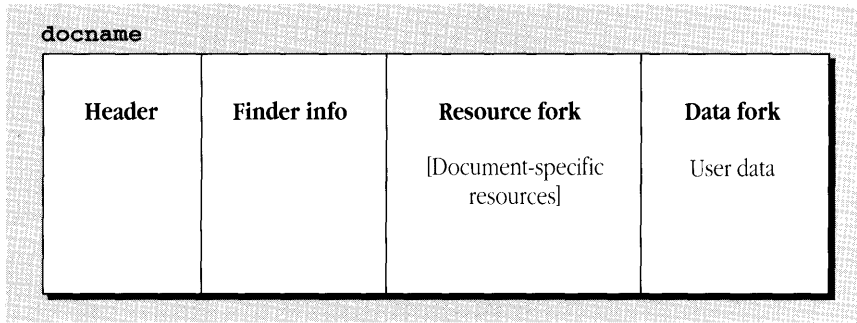
The internal formats of AppleSingle and AppleDouble files are discussed in the section “AppleSingle and AppleDouble Format Internals,” later in this chapter.

◆ **Note** From the point of view of an application or a user, the distinctions between the two file formats discussed here are not important. The A/UX Toolbox File Manager insulates applications from having to consider these details. ◆

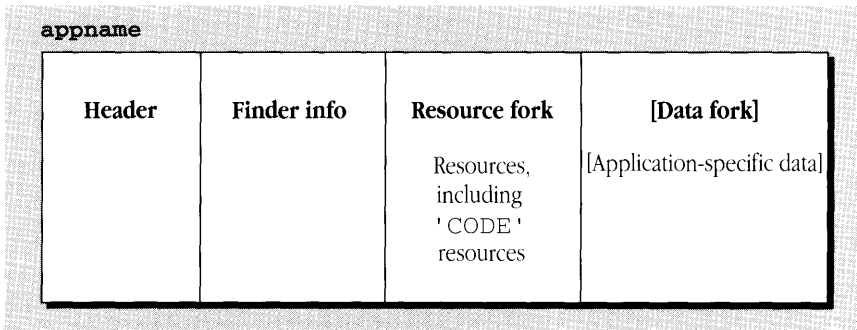
An A/UX file, standing alone, remains a Plain file but is recognized as an AppleDouble data file. (This convention allows A/UX Toolbox applications to access files created by conventional UNIX utilities, such as text editors.) An A/UX Toolbox application processing a Plain file may cause the creation of a header file for that data file in certain circumstances, resulting in an actual AppleDouble file. The four cases in which the file remains a Plain file are shown in the section “Automatic Conversion,” later in this chapter. If the combination of file type and creator is changed to anything other than a file type of 'COFF', 'SHEL', 'XAPP', or 'BIN', with a matching creator of 'A/UX', then resource information is generated and written to a header file with the same name as the data file, except that the header-file name is prefixed with a percent sign (%).

Like a Macintosh OS file, an A/UX AppleSingle file may contain both data and resources, data and no resources, or resources and no data. An AppleSingle file always contains file-information entries, although the entries for a newly created file might be undefined.

An AppleDouble data file is accompanied by a header file containing the file-attribute information. The header file can—but need not—contain resources. An AppleDouble header file can exist without an associated data file. Figures 6-2 and 6-3 illustrate the typical contents of AppleSingle and AppleDouble files in A/UX. Text in brackets in the figures represents elements that may be absent from the file.



AppleSingle document file



Macintosh binary application transferred to an A/UX AppleSingle file

Figure 6-2 Typical contents of an AppleSingle file

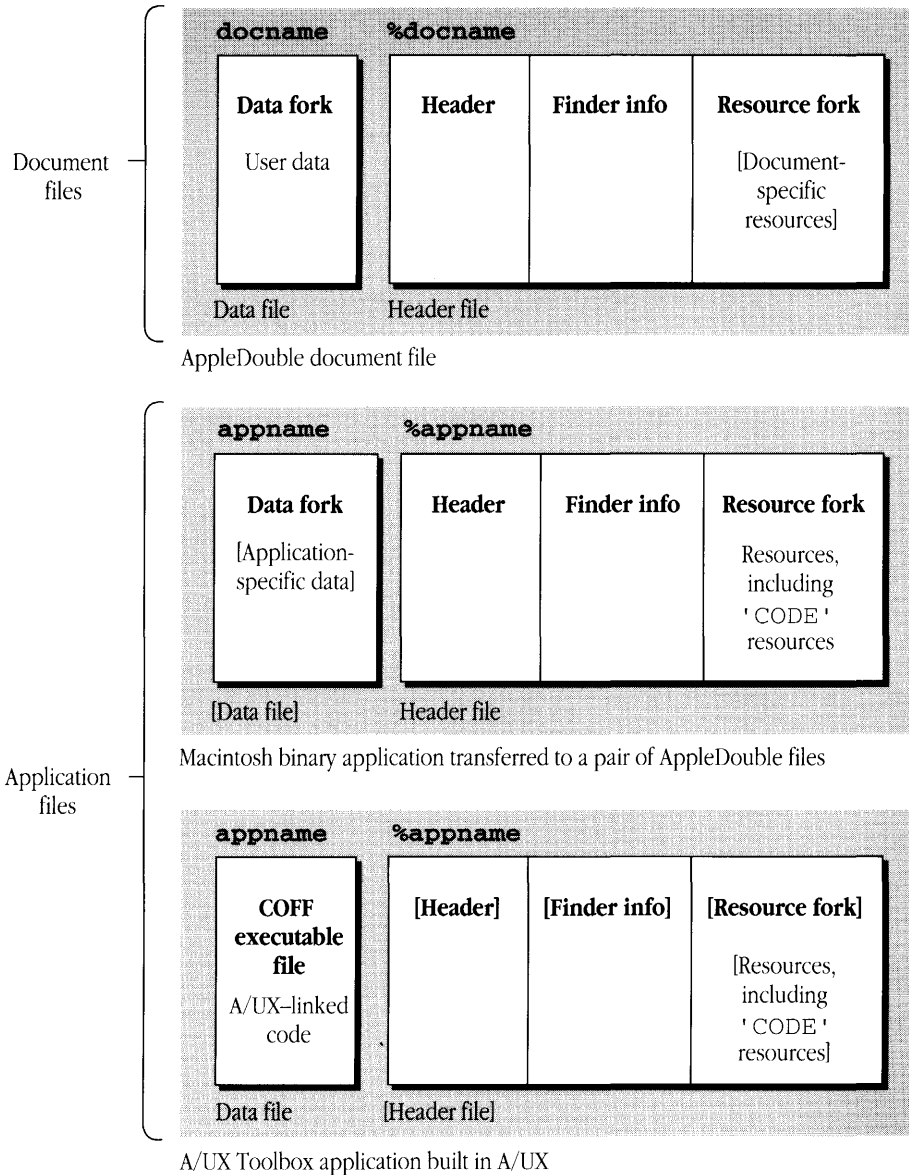
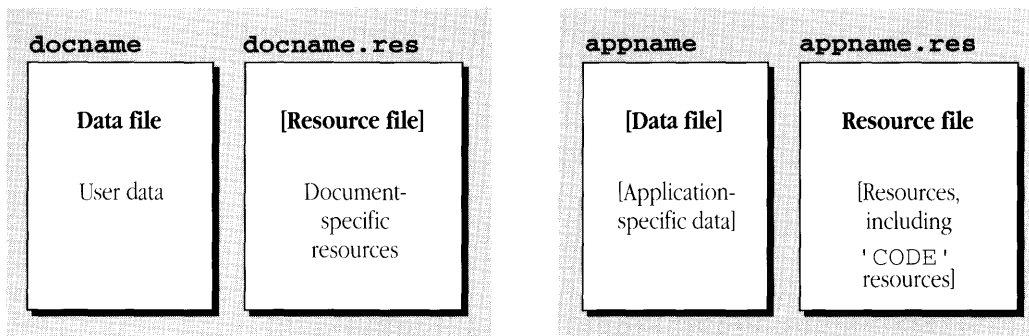


Figure 6-3 Typical contents of a pair of AppleDouble files

When you compile and link an application under A/UX, the result is a standard executable COFF file. The Macintosh OS will consider the COFF file to be an AppleDouble file. As mentioned earlier, so long as the type remains 'COFF' and the creator 'A/UX', no unnecessary header file is created.

If you have used a general-purpose utility to transfer files from the Macintosh OS to the A/UX operating system, you might also have Macintosh files stored in a simple A/UX format. The `kermit(1C)` utility, for example, transfers the two forks of a Macintosh file separately into a pair of A/UX files that follow neither AppleSingle nor AppleDouble format. The data fork is placed in one file, and the resource fork is placed in a file with the same name plus the extension `res`. (See *A/UX Command Reference* for a description of `kermit`.) For compatibility with other tools, the A/UX Toolbox file-conversion utility, `fcvt(1)`, recognizes this structure.

Figure 6-4 illustrates the possible contents of Macintosh files in simple A/UX format. Text in brackets in the figure represents elements that may be absent from the file.



Document file either created by an A/UX Toolbox application and converted to simple A/UX format, or created in Macintosh OS and transferred to a simple A/UX file

Macintosh binary application transferred to a simple A/UX file

Figure 6-4 Elements of Macintosh data and resource files in simple A/UX format

When you create a Macintosh-compatible file under A/UX, the A/UX Toolbox uses these formatting strategies:

- In almost all circumstances, the A/UX File Manager creates AppleSingle files. Therefore, when an A/UX Toolbox application creates a file through File Manager calls, it creates an AppleSingle file.
- When the File Manager receives a request to open an AppleDouble data file, it automatically looks for the associated header file. The application does not specify the format of the file when issuing the call; the File Manager itself checks the format of the file.
- The A/UX implementation of the resource compiler, `rez(1)`, creates only an AppleSingle file. See Chapter 3, “A/UX Toolbox Utilities and Extensions,” and Appendix E, “Resource Compiler and Decompiler,” for a description of `rez`.

The A/UX Toolbox provides the following utilities for converting files and manipulating their formats:

- The `fcnvt(1)` utility converts files among AppleSingle format, AppleDouble format, and four other formats.
- The `setfile(1)` utility adds or changes the file type and creator of an AppleSingle file or an AppleDouble header file.

See Chapter 3, “A/UX Toolbox Utilities and Extensions,” for more details.

Automatic conversion

When a file is transferred from one file system to the other, for example by having its icon dragged on the desktop, the file is automatically converted. When a Macintosh file is placed in the A/UX file system, it goes into one of three formats: AppleDouble, AppleSingle, or Plain. In brief, the AppleDouble format produces two files, one containing data and the other containing resource and Finder information; the AppleSingle file contains everything in one file; and the Plain file contains data only and corresponds to the data file of an AppleDouble pair. Which format is used depends on information kept in three fields of the extended-attribute portion of the file: the type, the creator, and the flag setting. The process is summarized in Table 6-2.

Table 6-2 Automatic conversion of Macintosh files

Attributes			
Type	Creator	Flag setting	Resulting format
'APPL'	[any]	No INITs=1	AppleDouble
'TEXT'	[any]	N/A	AppleDouble (or Plain)
'A/UX'	[any]	N/A	AppleDouble
'COFF'	'A/UX'	N/A	Plain
'SHEL'	'A/UX'	N/A	Plain
'XAPP'	'A/UX'	N/A	Plain
'BIN'	'A/UX'	N/A	Plain
All others			AppleSingle

As Table 6-2 shows, there are three ways to ensure conversion to AppleDouble format:

- Set the type to 'APPL' and set the flag as shown. (The No INITs flag is bit 7 of `Info.fdfFlags`.)
- Set the type to 'TEXT'.
- Set the type to 'A/UX'.

The first way allows programs (such as CommandShell) to have their own icon while ensuring conversion to the AppleDouble format. The second and third ways allow files that may have system extensions to be converted to AppleDouble format. In the second instance, the entry shows a special exception that occurs when there is no resource fork. When a Macintosh OS application processes an A/UX text file, the file remains a Plain file unless the application creates resource information for that file, in which case the file system makes a second file to hold that information. The two files constitute an AppleDouble pair.

An A/UX file transferred into the Macintosh file system simply becomes a standard Macintosh file. File permissions are lost unless they have been explicitly saved as foreign privileges. Text files and shell files have their line-termination characters automatically translated from line-feed to return characters, as described in “Text Files,” earlier in this chapter.

AppleSingle and AppleDouble format internals

AppleSingle format stores the data, resources, and attributes of a Macintosh file in a single A/UX file. AppleDouble format stores a file's data in one file and stores its resources and attributes in another file.

This section uses these terms:

- **Home file system** is the file system for which the file's contents were created, not necessarily the file system in which the file was created. The Macintosh file system is the home file system for all A/UX Toolbox applications and all documents created with A/UX Toolbox applications.
- **Foreign file system** is the other file system that will store or process the file. The UNIX file systems are the foreign file systems for all A/UX Toolbox applications and all documents created with A/UX Toolbox applications.

AppleSingle format

In AppleSingle format, all of a file's contents and attributes are stored in a single file in the foreign file system.

An AppleSingle file consists of a header followed by one or more data entries. The header consists of several fixed fields and a list of entry descriptors, each pointing to an entry. Table 6-3 describes the contents of an AppleSingle file header.

Table 6-3 AppleSingle file header

Field	Length
Magic number	4 bytes
Version number	4 bytes
Home file system	16 bytes, ASCII encoded
Number of entries	2 bytes
Entry descriptor for each entry:	
Entry ID	4 bytes
Offset	4 bytes
Length	4 bytes

Byte ordering in the file-header fields follows MC68000, MC68020, and MC68030 conventions. Here is a description of each field:

- *Magic number* This field, modeled after the A/UX magic-number feature, specifies the file's format. Apple has defined the magic number for AppleSingle format as 0x00051600.
- *Version number* This field allows for the evolution of AppleSingle format. This section describes version 0x00010000.
- *Home file system* This field defines the home file system. It contains a 16-byte ASCII string, which is not preceded by a length byte but which can be padded with spaces. Apple Computer has defined these strings:

Macintosh	'Macintosh'	or	0x4D616369	0x6E746F73	0x68202020 ...
ProDOS	'ProDOS'	or	0x50726F44	0x4F532020	0x20202020 ...
MS-DOS	'MS-DOS'	or	0x4D532D44	0x4F532020	0x20202020 ...
UNIX	'Unix'	or	0x556E6978	0x20202020	0x20202020 ...
VMS™	'VAX VMS'	or	0x56415820	0x564D5320	0x20202020 ...

All A/UX Toolbox applications work with files whose home file system is Macintosh.

- *Number of entries* This field reports how many different entries are included in the file. Its value is an unsigned 16-bit number. If the number of entries is any number other than 0, then that number of entry descriptors immediately follows.
- *Entry ID* This field defines what the entry is. The field holds an unsigned, 32-bit number. Apple Computer has defined a set of entry IDs and their values:

Data fork	1	standard Macintosh data fork
Resource fork	2	standard Macintosh resource fork
Real name	3	file's name in its home file system
Comment	4	standard Macintosh comments
Icon, B&W	5	standard Macintosh black-and-white icon
Icon, color	6	Macintosh color icon
file info	7	file information: attributes and so on
Finder info	9	standard Macintosh Finder information

Apple reserves the range of entry IDs from 0 to 0x7FFFFFFF. The rest of the range is available for other definitions. Apple does not arbitrate the use of the rest of the range.

Icon entries do not appear in most files because they are typically stored as a bundle in the resource fork of the application file.

The structure of the “file info” entry is different for each home file system. For Macintosh HFS files, the entry is 16 bytes long and consists of three long-integer dates (create date, last modification date, and last backup date) and a long integer containing 32 Boolean flags. Where 0 is the least-significant bit and 31 is the most-significant bit, bit 0 of the Macintosh “file info” entry is the Locked bit, and bit 1 is the Protected bit. Figure 6-5 illustrates the formats for Macintosh HFS, A/UX, MS-DOS, and ProDOS “file info” entries.

The “Finder info” entry consists of 16 bytes of Finder information followed by 16 bytes of extended Finder information (the fields `ioFlFndrInfo` followed by `ioFlXFndrInfo`, as returned by the `PBGetCatInfo` call). These fields contain extended-file-attribute information. See *Inside Macintosh*, Volume VI, for a description of the subfields in these fields. Newly created files contain zeros in all “Finder info” fields. When you are creating a file whose home file system is Macintosh, you can use 0 in any subfield whose value is unknown, except that you should set the `fdType` and `fdCreator` subfields. Values should be set by means of standard File Manager calls such as `SetFInfo` and `PBSetCatInfo`.

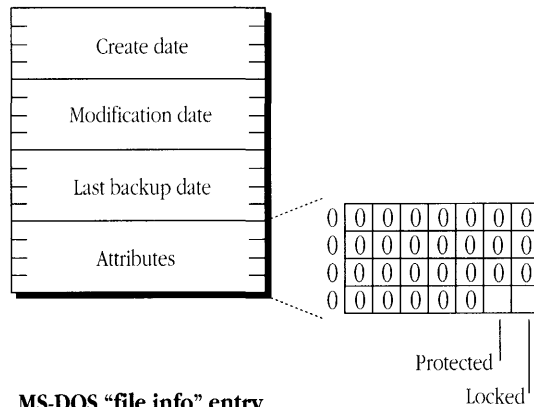
- *Offset* This field contains an unsigned 32-bit number that shows the offset of the beginning of the entry’s data from the beginning of the file.
- *Length* This field contains an unsigned 32-bit number that shows the length of the data in bytes. The length can be 0.

The entry data follows all of the entry descriptors. The data in each entry must be in a single, contiguous block. You can leave holes in the file for later expansion of data. For example, even if a file’s comment field is only 10 bytes long, you can place the offset of the next field 200 bytes beyond the offset of the comment field, to leave room for the comment to grow to its maximum length of 200 bytes.

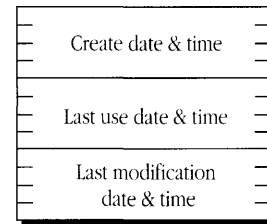
The entries can appear in any order, but you can maximize the efficiency of file access by following these recommendations:

- Put the data fork entry at the end of the file. The data fork is the most commonly extended entry, and it is easier to increase its length if it is the last thing in the file.
- Put the entries that are most often read, such as “Finder info,” as close as possible to the header, to increase the probability that a read of the first block or two will retrieve these entries.

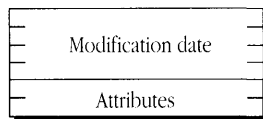
Macintosh “file info” entry



A/UX “file info” entry



MS-DOS “file info” entry



ProDOS “file info” entry

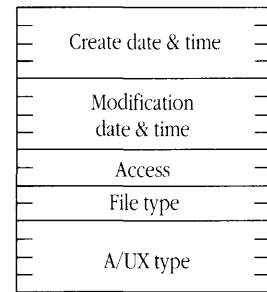


Figure 6-5 Formats for “file info” field entries

AppleDouble format

In AppleDouble format, the file’s data fork is stored in a file called the *AppleDouble data file*, and the file’s attributes and resources are stored in a separate file called the *AppleDouble header file*.

The AppleDouble data file contains the data fork, in exactly the form in which it appears in a Macintosh file, with no extra header.

The AppleDouble header file has the same format as an AppleSingle file, except that it contains no data fork entry. The magic number for an AppleDouble header file is 0x00051607. The entries in the header file can appear in any order. It is usually more efficient to put the resource fork at the end of the file because the resource fork is the entry most likely to expand.

Filename conventions

This section describes the conventions for naming Macintosh files in the A/UX environment. The filename needs are slightly different for the AppleSingle and AppleDouble formats. These considerations apply to both:

- Embedded spaces in filenames transfer between the Macintosh and UNIX environments. Filenames with embedded spaces are legal but cause problems in UNIX because UNIX commands consider that spaces delimit a filename. Changing the embedded spaces to less problematic characters may be preferable to leaving the spaces.
- Filenames containing characters with ASCII values greater than 127 may not be recognized by some UNIX implementations. Use character substitution to replace any illegal character with an underscore (`_`).
- Because different UNIX file systems impose different length restrictions, do not explicitly truncate the name to a specified length; allow the truncation to be done by the file-handling functions such as `creat(2)` and `open(2)`. Remember that A/UX supports three file systems, one of which (UFS) allows filenames to contain up to 255 characters.

AppleSingle format does not specify an algorithm for deriving an AppleSingle filename from the file's "real" name as stored on a native Macintosh volume. File systems (and your applications) can exercise some discretion in choosing filenames because the file's original name can be stored as data in the file and retrieved as necessary.

The general strategy for AppleDouble-format filenames is to derive the data-file name from the file's original Macintosh name and then to derive the header-file name from the data-file name. The most important connection is between the two AppleDouble filenames, which must often be treated as a single unit and therefore must be clearly connected.

- For an AppleDouble data-file name, the general considerations apply.
- For an AppleDouble header-file name, prefix a single percent sign (`%`) to the AppleDouble data-file name. If necessary, truncate the last character to keep the filename within the legal length range. The result is that the two files are kept together in a single subdirectory.



Appendix A: Additional Reading

Information sources / A-2

Required references / A-4

Supplementary references / A-5

This appendix tells you where to get more information about the A/UX Toolbox and lists required and supplementary documentation.

Information sources

APDA is Apple's source for a wide selection of Apple and third-party development tools and information.

APDA offers convenient worldwide access to more than 300 development tools, resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the *APDA Tools Catalog* featuring Apple and third-party development products. Ordering is easy; there are no membership fees, and signed agreements are not required to order most products. APDA offers convenient payment and shipping options, including site licensing. Anyone can receive the *APDA Tools Catalog* by contacting APDA. To order products or get additional information, contact

APDA

Apple Computer, Inc.

20525 Mariani Avenue, M/S 33-G

Cupertino, California 95014-6299 U.S.A.

800-282-2732 (U.S.)

800-637-0029 (Canada)

408-562-3910 (International)

408-562-3971 (Fax)

171-576 (TELEX)

AppleLink: APDA

Apple offers two developer programs: Associates and Partners.

- **Associates Program** A mainstream program for commercial developers; convenient access to essential technical and marketing information.

The Associates Program, Apple's mainstream program for developers of commercial products, is a convenient and cost-effective way to access essential technical and marketing information. The Associates Program offers self-help technical support, keeps you up-to-date with the latest products and technical documentation, and facilitates access to the Apple developer community through AppleLink. Associates also receive discounts on hardware, lowering the cost of getting started on a development project.

Who Should Apply? This program is designed for developers who are working on a standardized, commercial product which is sold publicly. Associates must deliver a product within two years of joining the program.

- **Partners Program** A program for Apple-selected strategic commercial developers. The Partners program is open to Apple-selected commercial developers. In addition to receiving the same program benefits as Associates, Partners receive direct technical support via electronic mail.

Who Should Apply? Apple limits membership in the Partners Program to developers of commercial, standardized products who contribute to Apple's long-term product strategies and business objectives. Partners are expected to focus their resources on the development of Apple-compatible products. Partners must deliver a product within two years of joining the program.

For further information, write to

Apple Developer Programs
Apple Computer, Inc.
20525 Mariani Avenue, M/S 51-W
Cupertino, California 95014-6299 U.S.A.

Apple also offers courses at Apple Developer University. You do not need to be in a developer program to attend. For information, write to

Developer University Registrar
Apple Computer, Inc.
20525 Mariani Avenue, M/S 51-M
Cupertino, California 95014-6299 U.S.A.

Required references

This section lists books that you will need for developing software under the A/UX system or creating hardware interfaces to Macintosh computers running the A/UX system. The guide *Road Map to A/UX*, listed in this section, provides a detailed description of each A/UX book published by Apple Computer.

A/UX Command Reference. Apple Computer, Inc., 1992. A collection of reference pages, also known as *manual pages*, for A/UX user commands and games. This document corresponds to Sections 1 and 6 of the traditional UNIX user manual. The information in *A/UX Command Reference* is provided on-line with A/UX Release 3.0. This document is available as part of the A/UX Administration Manuals product.

A/UX Essentials. Apple Computer, Inc., 1992. A user's introduction to A/UX Release 3.0.

A/UX Programmer's Reference. Apple Computer, Inc., 1992. A collection of reference pages, also known as *manual pages*, in two volumes for A/UX system calls, subroutines, file formats, and miscellaneous facilities. This document corresponds to Sections 2 through 5 of the traditional UNIX user manual. The information in *A/UX Programmer's Reference* is provided on-line with A/UX Release 3.0. This document is available as part of the A/UX Programming Manuals product.

A/UX Programming Languages and Tools, Volumes 1 and 2. Apple Computer, Inc., 1987, 1990, and 1992. A description of the A/UX C and Fortran languages and the libraries and tools used for program development and maintenance. These documents are available as part of the A/UX Programming Manuals product.

Inside Macintosh, Volumes I through III. Addison-Wesley, 1985. A complete description of the architecture and operation of the 128K and 512K Macintosh computers, including the ROM routines.

Inside Macintosh, Volume IV. Addison-Wesley, 1986. An update to the original volumes, covering the Macintosh 512K enhanced and Macintosh Plus computers.

Inside Macintosh, Volume V. Addison-Wesley, 1987. An update to Volumes I through IV, covering the Macintosh SE and Macintosh II computers.

Inside Macintosh, Volume VI. Addison-Wesley, 1991. An update to Volumes I through V, covering Macintosh System 7. An on-line version is available from APDA.

Inside the Macintosh Communications Toolbox. Addison-Wesley, 1990. A guide to the Communications Toolbox, which is Apple's communications development platform and is an integral part of System 7.

Road Map to A/UX. Apple Computer, Inc., 1992. A guide to the features of A/UX and to the A/UX documentation.

Supplementary references

This section lists useful books available for developing software under, or creating hardware interfaces to, the A/UX system. The list is not exhaustive. Many other excellent books are available on various aspects of developing under System V UNIX, developing with shell languages, and making use of BSD features. Useful books are also available on developing under the Macintosh User Interface Toolbox and Macintosh OS. The Macintosh Programmer's Workshop (MPW) references document a UNIX-like development environment that runs under the Macintosh OS.

The Motorola manuals listed are a selection of documentation provided by Motorola Corporation, useful to hardware developers and software developers working close to the hardware.

A/UX C89 C. APDA, 1991. Describes the implementation of Apple's ANSI-compliant C compiler. This document is available as part of the A/UX Developer's Tools product.

A/UX Development Tools. APDA, 1991. Describes the suite of enhanced development tools available for A/UX, including the assembler, linker, Commando interface, and other important tools. This document is available as part of the A/UX Developer's Tools product.

Building A/UX Device Drivers. APDA, 1992. A reference for developing device drivers for A/UX systems. Included is the source code for drivers used in A/UX. This document is available as part of the A/UX Device Drivers Kit product.

Chernicoff, Stephen. *Macintosh Revealed*, Volumes I through III. Hayden Book Company, 1985, 1987. A guide to writing programs that use the Macintosh User Interface Toolbox and Macintosh OS. Later editions have tracked developments of the Macintosh OS.

Designing Cards and Drivers for the Macintosh Family. Second Edition. Addison-Wesley, 1990. A general reference for developing expansion cards and device drivers for the Macintosh family of computers.

Harbison, Samuel P., and Guy L. Steele, Jr. *C: A Reference Manual*. Third Edition. Prentice-Hall, Inc., 1991. A standard reference book for the C language with the AT&T extensions used in most UNIX operating-system environments. The third edition covers both "traditional" and ANSI C.

Inside Macintosh X-Ref. Revised Edition. Addison-Wesley, 1991. A key to eleven of the Addison-Wesley books that document the Macintosh: *Inside Macintosh*, Volumes I through VI, *Programmer's Introduction to the Macintosh Family*, *Technical Introduction to the Macintosh Family*, *Inside the Macintosh Communications Toolbox*, *Guide to Macintosh Family Hardware*, and *Designing Cards and Drivers for the Macintosh Family*. Provides a general index to these volumes, a list of routines that move or purge memory, a list of system traps, a list of global variables, and a glossary.

- Kernighan, Brian W., and Rob Pike. *The UNIX Programming Environment*. Prentice-Hall, Inc., 1984. A guide with valuable information, including chapters on shell programming, `lex`, `yacc`, and text formatting.
- Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*. Second Edition. Prentice-Hall, Inc., 1988. An update of the original, official C manual, with tutorial information. This edition covers ANSI C, in addition to updating the first edition.
- Knaster, Scott. *How to Write Macintosh Software*. Hayden Book Company, 1986. A guide to the oddities of programming the Macintosh (non-A/UX), with full discussion of memory, stack, and pointer concepts.
- Knaster, Scott. *Macintosh Programming Secrets*. Hayden Book Company, 1986. A guide to the concepts and ideas of (non-A/UX) Macintosh programming, use of color, and sending PostScript commands to a PostScript laser printer.
- Macintosh Programmer's Workshop 3.0 Assembler Reference*. APDA, 1988. A reference on the MPW Assembler and its tools. This document is available as part of the MPW product.
- Macintosh Programmer's Workshop 3.0 C Reference*. APDA, 1988. A description of the MPW C Compiler and tools that let you write C programs that use the Pascal routines in the Macintosh ROM. The C language for MPW 3.0 and that for A/UX are closely linked. This document is available as part of the MPW product.
- Macintosh Programmer's Workshop 3.1 Pascal Reference*. APDA, 1988. A description of the Pascal Compiler and tools. This document is available as part of the MPW product.
- Macintosh Programmer's Workshop 3.1 Reference*. APDA, 1988. A full description of how to use the MPW program preparation tools. This document is available as part of the MPW product.
- Macintosh Technical Notes*. Apple Computer, Inc., 1984–1990. A set of technical bulletins distributed at no charge by Apple Computer to all affiliated developers. Available through APDA.
- MacsBug 6.2 Reference and Debugging Guide*. APDA, 1991. A complete description of the MacsBug debugger.
- Manis, Rod, and Marc H. Meyer. *The UNIX Shell Programming Language*. Howard W. Sams & Co., 1986. A clear exposition of shell programming, as of System V, Release 2.
- MC68020 32-Bit Microprocessor User's Manual*. Motorola Corporation, 1985. A detailed description of the MC68020 CPU for hardware and software engineers.
- MC68030 32-Bit Microprocessor User's Manual*. Motorola Corporation, 1987. A detailed description of the MC68030 CPU for hardware and software engineers.
- MC68851 Paged Memory Management Unit User's Manual*. Motorola Corporation, 1985. A detailed description of the Paged Memory Management Unit (PMMU) for hardware and software engineers.

- MC68881 Floating-Point Coprocessor User's Manual*. Motorola Corporation, 1985. A description of the instruction set and addressing conventions used by the MC68881 floating-point coprocessor, which is used in the Macintosh II.
- Othmer, Konstantin, and Jim Strauss. *Debugging Macintosh Software with MacsBug*. Addison-Wesley, 1991. A lucid description of techniques and tricks for using MacsBug.
- Programmer's Guide to MultiFinder*. APDA, 1988. A guide to writing applications compatible with MultiFinder; applicable to the A/UX Finder.
- Programmer's Introduction to the Macintosh Family*. Addison-Wesley, 1987. A programmer's technical overview of the Macintosh system (non-A/UX), introducing the most important features of the Macintosh User Interface Toolbox and Macintosh OS.
- ResEdit v. 2.1*. APDA, 1991. Describes the ResEdit resource editor.
- Technical Introduction to the Macintosh Family*. Addison-Wesley, 1987. An introduction to the hardware and software design of the Macintosh family of computers.



Appendix B: Toolbox Contents

This appendix lists directories and files that are part of the A/UX Toolbox or that are of special interest in application development.

The large text file `/FILES` contains an annotated list of files in A/UX Release 3.0. You can explore this file to get further information about the contents of the directories listed here.

The list of files given in this appendix is not exhaustive, but is meant to give a general view of directories and files of interest. To obtain further information, check the `/FILES` list, use the Commando facility associated with the utilities, and consult the on-line and printed manual pages.

`/mac`

The major directories relating to the A/UX Toolbox.

`/mac/bin`

The executables and associated resource files needed by the A/UX Toolbox and A/UX Finder, which include files for logging in with the CommandShell and 24-bit CommandShell applications, files for executing the Commando function, and several utilities, some of which are discussed in Chapter 3.

<code>changesize</code>	A utility that changes the 'SIZE' attribute.
<code>rez</code>	Resource compiler. (See Appendix E.)
<code>derez</code>	Resource decompiler. (See Appendix E.)
<code>fcnvt</code>	A utility that performs file conversion.
<code>launch</code>	A utility that launches a Macintosh binary application.
<code>setfile</code>	A utility that sets the file creator and type for a file.
<code>startmac</code>	A program that provides the A/UX environment.
<code>startmac24</code>	A program that provides the 24-bit A/UX environment.
<code>TextEditor</code>	Macintosh-style text editor.

`/mac/lib`

Specialized Macintosh files.

`/mac/lib/SystemFiles`

Equivalent to the Macintosh System Folder, with A/UX Finder equivalents of the system files.

`/mac/lib/cmdo/*`

Directories of Commando dialog boxes.

`/mac/lib/rincludes`

Contains resource header files. (See Chapter 3 and Appendix E.)

`scripttypes.r` Resource header file for Script Managers.

`systypes.r` Resource header file.

`types.r` Resource header file, generic.

`/mac/lib/sessiontypes`

Session-type description files. (See `Login` documentation.)

`/mac/src`

Sample Macintosh application sources.

`examples` Example application sources, resource, and makefile. (See Chapter 2.)

`sndDemo` Sound demonstration example application sources, resource, sound file, and makefile. (See Chapter 2.)

`/mac/sys/*`

Macintosh system files and directories relating to the A/UX Toolbox and A/UX Finder.

`/usr/lib`

Libraries for programmer use, some relating to the A/UX Toolbox.

`libmac.a` Code for accessing toolbox, nonshared archive.

`libmac_s.a` Code for accessing toolbox, shared version (host).

`libc.a` Standard C library, nonshared archive.

`libc_s.a` Standard C library, shared version (host).

`/shlib`

Contains shared-library executables.

`libmac_s` Executable shared code (target) for accessing toolbox, linked by `libmac_s.a`.

`libc_s` Executable shared code (target) for standard C library, linked by `libc_s.a`.

`/dev/uinter0`

Special file for user interface device used internally by the A/UX Toolbox.

/usr/include/mac

Library of header files that define the constants, types, and functions used by the A/UX Toolbox C implementation of the Macintosh ROM routines.

asd.h	Calls to Macintosh resource material in /usr/lib/libmr.a.
aux.h	Definitions for AUXDispatch.
aux_rsrc.h	UNIX calls for Macintosh resource material in /usr/lib/libmr.a.
controls.h	Control Manager.
desk.h	Desk Manager.
devices.h	Device Manager.
dialogs.h	Dialog Manager.
dtask.h	Deferred Task Manager.
errors.h	System Error Handler.
events.h	Toolbox Event Manager.
files.h	File Manager.
fonts.h	Font Manager.
gestalt.h	Gestalt Manager.
lists.h	List Manager.
memory.h	Memory Manager.
menus.h	Menu Manager.
notify.h	Notification Manager.
osevents.h	Operating System Event Manager.
osutils.h	Operating System Utilities.
packages.h	Package Manager, including Binary-Decimal Conversion Package, Disk Initialization Package, International Utilities Package, Standard File Package.
palettes.h	Palette Manager.
picker.h	Color Picker.
printing.h	Printing Manager.

<code>printtraps.h</code>	Print traps.
<code>processes.h</code>	Process Manager.
<code>quickdraw.h</code>	32-Bit QuickDraw with Color QuickDraw.
<code>resources.h</code>	Resource Manager.
<code>retrace.h</code>	Vertical Retrace Manager.
<code>romdefs.h</code>	Definitions for ROMs.
<code>scrap.h</code>	Scrap Manager.
<code>script.h</code>	Script Manager.
<code>segload.h</code>	Segment Loader.
<code>serial.h</code>	Serial Driver.
<code>shutdown.h</code>	Shutdown Manager.
<code>slots.h</code>	Slot Manager.
<code>sm.h</code>	Sound Manager.
<code>soundinput.h</code>	Sound Manager input.
<code>soundinputpriv.h</code>	Sound Manager input privileges.
<code>strings.h</code>	String conversion routines.
<code>sysequ.h</code>	Low-memory equates.
<code>textedit.h</code>	TextEdit.
<code>timer.h</code>	Time Manager.
<code>toolutils.h</code>	Toolbox Utilities.
<code>traps.h</code>	List of Macintosh traps.
<code>types.h</code>	Type definitions.
<code>video.h</code>	Video Driver.
<code>vmcalls.h</code>	Memory Manager virtual memory.
<code>windows.h</code>	Window Manager.

`/usr/lib/libmr.a`

Declarations and routines for reading Macintosh resources.



Appendix C: Implementation Notes

The A/UX Finder and Toolbox applications / C-2

Running an A/UX Toolbox application / C-2

Converting between C and Pascal conventions / C-7

This appendix describes how the A/UX Toolbox simulates the Macintosh environment. You can use the A/UX Toolbox without the information in this appendix, but you will need this information if you are writing an application that contains assembly-language routines or deviates from recommended Macintosh programming practices.

This appendix covers two main topics: running A/UX Toolbox applications, and converting between the Pascal-language conventions used by the Macintosh ROM and the C-language conventions typically used in A/UX.

The A/UX Finder and Toolbox applications

The A/UX Finder must be running to support execution of A/UX Toolbox applications. A/UX Toolbox applications cannot be launched without support of the A/UX Finder.

System 7-compatible applications will execute under A/UX 3.0; however, System 7-friendly applications can take advantage of the advanced features of A/UX 3.0. Developers should aim at the latter standard. Requirements for applications are listed in Chapter 4, “Compatibility Guidelines.”

Running an A/UX Toolbox application

The A/UX kernel contains a special user-interface device driver, `/dev/uinter0`, that handles communications between A/UX Toolbox applications and the kernel. The driver provides `ioctl(2)` functions. (The section “Serial Driver” in Chapter 5 describes some `ioctl` functions.) The A/UX Toolbox library routines make calls to this device driver to provide special control for the Macintosh environment.

An A/UX Toolbox application uses a special initialization routine that opens the user interface device driver and issues a series of setup instructions before starting the program itself. The initialization routine is in `/usr/lib/macCRT0.o`. Each A/UX Toolbox application, including `launch(1)`, is linked with this file rather than with `/lib/crt0.o`, which is used by non-Toolbox A/UX applications.

Once an A/UX Toolbox application is running, most A/UX Toolbox functions are called through an MC680x0 exception, known as an *A-line trap*, the same way that ROM code is called in the Macintosh environment. In the A/UX environment, however, trap handling must be routed through the kernel.

User interface device driver

The user interface device driver, `/dev/uinter0`, performs these functions:

- *Memory mapping* When an application is started, the device driver establishes memory mapping for the screen buffer and ROM code, and memory for the Macintosh environment.
- *Event-queue handling* The driver contains its own event-queue handler, similar to the Macintosh OS Event Manager. The driver's event-queue handler supports the queue-access routines of the Macintosh OS Event Manager. The driver posts mouse and keyboard events.
- *Cursor tracking* The device driver enables vertical retrace interrupts and tracks the cursor at each interrupt. The cursor data is shared by the kernel and the application.
- *A-line trap dispatching* During startup, the driver installs in shared memory a pointer to the A-line trap handler. When the kernel identifies an exception as a Macintosh ROM call, it copies the return address from the kernel stack to the user stack and invokes the trap handler. For more information on trap dispatching, see “A-Line Traps” later in this appendix.

Initialization routine

The A/UX Toolbox initialization routine in `/usr/lib/macCRT0.o` performs these steps:

1. Calls `set42sig(3)`, which invokes BSD 4.2 signaling conventions.
2. Attaches to the shared data segment.
3. Opens the device driver and invokes the initialization steps described in the preceding section, “User Interface Device Driver.”
4. Initializes the dispatch tables and the Macintosh global variables.
5. Initializes various A/UX Toolbox modules.
6. Calls the application's main routine.

A-line traps

The primary function of the A/UX Toolbox is to make available to programs running under A/UX the Macintosh support code described in *Inside Macintosh*. Most of the support code represents routines built into the Macintosh ROM and available as A-line traps, that is, MC680x0 opcodes in the range 0xA000 to 0xFFFF.

Under the standard Macintosh OS, A-line traps are routed by the CPU to an exception handler. The exception handler uses a pair of dispatch tables (one for User Interface Toolbox routines and one for Macintosh OS routines) to route the A-line traps either to the ROM or to a ROM patch. A ROM patch is a change or bug fix to the Macintosh ROM.

In the standard Macintosh OS, the patches are stored in the System file. During startup, the patches are loaded into memory, and the dispatch tables are updated as necessary to point to patch routines rather than to ROM code. See *Inside Macintosh*, Volumes I, II, IV, and V, for descriptions of the dispatch tables.

Because all exceptions put the CPU into supervisor mode, an A-line trap in A/UX must be handled by the kernel. When the kernel recognizes an exception as a Macintosh A-line trap, it invokes a trap handler that resides in user process memory, leaving the processor in user mode.

The ROM dispatch tables in A/UX use two sets of ROM patches, the standard set and the A/UX set. The standard set incorporates the standard Macintosh ROM changes and the A/UX set accesses native ROM calls directly or provides A/UX alternatives, as appropriate. As each application is started, startup files build dispatch tables from data in the A/UX Toolbox libraries and the System file. No action by the application is necessary. An application can install its own patches to the tables.

◆ **Note** A-line traps cannot be called by UNIX device drivers. ◆

Figure C-1 illustrates the A-line trap-handling sequence in A/UX. The A/UX trap-dispatch code uses the application's trap-dispatch tables to route an A-line trap to one of two places:

- *ROM* If the trap has no A/UX alternative, the table points to the ROM code.
- *User RAM* If the trap has an A/UX alternative, the table points to the alternative routine in user RAM.

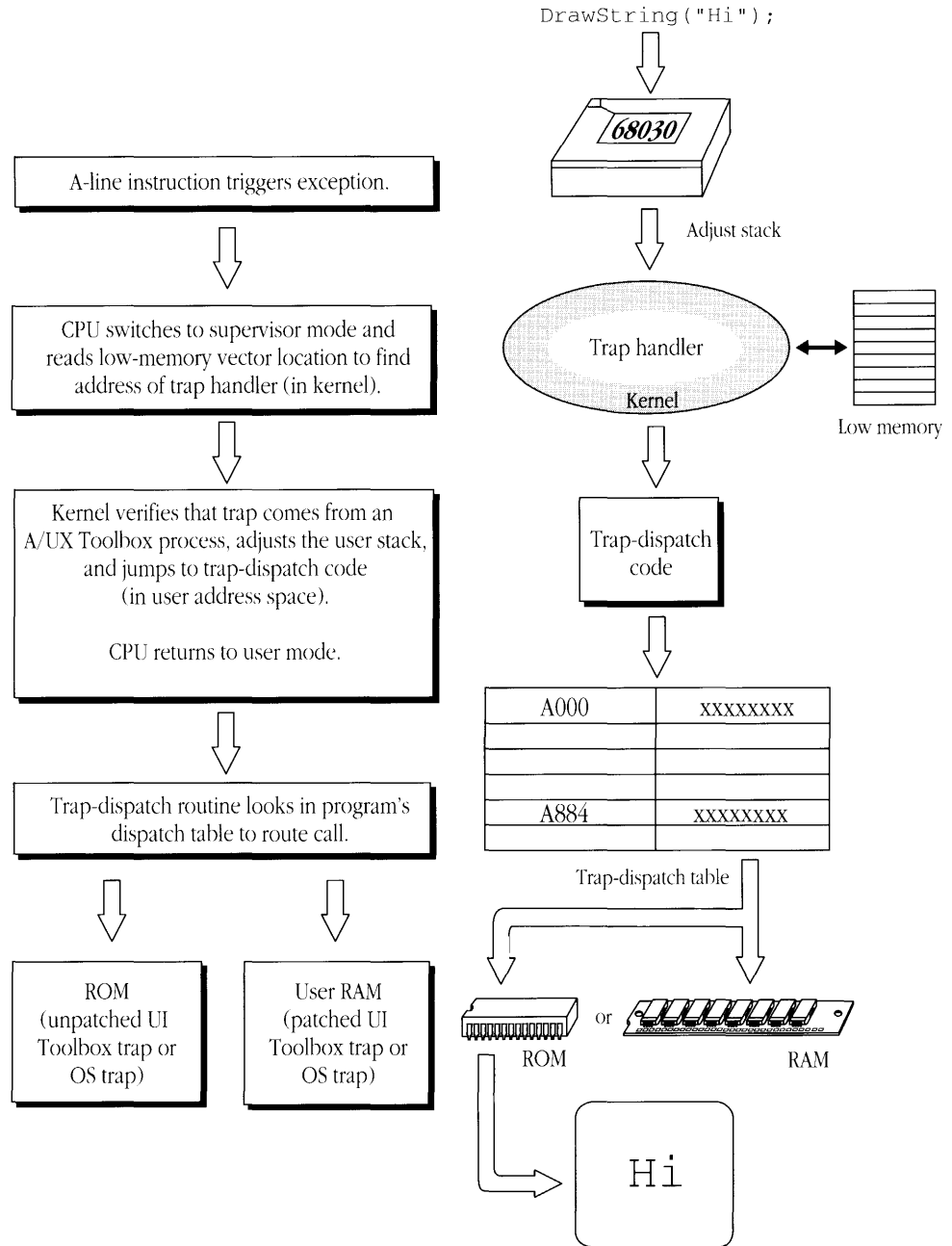


Figure C-1 A-line trap handling in A/UX

“Not in ROM” routines

The A/UX Toolbox also supports the “not in ROM” calls described in the *Inside Macintosh* volumes. (“Not in ROM” is explained at the end of the Preface in each volume of *Inside Macintosh*.) There are two versions of code for these glue routines, both in `/usr/lib`. The nonshared archive is `libmac.a`, and the shared version is `libmac_s.a`. The two are functionally equivalent. You can use either version by naming it on the command line for compiling or link editing, as with any archive file. The shared version saves some space in applications that use it and has the advantage of always providing the most current routines to applications that call on it. Shared libraries are discussed in *A/UX Programming Languages and Tools*, Volume 1. An A/UX Toolbox application compiled and linked according to the instructions in Chapter 2, “Using the A/UX Toolbox,” will access one of these archives. Applications compiled in the Macintosh environment must link to the appropriate libraries to use these calls.

Macintosh global variables

The standard Macintosh environment includes a set of global variables used by different parts of the system and stored in low memory. (These global variables are described in *Inside Macintosh*, Volumes III, IV, V, and VI.) To make room for these global variables, an A/UX Toolbox application compiled under A/UX is linked at virtual memory address `0x10000000`. The `launch(1)` program for executing Macintosh applications from the shell, itself an A/UX Toolbox application, is linked at this address.

Not all of the global variables listed in *Inside Macintosh* are supported by A/UX. In general, variables related to hardware are not supported. Appendix D, “Low-Memory Global Variables,” lists the supported Macintosh global variables.

File type and creator

A set of file information called the *Finder information*, which includes a file’s type and creator, is stored in a special entry in both AppleSingle-format and AppleDouble-format files in A/UX.

The Macintosh Standard File Package, which is supported by A/UX, uses a file's type and creator to filter the documents presented when the user opens a file from within an application. When an A/UX file goes into the Macintosh OS environment, if no creator is found, then 'A/UX' is assigned as creator. File types are assigned, if feasible. Files known to be text files receive the 'TEXT' type; known shell scripts receive 'SHEL'. Chapter 6, "File Systems and File Formats," provides general information on file handling across the boundary between the two environments.

Converting between C and Pascal conventions

Most of the Macintosh ROM routines use Pascal language conventions, which differ from the conventions used by the A/UX C compilers.

The C and Pascal conventions differ in six primary ways: how strings are stored, how a parameter list is evaluated, how the parameter types are stored, how QuickDraw point small structures are passed, how function results are returned, and how registers are used. This section describes the differences.

When necessary, the A/UX Toolbox interface routines convert C program calls to a form usable by the ROM and then convert the ROM's output to a form usable by the C program. The A/UX Toolbox routines that perform this conversion have three parts: the entry conversion code, the A-line trap, and the exit conversion code.

The libraries in Release 3.0 of the A/UX Toolbox include two versions of all routines that take strings or small structures (such as QuickDraw point values), or that return strings. One version, spelled as the routine appears in *Inside Macintosh*, uses Pascal string format and point-passing conventions. The second version, spelled in all lowercase letters, uses C string format and point-passing conventions. The lowercase version converts input parameters from C format to Pascal format before passing them to the ROM and converts return values back to C format. Both versions use interface routines to adjust for other differences in parameter-passing and return-value conventions.

If you are writing procedures that will be called from the ROM code, you must write assembly-language code to rework the parameters when your procedure is called.

Storing strings

In C, a string is normally stored as an array of characters, of any length, terminated by the null byte (`\0`). In Pascal, a string starts with a byte that specifies the length of the string, followed by a maximum of 255 characters. Because the length is specified explicitly, a Pascal string is not terminated by a null byte.

Because both conventions contain an extra byte of information (the null byte at the end of a C string and the count at the beginning of a Pascal string), it is possible to transform a string in place between the two formats. The A/UX Toolbox includes the routines `c2pstr` and `p2cstr` to perform these conversions. (See “String Conversion Between Pascal and C” in Appendix F.)

The lowercase versions of all ROM routines that take or return strings perform these conversions automatically. Use the lowercase version when you are passing a string directly to a routine. The mixed-case versions perform no conversion. Use the mixed-case version when you are using a string that is a field of a structure maintained by a ROM routine.

The routine and parameter descriptions in Appendix F, “C Interface Library,” follow these conventions:

- A pointer to a `char` data type (printed `char *`) represents a pointer to a C-format string.
- A parameter of type `str255` represents a Pascal-format string.

Ordering and storing parameters

Parameters in Pascal functions are evaluated from left to right and are pushed onto the stack in the order in which they are evaluated. For example, with the function `foo(a, b)`, `a` is pushed first, and then `b`.

Parameters in C functions are evaluated from right to left (by the Macintosh C compilers) and are pushed onto the stack in the order in which they are evaluated. With the function `foo(a, b)`, `b` is pushed first, and then `a`.

When necessary, the A/UX Toolbox routines reorder the parameters passed to a function before calling the ROM.

Characters and enumerated types whose literal values fall in the range of type `char` or `unsigned char` are pushed as bytes. (Pushing a byte on the stack requires a 16-bit word on the stack. The value is in the high-order 8 bits; the low-order 8 bits are unused.) `short` values and enumerated types whose literal values fall in the range of type `short` or `unsigned short` are passed as 16-bit values. `int` and `long` values and the remaining enumerated types are passed as 32-bit values. Pointers and arrays are passed as 32-bit addresses. SANE types `float`, `double`, `comp`, and `extended` are passed as extended 80-bit values.

Structures are also passed by value on the stack. Their size is rounded up to a multiple of 16 bits (2 bytes). If rounding occurs, the unused storage has the highest memory address. The function being called removes the parameters from the stack.

Passing small structures

The Pascal language always passes small structures (less than or equal to four bytes), such as QuickDraw point values, by value rather than by pointer, unless the structure is declared as a VAR. (This convention is a general rule for Pascal, which passes by value unless VAR is declared.) A/UX library calls with mixed-case names follow the Pascal convention.

The calls with lowercase names that pass small structures put the address of the structure on the stack.

Returning function results

A/UX C functions return pointer values in registers A0 and D0 and nonpointer values in register D0. MPW C functions return all values in D0.

A Pascal function places its result on the stack. The caller reserves stack space for the function result before pushing any parameters. Characters and enumerated types whose literal values fall in the range of type `char` or `unsigned char` are returned as bytes. (These values returned as bytes require a 16-bit word on the stack. The value is in the high-order 8 bits; the low-order 8 bits are unused.) All `short` values and enumerated types whose literal values fall in the range of type `short` or `unsigned short` are returned as 16-bit values. All `int` and `long` values and the remaining

enumerated types are returned as 32-bit values. Pointers are returned as 32-bit addresses. Arrays cannot be returned as function results. Results of type `float` are returned as 32-bit values. For types `double`, `comp`, and `extended`, the caller pushes the address for a `double`, `comp`, or `extended` result, respectively, in the function-result location on the stack. The procedure being called stores the result at this address. The caller removes the function results from the stack.

For structure results, if the Pascal function returns a structure of more than 4 bytes, the caller pushes a pointer to a result space before pushing any parameters. If the structure is 4 bytes or fewer, the caller reserves 4 or 2 bytes on the stack for it.

The A/UX Toolbox routines move the results returned by a Pascal-like ROM call to the location appropriate for a C call.

Register conventions

Pascal treats registers D0, D1, D2, A0, and A1 as scratch registers. All other registers are preserved. Register A5 is the global frame pointer, register A6 is the local frame pointer, and register A7 is the stack pointer. A/UX C treats only registers D0, D1, A0, and A1 as scratch registers. A6 is the frame pointer, A7 the stack pointer.

An A/UX Toolbox routine automatically saves and restores register D2 when using ROM code.

Appendix D: Low-Memory Global Variables

This appendix lists the low-memory global variables that are supported in the A/UX Toolbox. For the function and memory location of each variable, see the appendixes titled “Global Variables” in *Inside Macintosh*, Volumes III, IV, V, and VI.

Generally, your software will have maximum portability if you don't rely on the low-memory global variables, but instead use available routines that return the desired information. For example, the `TickCount` function returns the same value that is contained in the low-memory global variable `Ticks`.

The low-memory global variables are listed by the name used in the C include file `sysequ.h`, available in `/lib/include/mac`. The name of each variable is followed by its low-memory address, which is provided only for identification and reference to Macintosh documentation and should never be used as an address.

The general list is followed by three brief lists of associated global variables for the Window Manager, TextEdit, and the Resource Manager.

Table D-1 General global variables

Name	Reference	Description
ABusVars	0x2D8	Pointer to AppleTalk local variables
AppLLimit	0x130	Application limit [pointer]
AppLZone	0x2AA	Application heap zone [pointer]
BootDrive	0x210	Drive number of boot drive [word]
BufPtr	0x10C	Top of application memory [pointer]
BufTgDate	0x304	Time stamp [word]
BufTgFBkNum	0x302	Logical block number [word]
BufTgFFlg	0x300	Flags [word]
BufTgFNum	0x2FC	File number [long]
CaretTime	0x2F4	Caret blink ticks [long]
CPUFlag	0x12F	\$00=68000, \$01=68010, \$02=68020 (old ROM inits to \$00)
CurApName	0x910	Name of application [STRING[31]]
CurApRefNum	0x900	refNum of application's resFile [word]
CurDirStore	0x398	Save directory across calls to Standard File [long]
CurJTOffset	0x934	Current jump table offset [word]
CurPageOption	0x936	Current page 2 configuration [word]
CurPitch	0x280	Current pitch value [word]
CurrentA5	0x904	Current value of A5 [pointer]
CurStackBase	0x908	Current stack base [pointer]
DefltStack	0x322	Default size of stack [long]
DeviceList	0x8A8	List of display devices [long]
DoubleTime	0x2F0	Double-click ticks [long]
DrvQHdr	0x308	Queue header of drives in system [10 bytes]
DSAlertRect	0x3F8	Rectangle for disk-switch alert [8 bytes]
DSAlertTab	0x2BA	System error alerts [pointer]
DSErrCode	0xAF0	Last system error alert ID
DTQueue	0x0D92	Deferred task queue header [10 bytes]
EventQueue	0x14A	Event queue header [10 bytes]
ExtStsDT	0x2BE	SCC external status interrupt vector table [16 bytes]

(continued) ➡

Table D-1 General global variables (*continued*)

Name	Reference	Description
GZRootHnd	0x328	Root handle for GrowZone [handle]
HeapEnd	0x114	End of heap [pointer]
HiliteMode	0x938	Used for color highlighting
HiliteRGB	0x0DA0	6 bytes: RGB of highlight color
IntlSpec	0xBA0	International software installed if not equal to -1 [long]
JDTInstall	0x0D9C	Pointer to deferred task install routine [long]
JFetch	0x8F4	Fetch a byte routine for drivers [pointer]
IODone	0x8FC	IODone entry location [pointer]
JournalRef	0x8E8	Journaling driver's refNum [word]
JStash	0x8F8	Stash a byte routine for drivers [pointer]
JVBLTask	0x0D28	Vector to slot VBL task interrupt handler
KbdLast	0x218	Same as KbdVars + 2
KbdType	0x21E	Keyboard model number [byte]
KeyRepThresh	0x190	Key repeat speed [word]
KeyThresh	0x18E	Threshold for key repeat [word]
Lo3Bytes	0x31A	Constant \$00FFFFFF [long]
Lvl2DT	0x1B2	Interrupt level-2 dispatch table [32 bytes]
MainDevice	0x8A4	The main screen device [long]
MemErr	0x220	Last Memory Manager error [word]
MemTop	0x108	Top of memory [pointer]
MinStack	0x31E	Minimum stack size used in InitApplZone [long]
MinusOne	0xA06	Constant \$FFFFFFFF [long]
MMU32bit	0x0CB2	Boolean value reflecting current machine MMU mode [byte]
OneOne	0xA02	Constant \$00010001 [long]
PortBUse	0x291	Port B use, same format as PortAUse
QDColors	0x8B0	Handle to default colors [long]
RAMBase	0x2B2	RAM base address [pointer]

(continued) ➡

Table D-1 General global variables (*continued*)

Name	Reference	Description
ResumeProc	0xA8C	Address of resume procedure from InitDialogs [pointer]
RndSeed	0x156	Random seed/number [long]
ROM85	0x28E	Actually high bit — 0 for ROM version \$75 (sic) and later [word]
ROMBase	0x2AE	ROM base address [pointer]
SCCRd	0x1D8	SCC base read address [pointer]
SCCWrt	0x1DC	SCC base write address [pointer]
ScrapCount	0x968	Validation byte [word]
ScrapHandle	0x964	Memory scrap [handle]
ScrapName	0x96C	Pointer to scrap name [pointer]
ScrapSize	0x960	Scrap length [long]
ScrapState	0x96A	Scrap state [word]
Scratch8	0x9FA	Scratch [8 bytes]
Scratch20	0x1E4	Scratch [20 bytes]
ScrDmpEnb	0x2F8	Screen dump enabled? [byte]
ScrHRes	0x104	Screen horizontal dots/inch [word]
ScrnBase	0x824	Screen base [pointer]
ScrVRes	0x102	Screen vertical dots/inch [word]
SdVolume	0x260	Global volume (sound) control [byte]
SEvtEnb	0x15C	Enable <code>SysEvent</code> calls from GNE [byte]
SFSaveDisk	0x214	Last <code>vRefNum</code> seen by standard file [word]
SoundBase	0x266	Base address for sound buffer [pointer]
SoundLevel	0x27F	Current level in buffer [byte]
SoundPtr	0x262	4VE sound definition table [pointer]
SPAlarm	0x200	Alarm time [long]
SPATalkA	0x1F9	AppleTalk node number hint for port A
SPATalkB	0x1FA	AppleTalk node number hint for port B
SPClikCaret	0x209	Double click/caret time in 4/60ths [4 bits]
SPConfig	0x1FB	Serial port config bits: 4–7 port A, 0–3 port B

(continued) ➔

Table D-1 General global variables (*continued*)

Name	Reference	Description
SPFont	0x204	Default application font number minus 1 [word]
SPKbd	0x206	Keyboard repeat threshold in 4/60ths [4 bits] (ignored under A/UX)
SPMisc2	0x20B	Miscellaneous [1 byte]
SPPortA	0x1FC	SCC port A configuration [word]
SPPortB	0x1FE	SCC port B configuration [word]
SPPrint	0x207	Print stuff [byte]
SPValid	0x1F8	Validation field (\$A7) [byte]
SPVolCtl	0x208	Volume control [byte]
SysEvtMask	0x144	System event mask [word]
SysParam	0x1F8	System parameter memory [20 bytes]
SysZone	0x2A6	System heap zone [pointer]
TheGDevice	0x0CC8	The current graphics device [long]
TheZone	0x118	Current heap zone [pointer]
Ticks	0x16A	Tick-count, time since boot [long]
Time	0x20C	Clock time (extrapolated) [long]
TimeDBRA	0x0D00	Number of iterations of DBRA per millisecond [word]
TimeSCCDB	0x0D02	Number of iterations of SCC access and DBRA [word]
TimeSCSIDB	0x0DA6	Number of iterations of SCSI access and DBRA [word]
UTableBase	0x11C	Unit I/O table [pointer]
VBLQueue	0x160	VBL queue header [10 bytes]
VIA	0x1D4	VIA base address [pointer]

Table D-2 Window Manager globals

Name	Reference	Description
CurActivate	0xA64	Window slated for activate event [pointer]
CurDeactive	0xA68	Window slated for deactivate event [pointer]
DeskHook	0xA6C	Hook for painting the desk [pointer]
DeskPattern	0xA3C	Desk pattern [8 bytes]
DragHook	0x9F6	User hook during dragging [pointer]
GhostWindow	0xA84	Window hidden from FrontWindow [pointer]
GrayRgn	0x9EE	Rounded gray desk region [handle]
PaintWhite	0x9DC	Erase newly drawn windows? [word]
WindowList	0x9D6	Z-ordered linked list of windows [pointer]
WMgrPort	0x9DE	Window Manager's grafport [pointer]

Table D-3 TextEdit globals

Name	Reference	Description
TEDoText	0xA70	TextEdit doText procedure hook [pointer]
TERecal	0xA74	TextEdit recalText procedure hook [pointer]
TEScrpHandle	0xAB4	TextEdit scrap [handle]
TEScrpLength	0xAB0	TextEdit scrap length [word]
TESysJust	0xBAC	System justification (international TextEdit) [word]
WordRedraw	0xBA5	Used by TextEdit RecalDraw [byte]

Table D-4 Resource Manager globals

Name	Reference	Description
CurMap	0xA5A	Reference number of current map [word]
ResErr	0xA60	Resource error code [word]
ResErrProc	0xAF2	Resource error procedure [pointer]
ResLoad	0xA5E	Auto-load feature [word]
RomMapInsert	0xB9E	Determines if we should link in map [byte]
SysMap	0xA58	Reference number of system map [word]
SysMapHndl	0xA54	System map [handle]
SysResName	0xAD8	Name of system resource file [STRING[19]]
TmpResLoad	0xB9F	Second byte is temporary ResLoad value
TopMapHndl	0xA50	Topmost map in list [handle]



Appendix E: Resource Compiler and Decompiler

About the resource compiler and decompiler / E-2

Resource description statements / E-7

Preprocessor directives / E-39

Resource description syntax / E-43

This appendix explains how to build resources with the resource compiler, `rez`, and how to use the resource decompiler, `derez`. See *Inside Macintosh*, Volume I, for a description of resources.

About the resource compiler and decompiler

The resource compiler, `rez`, compiles one or more text files, called *resource description files*, and produces a resource file. The resource decompiler, `derez`, decompiles a resource file, producing a new resource description file that can be understood by `rez`. Figure E-1 illustrates the complementary relationship between `rez` and `derez`.

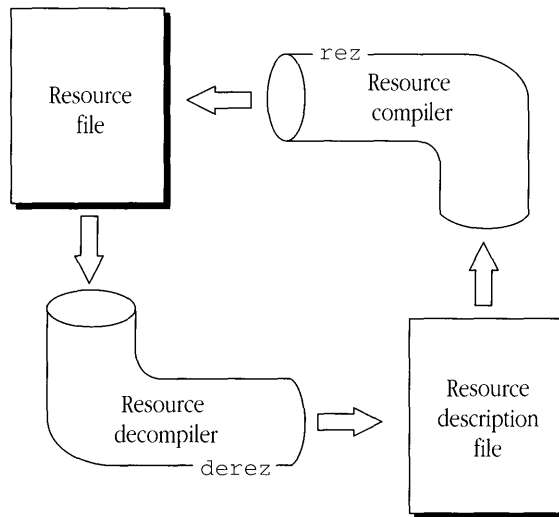


Figure E-1 `rez` and `derez`

In A/UX, `rez` always creates an AppleDouble header file. `derez` always creates a standard A/UX text file.

`rez` can combine resources or resource descriptions from a number of files into a single resource file. `rez` can also delete resources or change resource attributes. `rez` supports preprocessor directives that allow you to substitute macros, include other files, and use if-then-else constructs. (These directives are described in “Preprocessor Directives,” later in this appendix.)

`derez` creates a text representation of a resource file based on resource type declarations identical to those used by `rez`. (If you don't specify any type declarations, the output of `derez` is in the form of raw data statements.) The output of `derez` is a resource description file that can be used as input to `rez`. You can edit this file to add comments, translate resource data into a foreign language, or specify conditional resource compilation by using the if-then-else structures of the preprocessor. You can also use the A/UX `diff(1)` command to compare resource description files.

Standard type declaration files

Four text files—`types.r`, `systypes.r`, `scripttypes.r`, and `pict.r`—contain resource declarations for standard resource types. These files are located in the directory `/mac/lib/rincludes`. They contain definitions for the following types:

<code>types.r</code>	type declarations for the most common Macintosh resource types ('ALRT', 'DITL', 'MENU', and so on)
<code>systypes.r</code>	type declarations for 'DRVr', 'FOND', 'FONT', 'FWID', 'INTL', and 'NFMT' resources and many others
<code>scripttypes.r</code>	type declarations for resource descriptions specific to the Script Manager
<code>pict.r</code>	type declarations for debugging 'PICT' resources

Using rez and derez

rez and derez are primarily used to create and modify resource files. Figure E-2 illustrates the process of creating a resource file.

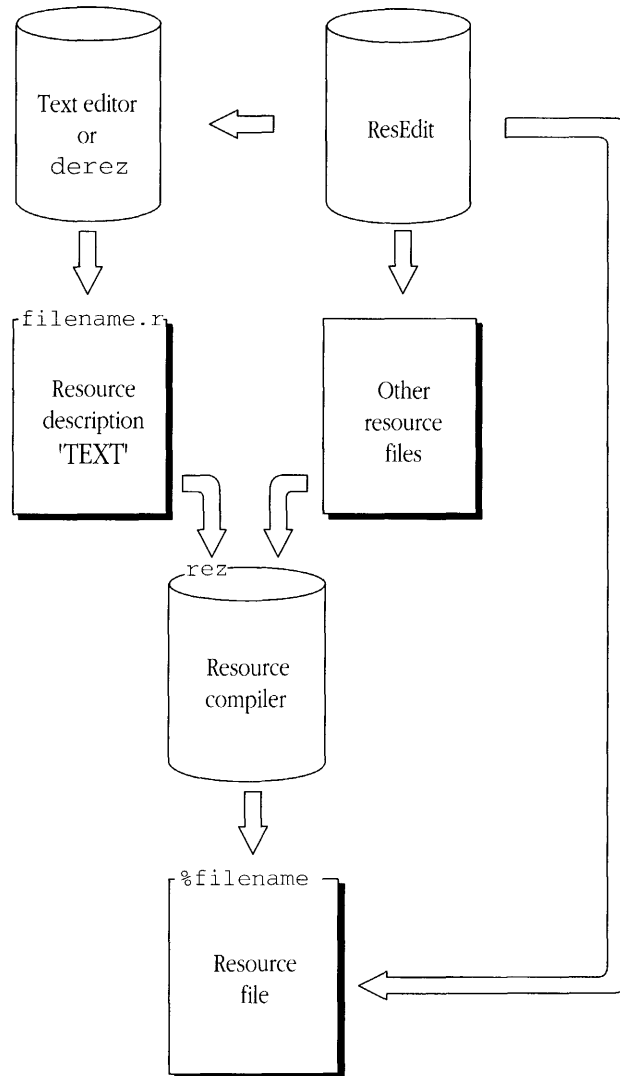


Figure E-2 Creating a resource file

Structure of a resource description file

The resource description file consists of resource type declarations (which can be included from another file) followed by resource data for the declared types. The resource compiler and resource decompiler have no built-in resource types. You must either define your own types or include the appropriate type declaration (`.r`) files.

A resource description file contains any number of the seven resource statements:

<code>change</code>	Changes the type, ID, name, or attributes of existing resources.
<code>data</code>	Specifies raw data.
<code>delete</code>	Deletes existing resources.
<code>include</code>	Includes resources from another file.
<code>read</code>	Reads data file and includes it as a resource.
<code>resource</code>	Specifies data for a resource type declared in a previous <code>type</code> statement.
<code>type</code>	Declares resource type descriptions for subsequent <code>resource</code> statements.

The section “Resource Description Statements,” later in this appendix, describes each of these statements.

A type declaration provides the pattern for any associated resource data specifications by indicating data types, alignment, size and placement of strings, and so on. You can intersperse type declarations and data in the resource description file as long as the declaration for a given resource precedes any `resource` statements that refer to it. An error is returned if data (that is, a `resource` statement) is given for a type that has not been previously defined. Whether a type was declared in a resource description file or in an include file, you can redeclare it by providing a new declaration later in a resource description file.

A resource description file can also include comments and preprocessor directives:

- Comments can be included anywhere that white space is allowed in a resource description file, within the comment delimiters `/*` and `*/`. Comments do not nest. For example, this is *one* comment:

```
/* Hello /* there */
```

`rez` also supports C++ style comments:

```
type 'tost' { // the rest of this line is ignored
```

- Preprocessor directives substitute macro definitions and include files and provide if-then-else processing before other `rez` processing takes place. The syntax of the preprocessor is similar to that of the C-language preprocessor. For details, see “Preprocessor Directives,” later in this appendix.

Sample resource description file

An easy way to learn about the resource description format is to decompile some existing resources. For example, the following command decompiles only the `'WIND'` resources in the `sample` application, according to the type declaration in `types.r`, in the `/mac/lib/r/includes` directory:

```
derez sample -only WIND types.r > derez.out
```

After this command is run, `derez.out` contains this text:

```
resource    'WIND' (128, "Sample Window") {
            {64, 60, 314, 460},
            documentProc,
            visible,
            noGoAway,
            0x0,
            "Sample Window"
};
```


Note that this statement is identical to the resource description in the file `sample.r`, which was originally used to build the resource. This resource data corresponds to the following type declaration, contained in `types.r`:

```
type 'WIND' {
    rect;      /* boundsRect */
    integer    documentProc, dBoxProc, plainDBox, /* procID */
              altDBoxProc, noGrowDocProc,
              zoomProc=8, rDocProc=16;
    byte      invisible, visible;                /* visible */
    fill byte;
    byte      noGoAway, goAway;                  /* goAway */
    fill byte;
    unsigned hex longint;                         /* refCon */
    pstring   Untitled = "Untitled";             /* title */
};
```

type and resource statements are explained in detail in the next section, “Resource Description Statements.”

Resource description statements

This section describes the syntax and use of the seven resource description statements: `change`, `data`, `delete`, `include`, `read`, `resource`, and `type`.

Syntax notation

The syntax notation in this appendix follows the conventions given in the Preface, with these additions:

- Words that are part of the resource description language are shown in *Courier* to distinguish them from other text. `rez` is not sensitive to the case of these words.
- Punctuation characters such as commas (,), semicolons (;), and quotation marks (‘ and ’) are to be written as shown.

- If one of the syntax notation characters (for example, [or]) must be written as a literal, it is shown enclosed by “curly” single quotation marks, like this:

```
bitstring ' [ length ]'
```

In this case, the brackets are typed literally. The brackets do *not* mean that the enclosed element is optional.
- Spaces between syntax elements, constants, and punctuation are optional. They are used only to make reading code easier.
- Hexadecimal numbers are flagged with a leading dollar sign. Tokens in resource description statements can be separated by spaces, tabs, newlines, or comments. Note that braces ({ and }) are to be written as shown.

Special terms

The following terms represent a minimal subset of the nonterminal symbols used to describe the syntax of commands in the resource description language.

Term	Definition
<i>resource-type</i>	<i>long-expression</i>
<i>resource-name</i>	<i>string</i>
<i>resource-ID</i>	<i>word-expression</i>
<i>ID-range</i>	<i>ID</i> [: <i>ID</i>]

◆ **Note** The placeholder *expression* is defined in “Expressions,” later in this appendix. ◆

For more information on syntax, see “Resource Description Syntax,” later in this appendix.

change—change a resource's vital information

The `change` statement changes a resource's vital information. Vital information includes the resource type, ID, name, attributes, or any combination of these.

Syntax `change resource-type1 ['(resource-name1 | ID1[:ID2]')]`
 `to resource-type2 '(ID[, resource-name2] [, attributes...)' ;`

Description Changes the resource of type *resource-type1* in the output file with the specified identifier *resource-name1*, *ID*, or range of ID numbers to a resource of type *resource-type2* with the specified ID. You can optionally specify *resource-name2* and *attributes* for the new resource. If neither *resource-name2* nor the attributes are specified, the name and attributes are not changed.

For example, here is a shell command (`echo`) that calls on `rez` to set the protected bit `On` for all `'CODE'` resources in the file `TestDA`:

```
echo "change 'CODE' to $$type ($$Id,$$Attributes | 8);" @
      | rez -a -o TestDA
```

The continuation character (`@`, obtained by pressing `OPTION-D`) at the end of the first line of this example has the effect of continuing the command onto the next line. The continuation character is used to escape the character that follows from performing its usual action. In this case, the subsequent character is a newline, and the line-termination function is escaped.

◆ **Note** The `change` statement is valid only when the `-a` (`append`) option is specified in the command line. It makes no sense to change resources when you're creating a new resource file from scratch. ◆

data—specify raw data

`data` statements specify raw data as a sequence of bits, without any formatting.

Syntax

```
data 'resource-type' (' ID[, "resource-name"][, attributes..]') {  
    "data-string"  
};
```

Description

Reads the data found in the string *data-string* and writes it as a resource with the type *resource-type* and the resource ID *ID*. You can optionally specify a resource name, resource attributes, or both.

For example, the following statement reads the data string shown and writes it as a 'PICT' resource with resource ID 128:

```
data 'PICT' (128) {  
    $"4F35FF8790000000"  
    $"FF234F35FF790000"  
};
```

◆ **Note** When `derez` generates a resource description, it uses the `data` statement to represent any resource type that doesn't have a corresponding type declaration or that cannot be disassembled for some other reason. ◆

delete—delete a resource

The `delete` statement deletes a resource. This statement can be useful, for example, in the process of translating menu and dialog box text in system disks or applications intended for use in non-English-speaking countries. The `delete` statement and the `change` statement (described earlier in this appendix) allow you to delete and change resources without switching to ResEdit.

Syntax `delete 'resource-type' [('resource-name' | ID1[:ID2]) '];`

Description Deletes the resource of type *resource-type* from the output file with the specified identifier *resource-name*, *ID*, or range of ID numbers. If both the resource name and the ID are omitted, all resources of type *resource-type* are deleted.

◆ **Note** The `delete` statement is valid only when the `-a` (append) option is specified in the command line. It makes no sense to delete resources when you're creating a new resource file from scratch. ◆

You can delete resources that have their protected bit set only if you use the `-ov` option.

Here is an example of a shell command (`echo`) that calls on `rez` to delete all resources of type `'ckid'` from the file `SomeTextFile`:

```
echo "delete 'ckid';" | rez -a -o SomeTextFile
```

include—include resources from another file

The `include` statement reads resources from an existing file and includes all or some of them.

Syntax `include "filename" [' resource-type' [' (' "resource-name" | ID1[:ID2] ')']];`

Description Reads the resource of type *resource-type* with the specified resource name, ID number, or range of ID numbers in the file *filename*. If both the resource name and the resource ID are omitted, `include` reads all resources of the type *resource-type* in the file *filename*. If *resource-type* is omitted, `include` reads all the resources in the file *filename*. These three possibilities are illustrated in the following examples:

```
include "otherfile" 'CODE' (128); /* read only CODE resource 128 */
include "otherfile" 'CODE';      /* read only the CODE resources */
include "otherfile";            /* read all resources from the file */
```

```
include "filename" not 'resource-type';
```

Read all resources not of the type *resource-type* in the file *filename*.

```
include "filename" 'resource-type1' as 'resource-type2';
```

Read all resources of type *resource-type1* and include them as resources of type *resource-type2*.

```
include "filename" 'resource-type1' (' "resource-name1" | ID1[:ID2] ')
                                   as 'resource-type2' (' ID[, "resource-name2" ][, attribute...]
');
```

Read the resource of type *resource-type1* with the specified resource name, ID number, or range of ID numbers in the file *filename*, and include it as a resource of type *resource-type2* with the specified ID. You can optionally specify *resource-name2* and new resource *attribute*. Resource attributes are defined in “Resource Attributes,” later in this section.

The following string variables can be used in the `include as resource` description statement to modify the resource information:

<code>\$\$Type</code>	type of resource from include file
<code>\$\$ID</code>	ID of resource from include file
<code>\$\$Name</code>	name of resource from include file
<code>\$\$Attributes</code>	attributes of resource from include file

For example, to include all 'DRVR' resources from one file and keep the same information, but also set the `SYSHEAP` attribute, you would use a statement like this:

```
include "file" 'DRVR' (0:40) as
    'DRVR' ($$ID, $$Name, $$Attributes | 64);
```

The `$$Type`, `$$ID`, `$$Name`, and `$$Attributes` variables are also set and legal within a normal `resource` statement. At any other time the values of these variables are undefined.

*Resource
attributes*

You can specify attributes by using a numeric expression (as described in the Resource Manager chapters of *Inside Macintosh*, Volumes I, IV, and V), or you can set them individually by specifying one of the keywords from any of the following pairs.

<i>Default</i>	<i>Alternative</i>	<i>Meaning</i>
<code>appheap</code>	<code>sysheap</code>	Specifies whether the resource is to be loaded into the application heap (<code>appheap</code>) or the system heap (<code>sysheap</code>). This attribute is meaningless if the resource is used only in A/UX.
<code>nonpurgeable</code>	<code>purgeable</code>	Specifies whether purgeable resources can be automatically purged by the Memory Manager (<code>purgeable</code>), or not (<code>nonpurgeable</code>).
<code>unlocked</code>	<code>locked</code>	Specifies whether locked resources can be moved by the Memory Manager (<code>unlocked</code>), or not (<code>locked</code>).
<code>unprotected</code>	<code>protected</code>	Specifies whether protected resources can be modified by the Resource Manager (<code>unprotected</code>), or not (<code>protected</code>).

<code>nonpreload</code>	<code>preload</code>	Specifies whether preloaded resources are placed in the heap as soon as the Resource Manager opens the resource file (<code>nonpreload</code>), or not (<code>preload</code>).
<code>unchanged</code>	<code>changed</code>	Tells the Resource Manager whether a resource has been changed (<code>changed</code>) or not (<code>unchanged</code>). <code>rez</code> does not allow you to set this bit, but <code>derez</code> displays it if it is set.

Bits 0 and 7 of the resource attributes are reserved for use by the Resource Manager and cannot be set by `rez`, but are displayed by `derez`.

You can list more than one attribute by separating the keywords with a comma (,). An example of attribute use is given in the next section, “`read`—Read Data as a Resource.”

read—read data as a resource

The `read` statement reads a data file or the data entry in a file as a resource.

Syntax `read 'resource-type' '(' ID[, "resource-name"][, attributes...]'`
 `"filename" ;`

Description Reads the file *filename* and writes it as a resource with the type *resource-type* and the resource ID *ID*, with the optional resource name *resource-name* and optional resource attributes *attributes* (as defined in the section “`include—Include Resources From Another File`”). For example, the statement

```
read 'STR ' (-789, "Test String", sysheap, preload) "Test8";
```

reads `Test8` and writes it as a `'STR '` resource with the resource ID `-789`, the resource name `Test String`, and the resource attributes `sysheap` and `preload`.

resource—specify resource data

The `resource` statement specifies an actual resource, based on previous `type` declarations.

Syntax

```
resource 'resource-type' (' ID[, resource-name][, attributes]') {  
    [data-statement1[, data-statement2]...]  
};
```

Description

Specifies the data for a resource of type *resource-type* and ID *ID*. The latest type declaration declared for *resource-type* is used to parse the data specification. The *data-statement* parameter specifies the actual data; the *data-statement* term appropriate to each resource type is defined in “Data Statements,” later in this section.

The `resource` definition causes an actual resource to be generated. A `resource` statement can appear anywhere in the resource description file, or in a separate file specified on the command line, or as an include file, as long as it comes after the relevant `type` declaration.

Data statements

The body of the data specification contains one data statement for each declaration in the corresponding type declaration. The base type must match the declaration.

<i>Base type</i>	<i>Instance types</i>
<code>string</code>	<code>string, cstring, pstring, wstring char</code>
<code>bitstring</code>	<code>boolean, byte, integer, longint, bitstring</code>
<code>rect</code>	<code>rect</code>
<code>point</code>	<code>point</code>

Switch data Switch data statements are specified in the following format:

switch-name case-body

For example, the following statement could be specified for the `'DITL'` type declaration example given in “Switch Type” in the description of the `type` declaration, later in this appendix. The *switch-name* example is `CheckBox`.

```
...  
CheckBox { enabled, "Check here" },  
...
```

The `boolean` and `pstring` values defined in the *case-body* section of the `CheckBox` case are set to `enabled` and to `"Check here"`; the key `bitstring` term was already set to a constant in the definition. Now data items are provided for all terms of the *case-body* section.

Array data Array data statements have the following format:

```
{[array-element[, array-element]...]}
```

An *array-element* parameter consists of any number of data statements, separated by commas.

For example, the following data might be given for the `'STR#'` type declaration example in “Array Type” in the description of the `type` declaration, later in this appendix.

```
resource 'STR#' (280) {
    {
        "this",
        "is",
        "a",
        "test"
    }
};
```

Sample resource definition The example given here describes a sample resource description file for a window. (See the Window Manager chapter in *Inside Macintosh*, Volume I, for information about resources for windows.)

Here is the `type` declaration given later in this appendix in “Sample `type` Statement”:

```
type 'WIND' {
    rect;                                /* boundsRect */
    integer    documentProc, dBoxProc, plainDBox, /* procID */
              altDBoxProc, noGrowDocProc,
              zoomProc=8, rDocProc=16;
    byte      invisible, visible;        /* visible */
    fill byte;
```

(continued) ➡

```

byte          noGoAway, goAway;          /* has close box*/
fill byte;

unsigned hex longint;                    /* refCon */

pstring      Untitled = "Untitled";     /* title */
};

```

Here is a typical example of the window data corresponding to this declaration:

```

resource 'WIND' (128, "My window", appheap, preload)
{ /*status report window*/

    {                                     /*status report window*/
        {40,80,120,300},                 /*bounding rectangle*/
        documentProc,                   /*documentProc etc.*/
        Visible,                         /*Visible or Invisible*/
        goAway,                          /*GoAway or NoGoAway*/
        0,                                /*reference value RefCon*/
        "Status Report"                  /*title*/
    };
};

```

This data definition declares a resource of type 'WIND', using whatever type declaration was previously specified for 'WIND'. The resource ID is 128; the resource name is `My window`. Because the resource name is represented by the Resource Manager as a `pstring` string, it should not contain more than 255 characters. The resource name can contain any character, including the null character (\$00). The resource is placed in the application heap when loaded, and it is loaded when the resource file is opened.

The first statement in the window type declaration declares a bounding rectangle for the window and corresponds to

```
rect;
```

in the type declaration. The rectangle is described by two points: the upper-left corner and the lower-right corner. The coordinates for these two points of a rectangle are separated by commas:

```
{ top, left, bottom, right }
```

Thus, the following values correspond to the coordinates *top*, *left*, *bottom*, and *right*:

```
{ 40, 80, 120, 300 }
```

Symbolic names Symbolic names can be associated with particular values of a numeric type. A symbolic name is given for the data in the second, third, and fourth fields of the window declaration. For example:

```
integer      documentProc=0, dBoxProc=1, plainDBox=2,  
             altDBoxProc=3, noGrowDocProc=4,  
             zoomProc=8, rDocProc=16;      /*windowType*/
```

This statement specifies a signed 16-bit integer field with symbolic names associated with the values 0 through 4 and 16. The values 0 through 4 need not be indicated in this case; if no values are given, symbolic names are automatically given values starting at 0, as explained earlier.

The sample window declaration assigns the values `TRUE` (1) and `FALSE` (0) to two different byte variables. For clarity, the window's resource data uses the symbolic names

```
visible,
```

```
goAway,
```

instead of their equivalents

```
TRUE,
```

```
TRUE,
```

```
or
```

```
1,
```

```
1,
```

type—declare resource type

The `type` declaration provides a template that defines the structure of the resource data for a single resource type or for individual resources. If more than one `type` declaration is given for a resource type, the last one read before the data definition is the one that's used. Therefore, you can override declarations from include files or previous resource description files.

Syntax

```
type 'resource-type' ['( ID1:ID2 )'] {  
    type-specification...  
};
```

Description

Causes any subsequent resource statement for the type *resource-type* to use the declaration { *type-specification...* }. The optional *ID1: ID2* specification causes the declaration to apply only to a given resource ID or range of IDs. The first 12 type specifications in the following list are data types.

type-specification can be any one of these options:

```
bitstring[n]  
byte  
integer  
longint  
boolean  
char  
string  
pstring  
wstring  
cstring  
point  
rect  
fill           zero fill  
align         zero fill to nibble, byte, word, or long word boundary  
switch       control construct (case statement)  
array       array data specification—zero or more instances of data types
```

These types can be used singly or together in a `type` statement. Each of these type specifiers is described later in this section.

◆ **Note** Several of these types require additional fields. The exact syntax is given later in this section. ◆

You can also declare a resource type that uses another resource's `type` declaration, by using the following variant of the `type` statement:

```
type 'resource-type1' ['(' ID1[:ID2] ')'] as 'resource-type2' ['(' ID ')'];
```

Data-type specifications

A data-type statement declares a field of the given data type. It can also associate symbolic names or constant values with the data type. Data-type specifications can take three forms, as shown in this example:

```
type 'XAMP' { /* declare a resource of type 'XAMP' */
    byte;
    byte      off=0, on=1;
    byte = 2;
};
```

- The first `byte` statement declares a byte field; the actual data is supplied in a subsequent `resource` statement.
- The second `byte` statement is identical to the first, except that the two symbolic names `off` and `on` are associated with the values 0 and 1. These symbolic names could be used in the `resource` data.
- The third `byte` statement declares a byte field whose value is always 2. In this case, no corresponding statement appears in the `resource` data.

◆ **Note** Numeric expressions and strings can appear in `type` statements; they are defined in “Expressions,” later in this appendix. ◆

Numeric types The numeric types (`bitstring`, `byte`, `integer`, and `longint`) are fully specified as follows:

```
[unsigned][radix] numeric-type[ =expr | symbol-definition...];
```

Explanations of these fields follow. Information on the optional *expr* and *symbol-definition* fields is given with the explanations of various *numeric-type* designations.

- The `unsigned` prefix signals `derez` that the number should be displayed without a sign—that the high-order bit may be used for data and the value of the integer cannot be negative. The `unsigned` prefix is ignored by `rez` but is needed by `derez` to correctly represent a decompiled number. `rez` uses a sign if it is specified in the data. Precede a signed negative constant with a minus sign (-); `$FFFFFF85` and `-$7B` are equivalent in value.

- *radix* is one of the following string constants:

`hex` Data is supplied as hexadecimal.

`decimal` Data is supplied as decimal.

`octal` Data is supplied as octal.

`binary` Data is supplied as binary.

`literal` Data is taken as literal input.

- *numeric-type* is one of the following types:

`bitstring` '[' *length* ']' Bitstring of *length* bits (maximum 32).

`byte` Byte (8-bit) field. This type is the same as `bitstring[8]`.

`integer` Integer (16-bit) field. This type is the same as `bitstring[16]`.

`longint` Long integer (32-bit) field. This type is the same as `bitstring[32]`.

`rez` uses integer arithmetic and stores numeric values as integer numbers. `rez` translates `boolean`, `byte`, `integer`, and `longint` values to `bitstring` equivalents. All computations are done in 32 bits and truncated.

An error is generated if a value won't fit in the number of bits defined for the type. The `byte`, `integer`, and `longint` constants are valid in the ranges shown in the list that follows.

<i>Type</i>	<i>Maximum</i>	<i>Minimum</i>
byte	255	-128
integer	65535	-32768
longint	4294967295	-2147483648

Boolean type A Boolean type is a single bit with two possible states: 0 (or `FALSE`) and 1 (or `TRUE`). (`TRUE` and `FALSE` are global predefined identifiers.) Boolean values are declared as follows:

```
boolean [= constant | symbolic-value...];
```

For example, the following Boolean type declaration declares a value of `FALSE`:

```
type          'DONE' {
                boolean = false;
            };
```

Type `boolean` declares a 1-bit field equivalent to

```
unsigned bitstring[1]
```

Note that this type is not the same as a `Boolean` variable as defined by Pascal.

Character type Characters are declared as follows:

```
char [= string | symbolic-value...];
```

For example:

```
type 'SYMB' {
                char  dollar = "$", percent = "%";
            };

resource 'SYMB' (128) {
                dollar
            };
```

Type `char` declares an 8-bit field equivalent to

```
string[1]
```

String type String data types are specified as follows:

string-type [*length* ''] [= *string* | *symbol-value*...];

string-type is one of the following types:

[hex] *string*

Plain string (contains no length indicator or termination character). The optional `hex` prefix tells `derezz` to display the string as a hex string. The expression `string[n]` contains n characters and is n bytes long. Type `char` is shorthand for `string[1]`.

pstring

Pascal string. (A leading byte contains the length information.) The expression `pstring[n]` contains n characters and is $n + 1$ bytes long. `pstring` has a built-in maximum length of 255 characters, the highest value the length byte can hold. If the string is too long to fit the field, `rez` issues a warning and truncates the string.

wstring

Very large Pascal string. (Two leading bytes contain the length information.) A string of this type can contain up to 65,535 characters. The expression `wstring[n]` contains n characters and is $n + 2$ bytes long.

cstring

C string. (A trailing null byte marks the end of the string.) The expression `cstring[n]` contains $n - 1$ characters and is n bytes long. A C string of length 1 can be assigned only the value `" "`, because `cstring[1]` has room only for the terminating null.

Each string type can be followed by an optional *length* indicator in brackets (`[n]`). *length* is an expression indicating the string length in bytes. *length* is a positive number in the range $1 \leq \textit{length} \leq 2147483647$ for `string` and `cstring`, in the range $1 \leq \textit{length} \leq 255$ for `pstring`, and in the range $1 \leq \textit{length} \leq 65535$ for `wstring`.

◆ **Note** You cannot assign the value of literals to string data types. ◆

If no length indicator is given, `pstring`, `wstring`, or `cstring` stores the number of characters in the corresponding data definition. If a length indicator is given, the data can be truncated on the right or padded on the right. The padding characters for all string types are nulls. If the data contains more characters than the length indicator provides for, `rez` issues a warning message and truncates the string.

- ▲ **Warning** A null byte within a C string is a termination indicator and may confuse `derezh` and C programs. However, the full string, including the explicit null and any text that follows it, is stored by `rez` as input. ▲

Point and rectangle types Because points and rectangles appear so frequently in resource files, they have their own simplified syntax:

```
point [= point-constant | symbolic-value..];
```

```
rect [= rect-constant | symbolic-value..];
```

where

```
point-constant = { x-integer-expr, y-integer-expr }
```

and

```
rect-constant = { integer-expr, integer-expr, integer-expr, integer-expr }
```

These type statements use integer expressions to declare a point (two 16-bit signed integers) or a rectangle (four 16-bit signed integers). The integers in a rectangle definition specify the rectangle's upper-left and lower-right points, respectively.

Fill and align types

The resource created by a `resource` definition has no implicit alignment. It's treated as a bit stream, and integers and strings can start at any bit. The `fill` and `align` type specifiers allow you to pad fields so that they begin on a boundary that corresponds to the field type. `align` is automatic, and `fill` is explicit. Both `fill` and `align` generate zero-filled fields.

Fill specification The `fill` statement causes `rez` to add the specified number of bits to the data stream. The fill is always 0. The form of the statement is

```
fill fill-size['length'];
```

where *fill-size* is one of the following strings:

```
bit  
nibble  
byte  
word  
long
```

These strings declare a fill of 1, 4, 8, 16, or 32 bits (optionally multiplied by *length*). *length* is an expression whose value is less than or equal to 2147483647.

The following `fill` statements are equivalent:

```
fill word[2];  
fill long;  
fill bit[32];
```

The full form of a `type` statement specifying a fill might be as follows:

```
type 'XRES' {type-specification, fill bit[2];};
```

◆ **Note** `rez` supplies zeros as specified by `fill` and `align` statements. `derez` does not supply any values for `fill` or `align` statements; it just skips the specified number of bits, or skips bits until data is aligned as specified. ◆

Align specification Alignment causes `rez` to add fill bits of zero value until the data is aligned at the specified boundary. An alignment statement takes the following form:

```
align align-size ;
```

where *align-size* is one of the following strings:

```
nibble
```

```
byte
```

```
word
```

```
long
```

Alignment pads with zeros until data is aligned on a 4-bit, 8-bit, 16-bit, or 32-bit boundary. This alignment affects all data from the point where it is specified to the beginning of the next `align` statement.

Array type

An array is declared as follows:

```
[ wide ] array [ array-name | '[' length ']' ] { array-list } ;
```

The *array-list* argument, a list of type specifications, is repeated zero or more times. The `wide` option generates the array data in a wide display format (in `derez`)—the elements that make up the array list are separated by a comma and space instead of a comma, newline, and tab. Either *array-name* or [*length*] can be specified. *array-name* is an identifier.

If the array is named, then a preceding statement must refer to that array in a constant expression with the `$$Countof(array-name)` function; otherwise, `derez` is unable to decompile resources of this type. For example, in the following declaration, the `$$Countof` function returns the number of array elements (in this case, the number of strings) from the `resource` data.

```
type 'STR#' {      /*define a string list resource*/
    integer = $$Countof(StringArray);
    array StringArray {
        pstring;
    };
};
```

If [*length*] is specified, there must be exactly *length* elements.

Array elements are generated by commas. Commas are element separators. Semicolons (;) are element terminators. However, semicolons can be used as element separators, as in this example:

```
type 'xyzy' {
    array Increment {
        integer = $$ArrayIndex(Increment);
    };
};
```

```
resource 'xyzy' (1) {
    {          /* zero elements */
        ,
    }
};
```

```
resource 'xyzy' (3) {
    } /* two elements */
    ;;
}
};
```

```
/* The only way to specify one element in an array that has
   all constant elements is to use a semicolon terminator.
   */
```

```
resource 'xyzy' (4) {
    { /* one element */
        ;
    }
};
```

Switch type

The `switch` statement specifies a number of case statements for a given field or fields in the resource. The format is as follows:

```
switch {case-statement...};
```

where *case-statement* has the following form:

```
case case-name : [case-body];...
```

The *case-name* argument is an identifier. The *case-body* argument can contain any number of type specifications and must include a single constant declaration per case, in the following form:

```
key data-type = constant
```

Which case applies is based on the key value, as illustrated in this example:

```
type 'DITL' { /* dialog item list declaration from types.r */  
    type specifications...  
    switch { /* one of the following */  
        case Button:  
            boolean    enabled, disabled;  
            key bitstring[7] = 4; /* key value */  
            pstring;  
        case CheckBox:  
            boolean    enabled, disabled;  
            key bitstring[7] = 5; /* key value */  
            pstring;  
        ...and so on.  
    };  
};
```

Sample type statement

The following sample `type` statement is the standard declaration for a 'WIND' resource, taken from the `types.r` file.

```
type 'WIND'{  
  
    rect;                               /* boundsRect */  
  
    integer    documentProc, dBoxProc, plainDBox, /* procID */  
              altDBoxProc, noGrowDocProc,  
              zoomProc=8, rDocProc=16;  
  
    byte    invisible, visible;         /* visible */  
  
    fill byte;  
  
    byte    noGoAway, goAway;         /* has close box*/  
  
    fill byte;  
  
    unsigned hex longint;             /* refCon */  
  
    pstring    Untitled = "Untitled"; /* title */  
  
};
```

The `type` declaration consists of header information followed by a series of statements, each terminated by a semicolon (;). The header of the sample window declaration is

```
type 'WIND'
```

The header begins with the `type` keyword followed by the name of the resource type being declared—in this case, a window. You can specify a standard Macintosh resource type, as listed in the Resource Manager chapters of *Inside Macintosh*, Volumes I, IV, and V, or you can declare a resource type specific to your application.

The left brace ({) introduces the body of the declaration. The declaration continues for as many lines as necessary and is terminated by a matching right brace (}). You can write more than one statement on a line, and a statement can be on more than one line (like the `integer` statement in the example). Each statement represents a field in the resource data. Comments can appear anywhere that white space can appear in the resource description file; comments begin with `/*` and end with `*/` as in C.

*Symbol
definitions*

Symbolic names for data-type fields simplify the reading and writing of resource definitions. Symbol definitions have the form

name = *value* [, *name* = *value*]...

For numeric data, the = *value* part of the statement can be omitted. If a sequence of values consists of consecutive numbers, the explicit assignment can be left out, and if *value* is omitted, it's assumed to be one greater than the previous value. (The value is assumed to be 0 if it's the first value in the list.) This implicit assignment is true for bitstrings (and their derivatives, `byte`, `integer`, and `longint`). For example, in the following statement, the symbolic names `documentProc`, `dBoxProc`, `plainDBox`, `altDBoxProc`, and `noGrowDocProc` are automatically assigned the numeric values 0, 1, 2, 3, and 4.

```
integer      documentProc, dBoxProc, plainDBox,  
             altDBoxProc, noGrowDocProc,  
             zoomProc=8, rDocProc=16;
```

Memory is the only limit to the number of symbolic values that can be declared for a single field. There is also no limit other than memory to the number of names you can assign to a given value; for example, this statement is valid:

```
integer      documentProc=0, dBoxProc=1, plainDBox=2, altDBoxProc=3,  
             rDocProc=16,  
             Document=0, Dialog=1, DialogNoShadow=2, ModelessDialog=3,  
             DeskAccessory=16;
```

Labels

Labels are needed to support some of the more complicated resources such as 'NFNT' and Color QuickDraw resources. Use labels within a resource type declaration to calculate offsets and permit accessing of data at the labels.

Syntax *label* → *character*{*alphanum*} * ':'
character → '_' | A | B | C ...
number → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
alphanum → *character* | *number*

Description Labeled statements are valid only within a resource type declaration. Labels are local to each type declaration. A single label can appear on any statement.

In expressions, only the identifier portion of the label (that is, everything up to, but excluding, the colon) can be used. See “Declaring Labels Within Arrays,” later in this section.

The value of a label is always the offset—in bits—between the beginning of the resource and the position at which the label occurs when mapped to the resource data. In the following label definition example, the label is defined as a `cstring` string followed by an integer containing the bit count of the particular label:

```
type 'strb' {  
    cstring;  
endOfString:  
    integer = endOfString;  
};
```

Here is an example of this label:

```
resource 'strb' (8) {  
    "Hello"  
}
```

The label `cstring` is "Hello", followed by an integer containing the value 48. The value is calculated as follows, based on the definition of `cstring` (string with an added null byte) and the bit value of 8 provided for resource 'strb':

```
( len("Hello") [5] + null byte [1] ) * 8 [bits per byte] = 48
```

*Built-in
functions that
access resource
data*

In some cases, it is desirable to access the actual resource data that a label points to. Several built-in functions allow access to that data:

`$$BitField(label, startingPosition, numberOfBits)`

Return value of the bitstring of length *numberOfBits* (maximum 32) found at *startingPosition* bits from *label*.

`$$Byte(label)`

Return the byte found at *label*.

`$$Word(label)`

Return the word found at *label*.

`$$Long(label)`

Return the long word found at *label*.

For example, you could redefine the resource type 'STR' without using a `pstring` string. Here is the definition of 'STR' from `types.r`:

```
type 'STR' {  
    pstring;  
}
```

Here is a redefinition of 'STR' using labels:

```
type 'STR' {  
len:      byte = (stop - len) / 8 - 1;  
          string[$$Byte(len)];  
stop:     ;  
};
```

Declaring labels within arrays Labels declared within arrays can have many values. Each element in the array corresponds to a value for each label defined within the array. Array subscripts provide access to the individual values of these labels. Subscript values range from 1 to *n*, where *n* is the number of elements in the array. Labels within arrays that are nested in other arrays require multidimensional subscripts. Each level of nesting adds another subscript. The rightmost subscript varies most quickly. Here is a label definition example:

```
type 'test' {
    integer = $$CountOf(array1);
    array array1 {
        integer = $$CountOf(array2);
        array array2 {
foo:                integer;
        }
    };
};
```

Here is an example of 'test' in use:

```
resource 'test' (128) {
    {
        {1,2,3},
        {4,5}
    }
};
```

In the example just given, the label `foo` would take on these values:

<code>foo[1,1]</code>	<code>= 32</code>	<code>\$\$Word(foo[1,1])</code>	<code>= 1</code>
<code>foo[1,2]</code>	<code>= 48</code>	<code>\$\$Word(foo[1,2])</code>	<code>= 2</code>
<code>foo[1,3]</code>	<code>= 64</code>	<code>\$\$Word(foo[1,3])</code>	<code>= 3</code>
<code>foo[2,1]</code>	<code>= 96</code>	<code>\$\$Word(foo[2,1])</code>	<code>= 4</code>
<code>foo[2,2]</code>	<code>= 112</code>	<code>\$\$Word(foo[2,2])</code>	<code>= 5</code>

A new built-in function may be helpful in using labels within arrays:

`$$ArrayIndex (array-name)`

This function returns the current array index of the array *array-name*. An error occurs if this function is used anywhere outside the scope of the array *array-name*.

Label limitations The `derez` decompiler is basically a one-pass decompiler. In order for `derez` to decompile a given type, no expression within that type can contain more than one undefined label. Any label that occurs lexically after the expression is undefined. The use of a label within an expression defines the label.

The decompiler can keep track of one unknown value at a time, pending definition of the value. This example demonstrates an expression with more than one undefined label:

```
type 'test' {
    /* In the expression below, start is defined, next is
    undefined. */
    start: integer = next - start;
    /* In the expression below, next is defined because it
    was used in a previous expression, but final is
    undefined. */
    middle: integer = final - next;
    next: integer;
    final: /* final is now defined */
};
```

In the example, if the expression defining `middle` (`middle: integer = final - next;`) had not been encountered while the value for `next` remained unresolved, then the decompiler could have correctly processed the other statements. Alternatively, if `start` had been defined in terms of `middle` (`start: integer = middle - start;`), then the entire expression could have been correctly processed.

The `rez` compiler can compile types that have expressions containing more than one undefined label, but `derez` is not able to decompile those resources and simply generates data resource statements.

◆ **Note** The label specified in `$$BitField(label)`, `$$Byte(label)`, `$$Word(label)`, or `$$Long(label)` must occur lexically before the expression; otherwise, an error is generated. ◆

Two examples The first example shows the modified 'ppat' declaration using the new rez labels.

◆ **Note** Boldface text in the examples indicates the differences between the prior and current versions of the type definition of 'ppat', that is, where using labels has changed the definitions. ◆

Without using labels, the whole end section of the resource (everything after the PixelData label) would have to be combined into a single hex string. Using labels, you can express the complete 'ppat' definition in rez language.

```
type 'ppat' {
    /* PixPat record */
    integer oldPattern, /* pattern type */
        newPattern,
        ditherPattern;
    unsigned longint = Pixmap / 8; /* offset to pixmap */
    unsigned longint = PixelData / 8; /* offset to data */
    fill long; /* expanded pixel image */
    fill word; /* pattern valid flag */
    fill long; /* expanded pattern */
    hex string [8]; /* old-style pattern */
    /* Pixmap record */
Pixmap:
    fill long; /* base address */
    unsigned bitstring[1] = 1; /* new Pixmap flag */
    unsigned bitstring[2] = 0; /* must be 0 */
    unsigned bitstring[13]; /* offset to next row */
    rect; /* bitmap bounds */
    integer; /* Pixmap vers number */
    integer unpacked; /* packing format */
    unsigned longint; /* size of pixel data */
    unsigned hex longint; /* h. resolution (ppi) (fixed) */
    unsigned hex longint; /* v. resolution (ppi) (fixed) */
}
```

```

integerchunky, chunkyPlanar, planar;          /* pixel storage format      */
integer;                                       /* # bits in pixel          */
integer;                                       /* # components in pixel    */
integer;                                       /* # bits per field         */
unsigned longint;      /* offset to next plane      */
unsigned longint = ColorTable / 8;          /* offset to color table    */
fill long;                                     /* reserved                  */

PixelData:
    hex string [(ColorTable - PixelData) / 8];

ColorTable:
    unsigned hex longint;                       /* ctSeed                   */
    integer;                                    /* transIndex               */
    integer = $$Countof(ColorSpec) - 1; /* ctSize                   */
    wide array ColorSpec {
        integer;      /* value                    */
        unsigned integer; /* RGB: red                */
        unsigned integer; /* green                   */
        unsigned integer; /* blue                    */
    };
};

```

Here is another example of a new resource definition. In this example, the `$$BitField()` function is used to access information stored in the resource in order to calculate the size of the various data areas added at the end of the resource. Without labels, all of the data would have to be combined into one hex string. As in the preceding example, boldface text indicates changes for the current (label) version.

```

type 'cicn' {
    /* IconPMap (pixMap) record */
    fill long;                                     /* base address            */
    unsigned bitstring[1] = 1;                    /* new pixMap flag        */
    unsigned bitstring[2] = 0;                    /* must be 0              */
pMapRowBytes: unsigned bitstring[13];          /* offset to next row     */
}

```

(continued) ➡

```

Bounds: rect; /* bitmap bounds */
    integer; /* PixMap vers number */
    integer unpacked; /* Packing format */
    unsigned longint; /* size of pixel data */
    unsigned hex longint; /* h. resolution (ppi) (fixed)*/
    unsigned hex longint; /* v. resolution (ppi) (fixed)*/
    integer chunky, chunkyPlanar, planar; /* pixel storage format */
    integer; /* # bits in pixel */
    integer; /* # components in pixel */
    integer; /* # bits per field */
    unsigned longint; /* offset to next plane */
    unsigned longint; /* offset to color table */
    fill long; /* reserved */
    /* IconMask (bitMap) record */
    fill long; /* base address */
maskRowBytes: integer; /* rrow bytes */
    rect; /* bitmap bounds */

    /* IconBMap (bitMap) record */
    fill long; /* base address */
iconBMapRowBytes: integer; /* Row bytes */
    rect; /* Bitmap bounds */
    fill long; /* Handle placeholder */

    /* Mask data */
    hex string [$$Word(maskRowBytes) *
        ($$BitField(Bounds, 32, 16) /*bottom*/
        - $$BitField(Bounds, 0, 16 /*top*/)];

    /* BitMap data */
    hex string [$$Word(iconBMapRowBytes) *
        ($$BitField(Bounds, 32, 16) /*bottom*/
        - $$BitField(Bounds, 0, 16 /* top */)];

```



```

/* Color Table */
unsigned hex longint; /* ctSeed      */
integer;             /* transIndex */
integer = $$Countof(ColorSpec) - 1; /* ctSize     */
wide array ColorSpec {
    integer;         /* value      */
    unsigned integer; /* RGB: red   */
    unsigned integer; /* green     */
    unsigned integer; /* blue*/
};

/* PixelMap data */
hex string [$$BitFields(pMapRowBytes, 0, 13) *
($$BitFields(Bounds, 32, 16)           /* bottom      */
- $$BitFields(Bounds, 0, 16)           /*top*/)];
};

```

Preprocessor directives

Preprocessor directives substitute macro definitions and include files and provide if-then-else processing before other `rez` processing takes place.

The syntax of the `rez` preprocessor is similar to that of the C-language preprocessor. Preprocessor directives must observe these rules and restrictions:

- Each preprocessor statement must be expressed on a single line and placed at the beginning of the line.
- The number sign (`#`) must be the first character on the line of the preprocessor statement (except for spaces and tabs).
- The placeholder *identifier* (used in macro names) can consist of letters (`A-Z`, `a-z`), digits (`0-9`), or the underscore character (`_`). Identifiers cannot start with a digit, are not case sensitive, and can be of any length.

Variable definitions

The `#define` and `#undef` directives let you assign values to identifiers:

```
#define macro data
#undef macro
```

The `#define` directive causes any occurrence of the identifier *macro* to be replaced with the text *data*. You can extend a macro over several lines by ending the line with the backslash character (`\`), which functions as the `rez` escape character. Quotation marks within strings must also be escaped, as shown here:

```
#define poem "I wander \
thru\' each \
charter\'d street"
```

`#undef` removes the previously defined identifier *macro*. Macro definitions can also be removed with the `-undef` option on the `rez` command line.

The following macros are predefined:

<i>Variable</i>	<i>Value</i>
TRUE	1
FALSE	0
rez	1 if rez is running; 0 if derez is running
derez	1 if derez is running; 0 if rez is running

include directives

The `#include` directive reads a text file by using this syntax:

```
#include filename
```

This directive includes the text file *filename*. The maximum directory nesting is to 10 levels. Here is an example of an `include` directive:

```
#include /mac/lib/rincludes/mytypes.r
```

Note that the `#include` preprocessor directive, which includes a file, is different from the `include` statement, described earlier in this appendix, which copies resources from another file.

If-then-else processing

The following directives provide conditional processing:

```
#if expression
[ #elif expression ]
[ #else ]
#endif
```

◆ **Note** The placeholder *expression* is defined in the section “Expressions,” later in this appendix. With the `#if` and `#elif` directives, *expression* can also include the following expression:

```
defined identifier                    or                    defined '(' identifier ')'
```

 ◆

The following directives can be used in place of `#if`:

```
#ifdef macro
#ifdef macro
```

Here is an example of if-then-else processing:

```
#define Thai

Resource 'STR ' (199) {
#ifdef English
    "Hello"
#elif defined (French)
    "Bonjour"
#elif defined (Thai)
    "Sawati"
#elif defined (Japanese)
    "Konnichiwa"
#endif
};
```

Print directive

The `#printf` directive is provided to aid in debugging resource description files:

```
#printf (format-string, arguments...)
```

The format of the `#printf` statement is exactly the same as that of the `printf` statement in the C language, with one exception: There can be no more than 20 arguments. (The same restriction applies to the `$$Format` function.) The `#printf` directive writes its output to standard error. Note that the `#printf` directive does *not* end with a semicolon.

Here is an example of the use of the print directive:

```
#define    Tuesday    3
#ifdef Monday
#printf("The day is Monday, day #%d\n", Monday)
#elif defined(Tuesday)
#printf("The day is Tuesday, day #%d\n", Tuesday)
#elif defined(Wednesday)
#printf("The day is Wednesday, day #%d\n", Wednesday)
#elif defined(Thursday)
#printf("The day is Thursday, day #%d\n", Thursday)
#else
#printf("DON'T KNOW WHAT DAY IT IS!\n")
#endif
```

The file just listed generates the following text:

```
The day is Tuesday, day #3
```

Resource description syntax

This section describes the details of the resource description syntax. It includes numbers, literals, expressions, variables, functions, and strings.

Numbers and literals

All arithmetic is performed as 32-bit signed arithmetic. The syntax uses the basic constants described in Table E-1.

Table E-1 Numeric constants

Numeric type	Form	Meaning
decimal	<i>nm...</i>	Signed decimal constant between 4294967295 and -2147483648.
hex	<i>0Xhhh...</i>	Signed hexadecimal constant between 0X7FFFFFFF and 0X80000000.
	<i>\$hhh...</i>	Alternate form for hexadecimal constants.
octal	<i>0ooo...</i>	Signed octal constant between 01777777777 and 020000000000.
binary	<i>0Bbbb...</i>	Signed binary constant between 0B11111111111111111111111111111111 and 0B10000000000000000000000000000000.
literal	' <i>aaaa</i> '	A literal with one to four characters. Characters are printable ASCII characters or escape characters (defined later in this section). If there are fewer than four characters in the literal, then the characters to the left (high bits) are assumed to be \$00. Characters that are not in the printable character set and are not the characters \ ' or \\ (which have special meanings) can be escaped according to the character escape rules. (See "Strings," later in this appendix.)

Literals and numbers are treated in the same way by the resource compiler. A literal is a value within single quotation marks; for instance, 'A' is a number with the value 65, whereas "A" is the character A expressed as a string. Both are represented in memory by the bitstring 01000001. (Note, however, that "A" is not a valid number and 'A' is not a valid string.) The following numeric expressions are equivalent:

'B'
66
'A'+1

Literals are padded with nulls on the left side so that the literal 'ABC' is stored as shown in Figure E-3.

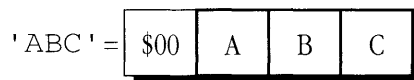


Figure E-3 Padding of literals

Expressions

An expression can consist of simply a number or a literal. Expressions can also include numeric variables and the system functions.

Table E-2 lists the operators in order of precedence, with highest precedence first. Groupings indicate equal precedence. Evaluation is always from left to right when the priority is the same. Variables are defined after the table.

Table E-2 Resource-description expression operators

Operator	Meaning
1. $(expr)$	Parentheses can be used in the normal manner to force precedence in expression calculation.
2. $- expr$ $\sim expr$ $! expr$	Arithmetic (two's complement) negation of $expr$. Bitwise (one's complement) negation of $expr$. Logical negation of $expr$.
3. $expr1 * expr2$ $expr1 / expr2$ $expr1 \% expr2$	Multiplication. Division. Remainder from dividing $expr1$ by $expr2$.
4. $expr1 + expr2$ $expr1 - expr2$	Addition. Subtraction.
5. $expr1 << expr2$ $expr1 >> expr2$	Shift left—shift $expr1$ left by $expr2$ bits. Shift right—shift $expr1$ right by $expr2$ bits.
6. $expr1 > expr2$ $expr1 >= expr2$ $expr1 < expr2$ $expr1 <= expr2$	Greater than. Greater than or equal to. Less than. Less than or equal to.
7. $expr1 == expr2$ $expr1 != expr2$	Equal to. Not equal to.
8. $expr1 \& expr2$	Bitwise AND.
9. $expr1 \wedge expr2$	Bitwise XOR.
10. $expr1 expr2$	Bitwise OR.
11. $expr1 \&\& expr2$	Logical AND.
12. $expr1 expr2$	Logical OR.

Note: The logical operators $!$, $>$, $>=$, $<$, $<=$, $==$, $!=$, $\&\&$, and $||$ evaluate to 1 (TRUE) or 0 (FALSE).

Variables and functions

Some resource compiler variables contain commonly used values. All resource compiler variables start with `$$` followed by an alphanumeric identifier.

String values

The following variables and functions have string values. (Typical values are given in parentheses.)

`$$Date`

Current date function, which is useful for putting time-stamps into the resource file. The format is generated through the ROM call `IUDateString`. (An example of this format is “Thursday, June 21, 1990”.)

`$$Format ("format-string" , arguments)`

Format function, which works just like the `#printf` directive except that `$$Format` returns a string rather than printing to standard output. (For a description of print directives, see “Print Directive,” earlier in this appendix.)

`$$Name`

Name of the current resource. The current resource is the resource being generated by a `resource` statement, being included with an `include` statement, being deleted by a `delete` statement, or being changed by a `change` statement. In addition to the `$$Name` string variable, three numeric variables (`$$Type`, `$$ID`, and `$$Attributes`) refer to the current resource. They are described in the next section, “Numeric Values.”

Here is example showing the use of three of these four variables in an `include` statement that includes all ‘DRVR’ resources from one file and keeps the same information, while also setting the `SYSHEAP` attribute:

```
include "file" 'DRVR' (0:40) as 'DRVR' ($$ID,  
    $$Name, $$Attributes | 64) ;
```

The `$$Type`, `$$ID`, `$$Name`, and `$$Attributes` variables are undefined outside a `resource`, `include`, `delete`, or `change` statement.

`$$Resource("filename", 'type', ID | "resource-name")`

Resource read function, which reads the resource *type* with the ID *ID* or the name *resource-name* from the resource file *filename* and returns a string.

`$$Time`

Current time function, which is useful for time-stamping the resource file. The format is generated through the ROM call `TUTimeString`. (An example of this format is "7:50:54 AM".)

`$$Version`

Version number of the resource compiler. (An example of this format is "V3.0".)

Numeric values

The following variables and functions have numeric values.

`$$Attributes`

Attributes of the current resource. See the description of the `$$Name` string variable in the preceding section.

`$$BitField(label, startingPosition, numberOfBits)`

Return value of the bitstring of length *numberOfBits* (maximum 32) found at *startingPosition* bits from *label*.

`$$Byte(label)`

Return value of the byte found at *label*.

`$$Day`

Current day, range 1–31.

`$$Hour`

Current hour, range 0–23.

`$$ID`

ID of resource from the current resource. See the description of the `$$Name` string variable in the preceding section.

`$$Long (label)`

Return value of the long word found at *label*.

`$$Minute`

Current minute, range 0–59.

`$$Month`

Current month, range 1–12.

`$$PackedSize (Start, RowBytes, RowCount)`

Reference to the current resource. (See the description of the `$$Name` string variable in the preceding section.) Provided with an offset, *Start*, into the current resource and two integers, *RowBytes* and *RowCount*, this function calls the A/UX Toolbox utility routine `UnpackBits`, the number of times specified by *RowCount*, and returns the unpacked size of the data found at *start*. Use `$$PackedSize()` only for decompiling resource files. For an example that uses this function, see `/mac/lib/rincludes/pict.r`.

`$$ResourceSize`

Current size of resource in bytes. When you are decompiling, `$$ResourceSize` is the actual size of the resource being decompiled. When you are compiling, `$$ResourceSize` returns the number of bytes that have been compiled so far for the current resource. For an example that uses this function, see the `'KCHR'` resource in `/mac/lib/rincludes/systypes.r`.

`$$Second`

Current second, range 0–59.

`$$Type`

Type of resource from the current resource. See the description of the `$$Name` string variable in the preceding section.

`$$Weekday`

Current day of the week, range 1–7 (that is, Sunday–Saturday).

`$$Word (label)`

The word found at *label*.

`$$Year`

Current year.

Strings

There are two basic types of strings:

- Text string `"a..."`

A text string can contain any printable character except the double quotation mark (") and the backslash (\). These and other characters can be created through escape sequences. (See Table E-3.) The string `" "` is a valid string of length 0.

- Hex string `$"hh..."`

Spaces and tabs inside a hexadecimal string are ignored. There must be an even number of hexadecimal digits. The string `$" "` is a valid hexadecimal string of length 0.

Any two strings (hexadecimal or text) are concatenated if they are placed next to each other with only white space between them. (In this case, newlines and comments are considered white space.)

Figure E-4 shows a Pascal string declared as

```
pstring [10];
```

whose data definition is

```
"Hello"
```

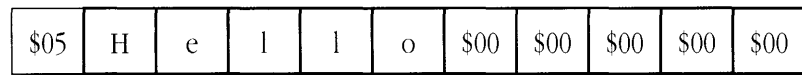


Figure E-4 Internal representation of a Pascal string

In the input file, string data is surrounded by quotation marks ("). You can continue a string on the next line. A separating token (for example, a comma) or brace signifies the end of the string data. A side effect of string continuation is that a sequence of two quotation marks (" ") is simply ignored. For example,

```
"Hello " "out "  
"there."
```

is the same string as

```
"Hello out there."
```

To place a quotation mark in a string, precede the quotation mark with a backslash (\").

Escape characters

The backslash character (\) is provided as an escape character to allow you to insert nonprintable characters in a string. For example, to include a return character in a string, you use the escape sequence `\r`. Table E-3 lists the valid escape sequences.

Table E-3 Resource compiler escape sequences

Escape sequence	Name	Hex value	Printable equivalent
<code>\t</code>	Tab	\$09	None
<code>\b</code>	Backspace	\$08	None
<code>\r</code>	Return	\$0D	None
<code>\n</code>	Newline	\$0A	None
<code>\f</code>	Form feed	\$0C	None
<code>\v</code>	Vertical tab	\$0B	None
<code>\?</code>	Rubout	\$7F	None
<code>\\</code>	Backslash	\$5C	\
<code>\'</code>	Single quotation mark	\$3A	'
<code>\"</code>	Double quotation mark	\$22	"

Note: Under the Macintosh OS, `\n` is treated as a return character (\$0D). Under the A/UX operating system, `\n` is treated as a line feed (\$0A). `\r` always follows the Macintosh convention.

You can also use octal, hexadecimal, decimal, and binary escape sequences to specify characters that do not have predefined escape equivalents. The forms are shown in Table E-4.

Table E-4 Numeric escape sequences

Base	Form	Number of digits	Example
2	<code>\0Bbbbbbbb</code>	8	<code>\0B01000001</code>
8	<code>\ooo</code>	3	<code>\101</code>
10	<code>\0Dddd</code>	3	<code>\0D065</code>
16	<code>\0Xhh</code>	2	<code>\0X41</code>
16	<code>\\$hh</code>	2	<code>\\$41</code>

Here are some examples of numeric escape sequences:

```

\077          /* 3 octal digits */
\0xFF        /* '0x' plus 2 hex digits */
\ $F1\ $F2\ $F3  /* '$' plus 2 hex digits */
\0d099      /* '0d' plus 3 decimal digits */

```

◆ **Note** An octal escape code consists of exactly three digits. For instance, to place an octal escape code with a value of 7 in the middle of an alphabetic string, write `AB\007CD`, not `AB\7CD`. ◆

You can use the `dereze` command-line option `-e` to print characters that would otherwise be escaped (characters preceded by a backslash, for example). Normally, characters with values between `$20` and `$D8` are printed as Macintosh characters. With this option, however, all characters (except null, newline, tab, backspace, form feed, vertical tab, and rubout) are printed as characters, not as escape sequences.

Appendix F: C Interface Library

Interface library files / F-2

Structures and calls by library / F-5

Calls in alphabetical order / F-63

The Macintosh C interface library, documented in this appendix, contains the C definitions of the constants, types, and functions defined in *Inside Macintosh* and used in the A/UX Toolbox. The information given here is the C equivalent of the Pascal definitions in the summary section at the end of each chapter of *Inside Macintosh*. For complete documentation of each of the constants, types, and functions defined here, see the corresponding section of *Inside Macintosh*. For a description of the functional differences between the standard Macintosh libraries as described in *Inside Macintosh* and the A/UX C versions, see Chapter 5, “A/UX and Macintosh User Interface Toolbox Differences.”

Libraries in this section appear in alphabetical order by library name, not in *Inside Macintosh* order.

Interface library files

The A/UX C definitions of the Macintosh libraries are provided in the header files in the directory `/usr/include/mac`. Include the header file for each software library (typically called a *manager* in the Macintosh environment) that you use in your program.

The material in this appendix is accurate as this manual goes to press, but the header files provided with your system may contain different information that reflects the most recent changes.

Many of the routines in the A/UX Toolbox call code that is in the Macintosh ROM. Most of these ROM routines use Pascal calling conventions, which differ from the C conventions used by A/UX. Ordinarily, the A/UX Toolbox handles the interface between the two. If you are writing your own definition functions or filter functions, or if you are making direct use of data in structures, you must take the differences into account. For more information, see “Converting Between C and Pascal Conventions” in Appendix C. (For a description of definition functions and filter functions, see *Inside Macintosh*, Volume I.)

The routine and parameter descriptions in the C interface libraries follow these conventions:

- A pointer to type `char` (printed `char *`) represents a pointer to a C-format string.
- A parameter of type `Str255` represents a Pascal-format string.

Table F-1 lists the libraries described in this section and the name of the header file for each library. Libraries appear in alphabetical order by library name. For a list of all libraries described in *Inside Macintosh* and their status in the A/UX Toolbox, see Chapter 5, “A/UX and Macintosh User Interface Toolbox Differences.”

The sections “Structures and Calls by Library” and “Calls in Alphabetical Order,” later in this appendix, provide information selected from the header files listed in Table F-1.

Table F-1 Interface library files

Library	Header file
32-Bit QuickDraw with Color QuickDraw	quickdraw.h
Color Picker	picker.h
Common type definitions	types.h
Control Manager	controls.h
Deferred Task Manager	dtask.h
Definitions for AUXDispatch	aux.h
Definitions for ROM	romdefs.h
Desk Manager	desk.h
Device Manager	devices.h
Dialog Manager	dialogs.h
Disk Driver	disks.h
Disk Initialization Package	diskinit.h
Event Manager, Operating System	osevents.h
Event Manager, Toolbox	events.h
File Manager	files.h
Font Manager	fonts.h
Gestalt Manager	gestalt.h
List Manager Package	lists.h
List of Macintosh traps	traps.h
Low-memory equates	sysequ.h
Memory Manager	memory.h
Menu Manager	menus.h
Notification Manager	notify.h
Package Manager	packages.h
Binary-Decimal Conversion Package	
Floating-Point Arithmetic and Transcendental Functions Packages	
International Utilities Package	
Standard File Package	
Palette Manager	palettes.h
Printing Manager	printing.h

(continued) ➡

Table F-1 Interface library files (*continued*)

Library	Header file
Print traps	printtraps.h
Process Manager	processes.h
Resource Manager	resources.h asd.h aux_rsrc.h
Scrap Manager	scrap.h
Script Manager	script.h
Segment Loader	segload.h
Serial Driver	serial.h
Shutdown Manager	shutdown.h
Slot Manager	slots.h
Sound Manager	sm.h soundinput.h soundinputpriv.h
String conversion between Pascal and C	strings.h
System Error Handler	errors.h
TextEdit	textedit.h
Time Manager	timer.h
Utilities, Operating System	osutils.h
Utilities, Toolbox	toolutils.h
Vertical Retrace Manager	retrace.h
Video Driver	video.h
Window Manager	windows.h

Most of these files contain data structures and calls; some contain only definitions or equates. These header files can be displayed, searched, and printed.

The next section, “Structures and Calls by Library,” lists the structures and calls in the header files listed in Table F-1. The subsequent section, “Calls in Alphabetical Order,” lists all calls in alphabetical order by name.

Structures and calls by library

This section lists the names of the structures and calls that are available in the header files in Table F-1. The structures and calls are arranged under the library name given in the table. For instance, the structures and calls available in `picker.h` are under “Color Picker.” See the header file itself for additional information.

Structure names are in alphabetical order. Calls are in alphabetical order by name, followed by the name of the return type for the call.

Chapter 5 contains additional information about those libraries that support Macintosh managers and provide other Macintosh support services. Where available, information will be found under the same library name in that chapter; for instance, information on the serial driver is under “Serial Driver.” Where information is available elsewhere, as with “Low-Memory Equates,” this appendix gives that reference.

32-Bit QuickDraw with Color QuickDraw

The following structures and calls are available in `quickdraw.h`:

Structure name

BitMap	FontInfo	PixMap
CCrsr	GammaTbl	PixPat
CGrafPort	GDevice	Polygon
CIcon	GrafPort	QDProcs
ColorSpec	GrafVars	qdvar
ColorTable	ITab	Region
CProcRec	MatchRec	ReqListRec
CQDProcs	PenState	RGBColor
Cursor	Picture	SProcRec

<i>Call</i>	<i>Return type</i>
AddComp();	void
addpt();	void
AddPt();	void
AddSearch();	void
AllocCursor();	void
BackColor();	void
BackPat();	void
BackPixPat();	void
CalcCMask();	void
CalcMask();	void
CharExtra();	void
CharWidth();	short
ClipRect();	void
CloseCPort();	void
ClosePicture();	void
ClosePoly();	void
ClosePort();	void
CloseRgn();	void
Color2Index();	long
ColorBit();	void
CopyBits();	void
CopyMask();	void
CopyPixMap();	void
CopyPixPat();	void
CopyRgn();	void
DelComp();	void
DelSearch();	void
DiffRgn();	void

<i>Call</i>	<i>Return type</i>
DisposCCursor();	void
DisposCIcon();	void
DisposCTable();	void
DisposeRgn();	void
DisposGDevice();	void
DisposPixMap();	void
DisposPixPat();	void
DrawChar();	void
DrawPicture();	void
drawstring();	void
DrawString();	void
DrawText();	void
EmptyRect();	Boolean
EmptyRgn();	Boolean
equalpt();	Boolean
EqualPt();	Boolean
EqualRect();	Boolean
EqualRgn();	Boolean
EraseArc();	void
EraseOval();	void
ErasePoly();	void
EraseRect();	void
EraseRgn();	void
EraseRoundRect();	void
FillArc();	void
FillCArc();	void
FillCOval();	void
FillCPoly();	void

<i>Call</i>	<i>Return type</i>
FillCRect();	void
FillCRgn();	void
FillCRoundRect();	void
FillOval();	void
FillPoly();	void
FillRect();	void
FillRgn();	void
FillRoundRect();	void
ForeColor();	void
FrameArc();	void
FrameOval();	void
FramePoly();	void
FrameRect();	void
FrameRgn();	void
FrameRoundRect();	void
GetBackColor();	void
GetCCursor();	CCrsrHandle
GetCIcon();	CIconHandle
GetClip();	void
GetCPixel();	void
GetCTable();	CTabHandle
GetCTSeed();	long
GetDeviceList();	GDHandle
GetFontInfo();	void
GetForeColor();	void
GetGDevice();	GDHandle
GetMainDevice();	GDHandle
GetMaskTable();	Ptr

<i>Call</i>	<i>Return type</i>
GetMaxDevice();	GDHandle
GetNextDevice();	GDHandle
GetPen();	void
GetPenState();	void
GetPixel();	Boolean
GetPixPat();	PixPatHandle
GetPort();	void
GetSubTable();	void
GlobalToLocal();	void
GrafDevice();	void
HideCursor();	void
HidePen();	void
HiliteColor();	void
Index2Color();	void
InitCPort();	void
InitCursor();	void
InitGDevice();	void
InitGraf();	void
InitPort();	void
InsetRect();	void
InsetRgn();	void
InvertArc();	void
InvertColor();	void
InvertOval();	void
InvertPoly();	void
InvertRect();	void
InvertRgn();	void
InvertRoundRect();	void

<i>Call</i>	<i>Return type</i>
KillPicture();	void
KillPoly();	void
Line();	void
LineTo();	void
LocalToGlobal();	void
MakeITable();	void
MakeRGBPat();	void
MapPoly();	void
MapPt();	void
MapRect();	void
MapRgn();	void
MeasureText();	void
Move();	void
MovePortTo();	void
MoveTo();	void
NewGDevice();	GDHandle
NewPixMap();	PixMapHandle
NewPixPat();	PixPatHandle
NewRgn();	RgnHandle
ObscureCursor();	void
OffsetPoly();	void
OffsetRect();	void
OffsetRgn();	void
OpColor();	void
OpenCPort();	void
OpenPicture();	PicHandle
OpenPoly();	PolyHandle
OpenPort();	void

<i>Call</i>	<i>Return type</i>
OpenRgn();	void
PaintArc();	void
PaintOval();	void
PaintPoly();	void
PaintRect();	void
PaintRgn();	void
PaintRoundRect();	void
PenMode();	void
PenNormal();	void
PenPat();	void
PenPixPat();	void
PenSize();	void
PicComment();	void
PlotCIcon();	void
PortSize();	void
ProtectEntry();	void
pt2rect();	void
Pt2Rect();	void
ptinrect();	Boolean
PtInRect();	Boolean
ptinrgn();	Boolean
PtInRgn();	Boolean
pttoangle();	void
PtToAngle();	void
QDError();	short
Random();	short
RealColor();	Boolean
RectInRgn();	Boolean

<i>Call</i>	<i>Return type</i>
RectRgn();	void
ReserveEntry();	void
RestoreEntries();	void
RGBBackColor();	void
RGBForeColor();	void
SaveEntries();	void
ScalePt();	void
ScrollRect();	void
SectRect();	Boolean
SectRgn();	void
SeedCFill();	void
SeedFill();	void
SetCCursor();	void
SetClientID();	void
SetClip();	void
SetCPixel();	void
SetCursor();	void
SetDeviceAttribute();	void
SetEmptyRgn();	void
SetEntries();	void
SetGDevice();	void
SetOrigin();	void
SetPenState();	void
SetPort();	void
SetPortBits();	void
SetPortPix();	void
SetPt();	void
SetRect();	void

<i>Call</i>	<i>Return type</i>
SetRectRgn();	void
SetStdCProcs();	void
SetStdProcs();	void
ShowCursor();	void
ShowPen();	void
SpaceExtra();	void
StdArc();	void
StdBits();	void
StdComment();	void
StdGetPic();	void
stdline();	void
StdLine();	void
StdOval();	void
StdPoly();	void
StdPutPic();	void
StdRect();	void
StdRgn();	void
StdRRect();	void
stdtext();	void
StdText();	void
StdTxMeas();	short
stringwidth();	short
StringWidth();	short
stuffhex();	void
StuffHex();	void
subpt();	void
SubPt();	void
TestDeviceAttribute();	Boolean

<i>Call</i>	<i>Return type</i>
TextFace();	void
TextFont();	void
TextMode();	void
TextSize();	void
TextWidth();	short
UnionRect();	void
UnionRgn();	void
XorRgn();	void

Color Picker

The following structures and calls are available in `picker.h`:

Structure name

CMYColor
HSLColor
HSVColor

<i>Call</i>	<i>Return type</i>
CMY2RGB();	void
Fix2SmallFract();	SmallFract
GetColor();	Boolean
HSL2RGB();	void
HSV2RGB();	void
RGB2CMY();	void
RGB2HSL();	void
RGB2HSV();	void
SmallFract2Fix();	Fixed

Common type definitions

The following structures and calls are available in `types.h`:

Structure name

`comp`

`Rect`

`Point`

Call

Return type

`Debugger();`

`void`

`debugstr();`

`void`

`DebugStr();`

`void`

Control Manager

The following structures and calls are available in `controls.h`:

Structure name

`AuxCtlRec`

`ControlRecord`

`CtlCTab`

Call

Return type

`DisposeControl();`

`void`

`dragcontrol();`

`void`

`DragControl();`

`void`

`Draw1Control();`

`void`

`DrawControls();`

`void`

`findcontrol();`

`short`

`FindControl();`

`short`

`GetAuxCtl();`

`Boolean`

`GetCRefCon();`

`long`

<i>Call</i>	<i>Return type</i>
getctitle();	void
GetCTitle();	void
GetCtlAction();	ProcPtr
GetCtlMax();	short
GetCtlMin();	short
GetCtlValue();	short
GetCVariant();	short
GetNewControl();	ControlHandle
HideControl();	void
HiliteControl();	void
KillControls();	void
MoveControl();	void
newcontrol();	ControlHandle
NewControl();	ControlHandle
SetCRefCon();	void
setctitle();	void
SetCTitle();	void
SetCtlAction();	void
SetCtlColor();	void
SetCtlMax();	void
SetCtlMin();	void
SetCtlValue();	void
ShowControl();	void
SizeControl();	void
testcontrol();	short
TestControl();	short
trackcontrol();	short
TrackControl();	short
UpdtControl();	void

Deferred Task Manager

The following structure and calls are available in `dtask.h`:

Structure name

DeferredTask New in this release.

Call

Return type

DTInstall(); OSErr New in this release.

Definitions for AUXDispatch

The following structures and calls are available in `aux.h`. The section “AUXDispatch Trap” in Chapter 3 contains additional information, including the selector codes used with this trap.

Structure name

AuxSigio
ForkExecRec
GetAnyEventRec
IDToPathRec
TBLaunchRec

Call

Return type

AUXCOFFLaunch(); pascal short New in this release.
AUXDispatch(); pascal long

Definitions for ROM

No structures or calls are available in `romdefs.h`, which provides slot declaration values for ROMs.

Desk Manager

The following calls are available in `desk.h`, which has no structures:

<i>Call</i>	<i>Return type</i>
<code>CloseDeskAcc()</code> ;	void
<code>opendeskacc()</code> ;	short
<code>OpenDeskAcc()</code> ;	short
<code>SystemClick()</code> ;	void
<code>SystemEdit()</code> ;	Boolean
<code>SystemEvent()</code> ;	Boolean
<code>SystemMenu()</code> ;	void
<code>SystemTask()</code> ;	void

Device Manager

The following structures and calls are available in `devices.h`:

Structure name

AuxDCE

DctlEntry

<i>Call</i>	<i>Return type</i>
<code>CloseDriver()</code> ;	OSErr
<code>Control()</code> ;	OSErr
<code>GetDctlEntry()</code> ;	DctlHandle
<code>KillIO()</code> ;	OSErr
<code>opendriver()</code> ;	OSErr
<code>OpenDriver()</code> ;	OSErr
<code>PBControl()</code> ;	OSErr
<code>PBKillIO()</code> ;	OSErr

<i>Call</i>	<i>Return type</i>
PBStatus();	OSErr
SetChooserAlert();	Boolean
Status();	OSErr

Dialog Manager

The following structures and calls are available in `dialogs.h`:

Structure name

AlertTemplate
DialogRecord
DialogTemplate

<i>Call</i>	<i>Return type</i>
Alert();	short
CautionAlert();	short
CloseDialog();	void
CouldAlert();	void
CouldDialog();	void
DialogSelect();	Boolean
DisposDialog();	void
DlgCopy();	void
DlgCut();	void
DlgDelete();	void
DlgPaste();	void
DrawDialog();	void
ErrorSound();	void
findditem();	short
FindDItem();	short
FreeAlert();	void

<i>Call</i>	<i>Return type</i>
FreeDialog();	void
GetAlrtStage();	short
GetDItem();	void
getitext();	void
GetIText();	void
GetNewDialog();	DialogPtr
HideDItem();	void
InitDialogs();	void
IsDialogEvent();	Boolean
ModalDialog();	void
newcdialog();	DialogPtr
NewCDialog();	DialogPtr
newdialog();	DialogPtr
NewDialog();	DialogPtr
NoteAlert();	short
paramtext();	void
ParamText();	void
ResetAlrtStage();	void
SelIText();	void
SetDAFont();	void
SetDItem();	void
setitext();	void
SetIText();	void
ShowDItem();	void
StopAlert();	short
UpdtDialog();	void

Disk Driver

The following structures and calls are available in `disks.h`:

Structure name

`DrvSts`

`DrvSts2`

Call

Return type

`DiskEject()`;

`OSErr`

`SetTagBuffer()`;

`OSErr`

`DriveStatus()`;

`OSErr`

Disk Initialization Package

The following structure and calls are available in `diskinit.h`:

Structure name

`HFSDefaults`

Call

Return type

`DIload()`;

`void`

`DIunload()`;

`void`

`dibadmnt()`;

`OSErr`

`DIBadMount()`;

`short`

`DIFormat()`;

`OSErr`

`DIVerify()`;

`OSErr`

`DIZero()`;

`OSErr`

`dizero()`;

`OSErr`

Event Manager, Operating System

The following structure and calls are available in `osevents.h`:

Structure name

EvQEL

<i>Call</i>	<i>Return type</i>
FlushEvents();	void
GetEvQHdr();	QHdrPtr
GetOSEvent();	Boolean
OSEventAvail();	Boolean
PostEvent();	OSErr
PPostEvent();	OSErr
SetEventMask();	void

Event Manager, Toolbox

The following structure and calls are available in `events.h`:

Structure name

EventRecord

<i>Call</i>	<i>Return type</i>
Button();	Boolean
EventAvail();	Boolean
GetCaretTime();	unsigned long
GetDbtTime();	unsigned long
GetKeys();	void
GetMouse();	void
GetNextEvent();	Boolean
StillDown();	Boolean

<i>Call</i>	<i>Return type</i>
TickCount ();	unsigned long
WaitMouseUp ();	Boolean
WaitNextEvent ();	Boolean

File Manager

The following structures and calls are available in `files.h`:

Structure name

AFPVolMountBlock	FCBPBRec	HVolumeParam
AFPVolMountInfo	FIDParam	IOParam
CatPositionRec	FileParam	MultiDevParam
CInfoPBRec	FInfo	NumVersion
CMovePBRec	ForeignPrivParam	ObjParam
CntrlParam	FSSpec	ParamBlockRec
CopyParam	FXInfo	SlotDevParam
CSParam	GetVolParmsInfoBuffer	VCB
DInfo	HFileInfo	VersRec
DirInfo	HFileParam	VolumeParam
DrvQEl	HIOParam	WDPParam
DTPBRec	HParamBlockRec	WDPBRec
DXInfo		

<i>Call</i>	<i>Return type</i>
AddDrive ();	void
Allocate ();	OSErr
AllocContig ();	OSErr
CatMove ();	OSErr
CloseWD ();	OSErr

<i>Call</i>	<i>Return type</i>
<code>create();</code>	OSErr
<code>Create();</code>	OSErr
<code>DirCreate();</code>	OSErr
<code>eject();</code>	OSErr
<code>Eject();</code>	OSErr
<code>FInitQueue();</code>	void
<code>flushvol();</code>	OSErr
<code>FlushVol();</code>	OSErr
<code>FSClose();</code>	OSErr
<code>fsdelete();</code>	OSErr
<code>FSDelete();</code>	OSErr
<code>fsopen();</code>	OSErr
<code>FSOpen();</code>	OSErr
<code>FSRead();</code>	OSErr
<code>fsrename();</code>	OSErr
<code>FSWrite();</code>	OSErr
<code>GetDrvQHdr();</code>	QHdrPtr
<code>GetEOF();</code>	OSErr
<code>getfinfo();</code>	OSErr
<code>GetFInfo();</code>	OSErr
<code>GetFPos();</code>	OSErr
<code>GetFSQHdr();</code>	QHdrPtr
<code>GetVCBQHdr();</code>	QHdrPtr
<code>getvinfo();</code>	OSErr
<code>GetVInfo();</code>	OSErr
<code>getvol();</code>	OSErr
<code>GetVol();</code>	OSErr
<code>GetVRefNum();</code>	OSErr

<i>Call</i>	<i>Return type</i>
GetWDInfo ();	OSErr
HCreate ();	OSErr
HDelete ();	OSErr
HGetFInfo ();	OSErr
HGetVol ();	OSErr
HOpen ();	OSErr
HOpenRF ();	OSErr
HRename ();	OSErr
HRstFLock ();	OSErr
HSetFInfo ();	OSErr
HSetFLock ();	OSErr
HSetVol ();	OSErr
openrf ();	OSErr
OpenRF ();	OSErr
OpenWD ();	OSErr
PBAllocate ();	OSErr
PBAllocContig ();	OSErr
PBCatMove ();	OSErr
PBClose ();	OSErr
PBCloseWD ();	OSErr
PBCreate ();	OSErr
PBDelete ();	OSErr
PBDirCreate ();	OSErr
PBEject ();	OSErr
PBFlushFile ();	OSErr
PBFlushVol ();	OSErr
PBGetCatInfo ();	OSErr
PBGetEOF ();	OSErr

<i>Call</i>	<i>Return type</i>
PBGetFCBInfo();	OSErr
PBGetFInfo();	OSErr
PBGetFPos();	OSErr
PBGetVInfo();	OSErr
PBGetVol();	OSErr
PBGetWDInfo();	OSErr
PBHCopyFile();	OSErr
PBHCreate();	OSErr
PBHDelete();	OSErr
PBHGetDirAccess();	OSErr
PBHGetFInfo();	OSErr
PBHGetLogInInfo();	OSErr
PBHGetVInfo();	OSErr
PBHGetVol();	OSErr
PBHGetVolParms();	OSErr
PBHMapID();	OSErr
PBHMapName();	OSErr
PBHMoveRename();	OSErr
PBHOpen();	OSErr
PBHOpenDeny();	OSErr
PBHOpenRF();	OSErr
PBHOpenRFDeny();	OSErr
PBHRename();	OSErr
PBHRstFLock();	OSErr
PBHSetDirAccess();	OSErr
PBHSetFInfo();	OSErr
PBHSetFLock();	OSErr
PBHSetVol();	OSErr

<i>Call</i>	<i>Return type</i>
PBLockRange();	OSErr
PBMountVol();	OSErr
PBOffLine();	OSErr
PBOpen();	OSErr
PBOpenRF();	OSErr
PBOpenWD();	OSErr
PBRead();	OSErr
PBRename();	OSErr
PBRstFLock();	OSErr
PBSetCatInfo();	OSErr
PBSetEOF();	OSErr
PBSetFInfo();	OSErr
PBSetFLock();	OSErr
PBSetFPos();	OSErr
PBSetFVers();	OSErr
PBSetVInfo();	OSErr
PBSetVol();	OSErr
PBUnlockRange();	OSErr
PBUnmountVol();	OSErr
PBWrite();	OSErr
Rename();	OSErr
rstfLock();	OSErr
RstFLock();	OSErr
SetEOF();	OSErr
setfinfo();	OSErr
SetFInfo();	OSErr
setflock();	OSErr
SetFLock();	OSErr

<i>Call</i>	<i>Return type</i>
SetFPos();	OSErr
setvol();	OSErr
SetVol();	OSErr
unmountvol();	OSErr
UnmountVol();	OSErr

Font Manager

The following structures and calls are available in `fonts.h`:

Structure name

AsscEntry	FontAssoc	NameTable
FamRec	FontRec	StyleTable
FMetricRec	KernEntry	WidEntry
FMInput	KernPair	WidTable
FMOutput	KernTable	WidthTable

<i>Call</i>	<i>Return type</i>
FMSwapFont();	FMOutPtr
FontMetrics();	void
getfnum();	void
GetFNum();	void
getfontname();	void
GetFontName();	void
InitFonts();	void
RealFont();	Boolean
SetFontLock();	void
SetFractEnable();	void
SetFScaleDisable();	void

Gestalt Manager

No structures or calls are available in `gestalt.h`:

List Manager Package

The following structure and calls are available in `lists.h`:

Structure name

ListRec

Call

Return type

LActivate();	void
LAddColumn();	short
LAddRow();	short
LAddToCell();	void
LAutoScroll();	void
lcellsize();	void
LCellSize();	void
lclick();	Boolean
LClick();	Boolean
LClrCell();	void
LDelColumn();	void
LDelRow();	void
LDispose();	void
LDoDraw();	void
ldraw();	void
LDraw();	void
LFind();	void
LGetCell();	void
LGetSelect();	Boolean

<i>Call</i>	<i>Return type</i>
<code>LLastClick()</code> ;	<code>Cell</code>
<code>lnew()</code> ;	<code>ListHandle</code>
<code>LNew()</code> ;	<code>ListHandle</code>
<code>LNextCell()</code> ;	<code>Boolean</code>
<code>LRect()</code> ;	<code>void</code>
<code>LScroll()</code> ;	<code>void</code>
<code>LSearch()</code> ;	<code>Boolean</code>
<code>LSetCell()</code> ;	<code>void</code>
<code>LSetSelect()</code> ;	<code>void</code>
<code>LSize()</code> ;	<code>void</code>
<code>LUpdate()</code> ;	<code>void</code>

List of Macintosh traps

No structures or calls are available in `traps.h`, which provides a list of definitions for A-line traps accessible through C code.

Low-memory equates

No structures or calls are available in `sysequ.h`, which provides definitions for low-memory global variables. See Appendix D for further information.

Memory Manager

The Memory Manager uses definitions provided in the files `memory.h` and `vmcalls.h`. The following structure and calls are available in the file `memory.h`:

Structure name

Zone

<i>Call</i>	<i>Return type</i>
<code>ApplicZone()</code> ;	THz
<code>BlockMove()</code> ;	void
<code>CompactMem()</code> ;	Size
<code>DisposHandle()</code> ;	void
<code>DisposPtr()</code> ;	void
<code>EmptyHandle()</code> ;	void
<code>FreeMem()</code> ;	long
<code>GetApplLimit()</code> ;	Ptr
<code>GetHandleSize()</code> ;	Size
<code>GetPtrSize()</code> ;	Size
<code>GetZone()</code> ;	THz
<code>GZSaveHnd()</code> ;	Handle
<code>HandleZone()</code> ;	THz
<code>HClrRBit()</code> ;	void
<code>HGetState()</code> ;	short
<code>HLock()</code> ;	void
<code>HNoPurge()</code> ;	void
<code>HPurge()</code> ;	void
<code>HSetRBit()</code> ;	void
<code>HSetState()</code> ;	void

<i>Call</i>	<i>Return type</i>
HUnlock();	void
InitApplZone();	void
InitZone();	void
MaxApplZone();	void
MaxBlock();	long
MaxMem();	Size
MemError();	OSErr
MFFreeMem();	long
MFMaxMem();	Size
MFTempDisposHandle();	void
MFTempHLock();	void
MFTempHUnlock();	void
MFTempNewHandle();	Handle
MFTopMem();	Ptr
MoreMasters();	void
MoveHHi();	void
NewEmptyHandle();	Handle
NewHandle();	Handle
NewPtr();	Ptr
PtrZone();	THz
PurgeMem();	void
PurgeSpace();	void
ReallocHandle();	void
RecoverHandle();	Handle
ResrvMem();	void

<i>Call</i>	<i>Return type</i>
SetApplBase();	void
SetApplLimit();	void
SetGrowZone();	void
SetHandleSize();	void
SetPtrSize();	void
SetZone();	void
StackSpace();	long
StripAddress();	Ptr
SystemZone();	THz
TopMem();	Ptr

The following structures and calls are available in `vmcalls.h`:

Structure name

LogicalToPhysicalTable	New in this release.
MemoryBlock	New in this release.

Call

Return type

DeferUserFn()	OSErr	New in this release.
GetPhysical()	OSErr	New in this release.
HoldMemory()	OSErr	New in this release.
LockMemory()	OSErr	New in this release.
LockMemoryContiguous()	OSErr	New in this release.
UnholdMemory()	OSErr	New in this release.
UnlockMemory()	OSErr	New in this release.

Menu Manager

The following structures and calls are available in `menus.h`:

Structure name

MCEntry

MenuInfo

Call

Return type

AddResMenu();	void
appendmenu();	void
AppendMenu();	void
CalcMenuSize();	void
CheckItem();	void
ClearMenuBar();	void
CountMItems();	short
DeleteMenu();	void
DelMCEntries();	void
DelMenuItem();	void
DisableItem();	void
DispMCInfo();	void
DisposeMenu();	void
DrawMenuBar();	void
EnableItem();	void
FlashMenuBar();	void
getitem();	void
GetItem();	void
GetItemCmd();	void
GetItemIcon();	void
GetItemMark();	void
GetItemStyle();	void
GetMCEntropy();	MCEntropyPtr

<i>Call</i>	<i>Return type</i>
GetMCInfo();	MCTableHandle
GetMenu();	MenuHandle
GetMenuBar();	Handle
GetMHandle();	MenuHandle
GetNewMBar();	Handle
HiliteMenu();	void
InitMenus();	void
InitProcMenu();	void
InsertMenu();	void
InsertResMenu();	void
insmenuitem();	void
InsMenuItem();	void
MenuChoice();	long
MenuKey();	long
menuselect();	long
MenuSelect();	long
newmenu();	MenuHandle
NewMenu();	MenuHandle
PopUpMenuSelect();	long
setitem();	void
SetItem();	void
SetItemCmd();	void
SetItemIcon();	void
SetItemMark();	void
SetItemStyle();	void
SetMCEntries();	void
SetMCInfo();	void
SetMenuBar();	void
SetMenuFlash();	void

Notification Manager

The following structure and calls are available in `notify.h`:

Structure name

NMRec

Call *Return type*

NMInstall(); OSErr

NMremove(); OSErr

Package Manager

The following structures and calls are available in `packages.h`:

Structure name

Int10Rec

Int11Rec

SFReply

Call *Return type*

InitAllPacks(); void

InitPack(); void

iucompstring(); short

IUCompString(); short

iudatepstring(); void

IUDatePString(); void

iudatesting(); void

IUDateString(); void

iuequalstring(); short

<i>Call</i>	<i>Return type</i>
IUEqualString();	short
IUGetIntl();	Handle
IUMagIDString();	short
IUMagString();	short
IUMetric();	Boolean
IUSetIntl();	void
iutimepstring();	void
IUTimePString();	void
iutimestring();	void
IUTimeString();	void
numtostring();	void
NumToString();	void
sfgetfile();	void
SFGetFile();	void
sfpgetfile();	void
SFPGetFile();	void
sfpputfile();	void
SFPPutFile();	void
sfputfile();	void
SFPutFile();	void
stringtonum();	void
StringToNum();	void

Palette Manager

The following structures and calls are available in `palettes.h`:

Structure name

ColorInfo

Palette

Call

Return type

<code>ActivatePalette();</code>	<code>void</code>
<code>AnimateEntry();</code>	<code>void</code>
<code>AnimatePalette();</code>	<code>void</code>
<code>CopyPalette();</code>	<code>void</code>
<code>CTab2Palette();</code>	<code>void</code>
<code>DisposePalette();</code>	<code>void</code>
<code>GetEntryColor();</code>	<code>void</code>
<code>GetEntryUsage();</code>	<code>void</code>
<code>GetNewPalette();</code>	<code>PaletteHandle</code>
<code>GetPalette();</code>	<code>PaletteHandle</code>
<code>InitPalettes();</code>	<code>void</code>
<code>NewPalette();</code>	<code>PaletteHandle</code>
<code>NSetPalette();</code>	<code>void</code>
<code>Palette2CTab();</code>	<code>void</code>
<code>PmBackColor();</code>	<code>void</code>
<code>PmForeColor();</code>	<code>void</code>
<code>SetEntryColor();</code>	<code>void</code>
<code>SetEntryUsage();</code>	<code>void</code>
<code>SetPalette();</code>	<code>void</code>

Printing Manager

The following structures and calls are available in `printing.h`:

Structure name

TDftBitsBlk	TPrInfo	TPrStl
TGetRotnBlk	TPrint	TPrXInfo
TGetRslBlk	TPrJob	TRslRec
TGnlData	TPrPort	TRslRg
TPfPgDir	TPrStatus	TSetRslBlk
TPrDlg		

Call

Return type

PrClose();	void
PrCloseDoc();	void
PrClosePage();	void
PrCtlCall();	void
PrDlgMain();	Boolean
PrDrvrClose();	void
PrDrvrDCE();	Handle
PrDrvrOpen();	void
PrDrvrVers();	short
PrError();	short
PrGeneral();	void
PrintDefault();	void
PrJobDialog();	Boolean
PrJobInit();	TPPrDlg
PrJobMerge();	void
PrNoPurge();	void
PrOpen();	void
PrOpenDoc();	TPPrPort

<i>Call</i>	<i>Return type</i>
PrOpenPage();	void
PrPicFile();	void
PrPurge();	void
PrSetError();	void
PrStlDialog();	Boolean
PrStlInit();	TPrDlg
PrValidate();	Boolean

Print traps

The following structures and calls are available in `printtraps.h`:

Structure name

TDftBitsBlk	TPrInfo	TPrStl
TGetRotnBlk	TPrint	TPrXInfo
TGetRslBlk	TPrJob	TRslRec
TGnlData	TPrPort	TRslRg
TPfPgDir	TPrStatus	TSetRslBlk
TPrDlg		

<i>Call</i>	<i>Return type</i>
PrClose();	void
PrCloseDoc();	void
PrClosePage();	void
PrCtlCall();	void
PrDlgMain();	Boolean
PrDrvrClose();	void
PrDrvrDCE();	Handle
PrDrvrOpen();	void

<i>Call</i>	<i>Return type</i>
PrDrvrVers();	short
PrError();	short
PrGeneral();	void
PrintDefault();	void
PrJobDialog();	Boolean
PrJobInit();	TPPrDlg
PrJobMerge();	void
PrNoPurge();	void
PrOpen();	void
PrOpenDoc();	TPPrPort
PrOpenPage();	void
PrPicFile();	void
PrPurge();	void
PrSetError();	void
PrStlDialog();	Boolean
PrStlInit();	TPPrDlg
PrValidate();	Boolean

Process Manager

The following structures and calls are available in `processes.h`:

Structure name

AppParameters	New in this release.
LaunchParamBlockRec	New in this release.
ProcessInfoRec	New in this release.
ProcessSerialNumber	New in this release.

<i>Call</i>	<i>Return type</i>	
GetCurrentProcess	OSErr	New in this release.
GetFrontProcess	OSErr	New in this release.
GetNextProcess	OSErr	New in this release.
GetProcessInformation	OSErr	New in this release.
LaunchApplication	OSErr	New in this release.
LaunchDeskAccessory	OSErr	New in this release.
SameProcess	OSErr	New in this release.
SetFrontProcess	OSErr	New in this release.
WakeUpProcess	OSErr	New in this release.

Resource Manager

Three header files support working with Macintosh resources: `resources.h`, `asd.h`, and `aux_rsrc.h`. The first two header files provide Macintosh OS structures and calls. The `aux_rsrc.h` header file provides UNIX calls. The following calls are available in `resources.h`, which has no structures:

<i>Call</i>	<i>Return type</i>
<code>addresource();</code>	<code>void</code>
<code>AddResource();</code>	<code>void</code>
<code>ChangedResource();</code>	<code>void</code>
<code>CloseResFile();</code>	<code>void</code>
<code>Count1Resources();</code>	<code>short</code>
<code>Count1Types();</code>	<code>short</code>
<code>CountResources();</code>	<code>short</code>
<code>CountTypes();</code>	<code>short</code>
<code>createresfile();</code>	<code>void</code>
<code>CreateResFile();</code>	<code>void</code>
<code>CurResFile();</code>	<code>short</code>

<i>Call</i>	<i>Return type</i>
DetachResource();	void
Get1IndResource();	Handle
Get1IndType();	void
get1namedresource();	Handle
Get1NamedResource();	Handle
Get1Resource();	Handle
GetIndResource();	Handle
GetIndType();	void
getnamedresource();	Handle
GetNamedResource();	Handle
GetResAttrs();	short
GetResFileAttrs();	short
getresinfo();	void
GetResInfo();	void
GetResource();	Handle
HCreateResFile();	void
HomeResFile();	short
HOpenResFile();	short
InitResources();	short
LoadResource();	void
MaxSizeRsrc();	long
openresfile();	short
OpenResFile();	short
openrfperm();	short
OpenRFPerm();	short
ReleaseResource();	void
ResError();	short
RGetResource();	Handle

<i>Call</i>	<i>Return type</i>
RmveResource();	void
RsrcMapEntry();	long
RsrcZoneInit();	void
SetResAttrs();	void
SetResFileAttrs();	void
setresinfo();	void
SetResInfo();	void
SetResLoad();	void
SetResPurge();	void
SizeResource();	long
Unique1ID();	short
UniqueID();	short
UpdateResFile();	void
UseResFile();	void
WriteResource();	void

The following structures and calls are available in `asd.h`:

Structure name

FileHeader HFile
 FileInfo VFile
 FinderInfo

<i>Call</i>	<i>Return type</i>
CloseASD();	int
OpenASD();	FileHandle
ReadASD();	long
SeekASD();	long
WriteASD();	long

The following structures and calls are available in `aux_rsrc.h`:

Structure name

<code>ResData</code>	New in this release.
<code>ResFile</code>	New in this release.
<code>ResHdr</code>	New in this release.
<code>ResMap</code>	New in this release.
<code>ResReference</code>	New in this release.
<code>TypeEntry</code>	New in this release.
<code>TypeList</code>	New in this release.

Call

Return type

<code>mrattr()</code> ;	short
<code>mrclose()</code> ;	int
<code>mrget()</code> ;	Resource
<code>mrgetnamed()</code> ;	Resource
<code>mrinfo()</code> ;	int
<code>mropen()</code> ;	ResHandle
<code>mrrel()</code> ;	void

Scrap Manager

The following structures and calls are available in `scrap.h`:

Structure name

`ScrapStuff`

Call

Return type

<code>GetScrap()</code> ;	long
<code>InfoScrap()</code> ;	PScrapStuff
<code>LoadScrap()</code> ;	long
<code>PutScrap()</code> ;	long

<i>Call</i>	<i>Return type</i>
UnloadScrap();	long
ZeroScrap();	long

Script Manager

The following structures and calls are available in `script.h`:

Structure name

BreakTable	ItlrcRecord	TogglePB
DateCacheRecord	Location	Token
FindBlockStatus	NumberParts	TPrJob
FormatString	StartLength	UntokenTable
Itl4Rec	TokenBlock	WideCharArr
ItlbRecord	TDftBitsBlk	

<i>Call</i>	<i>Return type</i>
Char2Pixel();	short
CharByte();	short
CharType();	short
DrawJust();	void
FindBlock();	struct FindBlockStatus
FindWord();	void
Font2Script();	short
FontScript();	short
Form2Str();	FormatStatus
FormStr2X();	FormatStatus
FormX2Str();	FormatStatus
GetAppFont();	short
GetDefFontSize();	short

<i>Call</i>	<i>Return type</i>
GetEnvirons();	long
GetFormatOrder();	void
GetMBarHeight();	short
GetScript();	long
GetSysFont();	short
GetSysJust();	short
HiliteText();	void
InitDateCache();	OSErr
IntlScript();	short
IULDateString();	void
IULTimeString();	void
KeyScript();	void
LineBreak();	LineBreakCode
LongDate2Secs();	void
LongSecs2Date();	void
LwrText();	void
MeasureJust();	void
ParseTable();	Boolean
Pixel2Char();	short
PortionText();	Fixed
ReadLocation();	void
SetEnvirons();	OSErr
SetScript();	OSErr
SetSysJust();	void
Str2Form();	FormatStatus
String2Date();	String2DateStatus
String2Time();	String2DateStatus
ToggleDate();	ToggleResults

<i>Call</i>	<i>Return type</i>
Tokenize();	TokenResult
Transliterate();	OSErr
UprText();	void
ValidDate();	short
VisibleLength();	long
WriteLocation();	void

Segment Loader

The following structure and calls are available in `segload.h`:

Structure name

AppFile

<i>Call</i>	<i>Return type</i>
ClrAppFiles();	void
CountAppFiles();	void
ExitToShell();	void
GetAppFiles();	void
getappparms();	void
GetAppParms();	void
UnloadSeg();	void

Serial Driver

The following structures and calls are available in `serial.h`:

Structure name

`SerShk`

`SerStaRec`

Call

Return type

`RamSDClose()`;

`void`

`RamSDOpen()`;

`OSErr`

`SerClrBrk()`;

`OSErr`

`SerGetBuf()`;

`OSErr`

`SerHShake()`;

`OSErr`

`SerReset()`;

`OSErr`

`SerSetBrk()`;

`OSErr`

`SerSetBuf()`;

`OSErr`

`SerStatus()`;

`OSErr`

Shutdown Manager

The following calls are available in `shutdown.h`, which has no structures:

Call

Return type

`ShutDwnInstall()`;

`void`

New in this release.

`ShutDwnRemove()`;

`void`

New in this release.

Slot Manager

The following structures and calls are available in `slots.h`:

Structure name

FHeaderRec	SInfoRecord
SDMRecord	SlotIntQElement
SEBlock	SpBlock

<i>Call</i>	<i>Return type</i>
-------------	--------------------

InitSDeclMgr();	OSErr
OpenSlot();	OSErr
SCalcSPointer();	OSErr
SCalcStep();	OSErr
SCardChanged();	OSErr
SCKCardStat();	OSErr
SDeleteSRTRec();	OSErr
SExec();	OSErr
SFindBigDevBase();	OSErr
SFindDevBase();	OSErr
SFindSInfoRecPtr();	OSErr
SFindSRsrcPtr();	OSErr
SFindStruct();	OSErr
SGetBlock();	OSErr
SGetCString();	OSErr
SGetDriver();	OSErr
SGetSRsrc();	OSErr
SGetSRsrcInfo();	OSErr
SGetTypesRsrc();	OSErr
SInitPRAMRecs();	OSErr
SInitSRsrcTable();	OSErr

<i>Call</i>	<i>Return type</i>
SInsertSRTRec();	OSErr
SIntInstall();	OSErr
SIntRemove();	OSErr
SNextSRsrc();	OSErr
SNextTypeSRsrc();	OSErr
SOffsetData();	OSErr
SPrimaryInit();	OSErr
SPtrToSlot();	OSErr
SPutPRAMRec();	OSErr
SReadByte();	OSErr
SReadDrvrName();	OSErr
SReadFHeader();	OSErr
SReadInfo();	OSErr
SReadLong();	OSErr
SReadPBSize();	OSErr
SReadPRAMRec();	OSErr
SReadStruct();	OSErr
SReadWord();	OSErr
SRsrcInfo();	OSErr
SSearchSRT();	OSErr
SSetsRsrcState();	OSErr
SUpdateSRT();	OSErr
SVersion();	OSErr

Sound Manager

Three header files support working with Macintosh resources: `sm.h`, `soundinput.h`, and `soundinputpriv.h`. The first header file provides Macintosh OS structures and calls. The last two provide the calls that allow sound input under UNIX. The following structures and calls are available in `sm.h`:

Structure name

<code>CmpSoundHeader</code>	<code>SndDoubleBuffer</code>
<code>ExtSoundHeader</code>	<code>SndDoubleBufferHeader</code>
<code>LeftOverBlock</code>	<code>SndListResource</code>
<code>ModifierStub</code>	<code>SoundHeader</code>
<code>SndChannel</code>	<code>StateBlock</code>
<code>SndCommand</code>	

Call

Return type

<code>aSndDisposeChannel()</code> ;	short
<code>aSndAddModifier()</code> ;	short
<code>aSndDoCommand()</code> ;	short
<code>aSndDoImmediate()</code> ;	short
<code>aSndPlay()</code> ;	short
<code>aSndControl()</code> ;	short
<code>aSndNewChannel()</code> ;	short

The following structure and calls are available in `soundinput.h`:

Structure name

<code>SPB</code>	New in this release.
------------------	----------------------

Call

Return type

<code>SetupAIFFHeader</code>	<code>OSErr</code>	New in this release.
<code>SetupSndHeader</code>	<code>OSErr</code>	New in this release.
<code>SndRecord</code>	<code>OSErr</code>	New in this release.
<code>SndRecordToFile</code>	<code>OSErr</code>	New in this release.

<i>Call</i>	<i>Return type</i>	
SPBBytesToMilliseconds	OSErr	New in this release.
SPBCloseDevice	OSErr	New in this release.
SPBGetDeviceInfo	OSErr	New in this release.
SPBGetIndexedDevice	OSErr	New in this release.
SPBGetRecordingStatus	OSErr	New in this release.
SPBMillisecondsToBytes	OSErr	New in this release.
SPBOpenDevice	OSErr	New in this release.
SPBPauseRecording	OSErr	New in this release.
SPBRecord	OSErr	New in this release.
SPBRecordToFile	OSErr	New in this release.
SPBResumeRecording	OSErr	New in this release.
SPBSetDeviceInfo	OSErr	New in this release.
SPBSignInDevice	OSErr	New in this release.
SPBSignOutDevice	OSErr	New in this release.
SPBStopRecording	OSErr	New in this release.

The following structures and calls are available in `soundinputpriv.h`:

Structure name

AppRefRec	New in this release.
DrvParamBlockRec	New in this release.
SndInGlobals	New in this release.

Call

<i>Call</i>	<i>Return type</i>	
SoundInDevice		New in this release.
SPBGetDefaultDevice	OSErr	New in this release.
SPBSetDefaultDevice	OSErr	New in this release.

String conversion between Pascal and C

The following calls are available in `strings.h`, which has no structures:

<i>Call</i>	<i>Return type</i>
<code>*p2cstr()</code> ;	<code>char</code>
<code>c2pstr()</code> ;	<code>StringPtr</code>

System Error Handler

The following calls are available in `errors.h`, which has no structures:

<i>Call</i>	<i>Return type</i>
<code>SysError()</code> ;	<code>void</code>

TextEdit

The following structures and calls are available in `textedit.h`:

Structure name

<code>LHElement</code>	<code>StyleRun</code>
<code>NullStRec</code>	<code>TERec</code>
<code>ScrpSTElement</code>	<code>TEStyleRec</code>
<code>STElement</code>	<code>TextStyle</code>
<code>StScrpRec</code>	

<i>Call</i>	<i>Return type</i>
<code>GetStylHandle()</code> ;	<code>TEStyleHandle</code>
<code>GetStylScrap()</code> ;	<code>StScrpHandle</code>
<code>SetClikLoop()</code> ;	<code>void</code>
<code>SetStylHandle()</code> ;	<code>void</code>
<code>SetStylScrap()</code> ;	<code>void</code>

<i>Call</i>	<i>Return type</i>
SetWordBreak();	void
TEActivate();	void
TEAutoView();	void
TECalText();	void
teclick();	void
TEClick();	void
TEContinuousStyle();	Boolean
TECopy();	void
TECustomHook();	void
TECut();	void
TEDeactivate();	void
TEDelete();	void
TEDispose();	void
TEFromScrap();	OSErr
TEGetHeight();	long
TEGetOffset();	short
TEGetPoint();	struct Point
TEGetScrapLen();	long
TEGetStyle();	void
TEGetText();	CharsHandle
TEIdle();	void
TEInit();	void
TEInsert();	void
TEKey();	void
TENew();	TEHandle
TENumStyles();	long
TEPaste();	void
TEPinScroll();	void

<i>Call</i>	<i>Return type</i>
TEReplaceStyle();	void
TEScrapHandle();	Handle
TEScroll();	void
TESelView();	void
TESetJust();	void
TESetScrapLen();	void
TESetSelect();	void
TESetStyle();	void
TESetText();	void
TEStylInsert();	void
TEStylNew();	TEHandle
TEStylPaste();	void
TEToScrap();	OSErr
TEUpdate();	void
TextBox();	void

Time Manager

The following structure and calls are available in `timer.h`:

Structure name

TMTask

<i>Call</i>	<i>Return type</i>
InsTime();	void
PrimeTime();	void
RmvTime();	void

Utilities, Operating System

The following structures and calls are available in `osutils.h`:

Structure name

<code>DateTimeRec</code>	<code>SysEnvRec</code>
<code>QElem</code>	<code>SysParmType</code>
<code>QHdr</code>	

Call

Return type

<code>Date2Secs()</code> ;	<code>void</code>	
<code>Delay()</code> ;	<code>void</code>	
<code>Dequeue()</code> ;	<code>OSErr</code>	
<code>DTInstall()</code> ;	<code>OSErr</code>	
<code>Enqueue()</code> ;	<code>void</code>	
<code>Environs()</code> ;	<code>void</code>	
<code>equalstring()</code> ;	<code>Boolean</code>	
<code>EqualString()</code> ;	<code>Boolean</code>	
<code>GetDateTime()</code> ;	<code>void</code>	
<code>GetMMUMode()</code> ;	<code>char</code>	
<code>GetSysPPtr()</code> ;	<code>SysPPtr</code>	New in this release.
<code>GetTime()</code> ;	<code>void</code>	
<code>GetTrapAddress()</code> ;	<code>long</code>	
<code>HandAndHand()</code> ;	<code>OSErr</code>	
<code>HandToHand()</code> ;	<code>OSErr</code>	
<code>InitUtil()</code> ;	<code>OSErr</code>	
<code>KeyTrans()</code> ;	<code>long</code>	
<code>NGetTrapAddress()</code> ;	<code>long</code>	
<code>NSetTrapAddress()</code> ;	<code>void</code>	
<code>PtrAndHand()</code> ;	<code>OSErr</code>	
<code>PtrToHand()</code> ;	<code>OSErr</code>	

<i>Call</i>	<i>Return type</i>
PtrToXHand();	OSErr
ReadDateTime();	OSErr
relstring();	short
RelString();	short
Restart();	void
Secs2Date();	void
SetA5();	long
SetCurrentA5();	long
SetDateTime();	OSErr
SetTime();	void
SetTrapAddress();	void
SwapMMUMode();	void
SysBeep();	void
SysEnviron();	OSErr
UprString();	void
uprstring();	void
WriteParam();	OSErr

Utilities, Toolbox

The following structure and calls are available in `toolutils.h`:

Structure name

Int64Bit

<i>Call</i>	<i>Return type</i>
AngleFromSlope();	short
BitAnd();	long
BitClr();	void
BitNot();	long

<i>Call</i>	<i>Return type</i>
BitOr();	long
BitSet();	void
BitShift();	long
BitTst();	Boolean
BitXor();	long
deltapoint();	long
DeltaPoint();	long
FixMul();	Fixed
FixRatio();	Fixed
FixRound();	short
GetCursor();	CursHandle
GetIcon();	Handle
GetIndPattern();	void
getindstring();	void
GetIndString();	void
GetPattern();	PatHandle
GetPicture();	PicHandle
GetString();	StringHandle
HiWord();	short
LongMul();	void
LoWord();	short
Munger();	long
newstring();	StringHandle
NewString();	StringHandle
PackBits();	void
PlotIcon();	void
ScreenRes();	void
setstring();	void

<i>Call</i>	<i>Return type</i>
SetString();	void
shieldcursor();	void
ShieldCursor();	void
SlopeFromAngle();	Fixed
UnpackBits();	void

Vertical Retrace Manager

The following structure and calls are available in `retrace.h`:

Structure name

VBLTask

<i>Call</i>	<i>Return type</i>
AttachVBL();	OSErr
DoVBLTask();	OSErr
GetVBLQHdr();	QHdrPtr
SlotVInstall();	OSErr
SlotVRemove();	OSErr
VInstall();	OSErr
VRemove();	OSErr

Video Driver

The following structures are available in `video.h`, which has no calls:

Structure name

CSvidMsg	VDSetEntryRecord
VDEntryRecord	VDSettings
VDGrayRecord	VDSIZEInfo
VDPageInfo	VPBlock

Window Manager

The following structures and calls are available in `windows.h`:

Structure name

AuxWinRec	WindowRecord
CWindowRecord	WStateData
WinCTab	

Call

Return type

<code>BeginUpdate();</code>	void
<code>BringToFront();</code>	void
<code>CalcVis();</code>	void
<code>CalcVisBehind();</code>	void
<code>CheckUpdate();</code>	Boolean
<code>ClipAbove();</code>	void
<code>CloseWindow();</code>	void
<code>DisposeWindow();</code>	void
<code>draggrayrgn();</code>	long
<code>DragGrayRgn();</code>	long
<code>dragwindow();</code>	void
<code>DragWindow();</code>	void
<code>DrawGrowIcon();</code>	void
<code>DrawNew();</code>	void
<code>EndUpdate();</code>	void
<code>findwindow();</code>	short
<code>FindWindow();</code>	short
<code>FrontWindow();</code>	WindowPtr
<code>GetAuxWin();</code>	Boolean
<code>GetCWMgrPort();</code>	void
<code>GetGrayRgn();</code>	RgnHandle

<i>Call</i>	<i>Return type</i>
GetNewCWindow();	WindowPtr
GetNewWindow();	WindowPtr
GetWindowPic();	PicHandle
GetWMgrPort();	void
GetWRefCon();	long
getwttitle();	void
GetWTtitle();	void
GetWVariant();	short
growwindow();	long
GrowWindow();	long
HideWindow();	void
HiliteWindow();	void
InitWindows();	void
InvalRect();	void
InvalRgn();	void
MoveWindow();	void
newcwindow();	WindowPtr
NewCWindow();	WindowPtr
newwindow();	WindowPtr
NewWindow();	WindowPtr
PaintBehind();	void
PaintOne();	void
pinrect();	long
PinRect();	long
SaveOld();	void
SelectWindow();	void
SendBehind();	void
SetDeskCPat();	void

<i>Call</i>	<i>Return type</i>
SetWinColor();	void
SetWindowPic();	void
SetWRefCon();	void
setwtitle();	void
SetWTitle();	void
ShowHide();	void
ShowWindow();	void
SizeWindow();	void
trackbox();	Boolean
TrackBox();	Boolean
trackgoaway();	Boolean
TrackGoAway();	Boolean
ValidRect();	void
ValidRgn();	void
ZoomWindow();	void

Calls in alphabetical order

All calls in the preceding section are listed here in alphabetical order by name. This section also gives the return type and the name of the header file containing each call. A few calls are available in more than one header file. The major duplication is in `printing.h` and `printtraps.h`.

The names of two calls may differ only in case, one spelled as the name appears in *Inside Macintosh* (mixed case) and the other spelled in lowercase only. A call named in mixed case accepts Pascal-format strings and Pascal point-passing conventions. A call named in lowercase accepts input parameters in C format and converts them before passing them to the ROM routines, and converts string return values back to C format. For additional information on these differences, see the section “Differences in Language Conventions” in Chapter 4.

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
<code>*p2cstr();</code>	char	strings.h
<code>ActivatePalette();</code>	void	palettes.h
<code>AddComp();</code>	void	quickdraw.h
<code>AddDrive();</code>	void	files.h
<code>addpt();</code>	void	quickdraw.h
<code>AddPt();</code>	void	quickdraw.h
<code>AddResMenu();</code>	void	menus.h
<code>addressource();</code>	void	resources.h
<code>AddResource();</code>	void	resources.h
<code>AddSearch();</code>	void	quickdraw.h
<code>Alert();</code>	short	dialogs.h
<code>Allocate();</code>	OSErr	files.h
<code>AllocContig();</code>	OSErr	files.h
<code>AllocCursor();</code>	void	quickdraw.h
<code>AngleFromSlope();</code>	short	toolutils.h
<code>AnimateEntry();</code>	void	palettes.h
<code>AnimatePalette();</code>	void	palettes.h
<code>appendmenu();</code>	void	menus.h
<code>AppendMenu();</code>	void	menus.h
<code>ApplicZone();</code>	THz	memory.h
<code>aSndAddModifier();</code>	short	sm.h
<code>aSndControl();</code>	short	sm.h
<code>aSndDisposeChannel();</code>	short	sm.h
<code>aSndDoCommand();</code>	short	sm.h
<code>aSndDoImmediate();</code>	short	sm.h
<code>aSndNewChannel();</code>	short	sm.h
<code>aSndPlay();</code>	short	sm.h
<code>AttachVBL();</code>	OSErr	retrace.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
AUXCOFFLaunch();	pascal short	aux.h
AUXDispatch();	pascal long	aux.h
BackColor();	void	quickdraw.h
BackPat();	void	quickdraw.h
BackPixPat();	void	quickdraw.h
BeginUpdate();	void	windows.h
BitAnd();	long	toolutils.h
BitClr();	void	toolutils.h
BitNot();	long	toolutils.h
BitOr();	long	toolutils.h
BitSet();	void	toolutils.h
BitShift();	long	toolutils.h
BitTst();	Boolean	toolutils.h
BitXor();	long	toolutils.h
BlockMove();	void	memory.h
BringToFront();	void	windows.h
Button();	Boolean	events.h
c2pstr();	StringPtr	strings.h
CalcCMask();	void	quickdraw.h
CalcMask();	void	quickdraw.h
CalcMenuSize();	void	menus.h
CalcVis();	void	windows.h
CalcVisBehind();	void	windows.h
CatMove();	OSErr	files.h
CautionAlert();	short	dialogs.h
ChangedResource();	void	resources.h
Char2Pixel();	short	script.h
CharByte();	short	script.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
CharExtra();	void	quickdraw.h
CharType();	short	script.h
CharWidth();	short	quickdraw.h
CheckItem();	void	menus.h
CheckUpdate();	Boolean	windows.h
ClearMenuBar();	void	menus.h
ClipAbove();	void	windows.h
ClipRect();	void	quickdraw.h
CloseASD();	int	asd.h
CloseCPort();	void	quickdraw.h
CloseDeskAcc();	void	desk.h
CloseDialog();	void	dialogs.h
CloseDriver();	OSErr	devices.h
ClosePicture();	void	quickdraw.h
ClosePoly();	void	quickdraw.h
ClosePort();	void	quickdraw.h
CloseResFile();	void	resources.h
CloseRgn();	void	quickdraw.h
CloseWD();	OSErr	files.h
CloseWindow();	void	windows.h
ClrAppFiles();	void	segload.h
CMY2RGB();	void	picker.h
Color2Index();	long	quickdraw.h
ColorBit();	void	quickdraw.h
CompactMem();	Size	memory.h
Control();	OSErr	devices.h
CopyBits();	void	quickdraw.h
CopyMask();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
CopyPalette();	void	palettes.h
CopyPixMap();	void	quickdraw.h
CopyPixPat();	void	quickdraw.h
CopyRgn();	void	quickdraw.h
CouldAlert();	void	dialogs.h
CouldDialog();	void	dialogs.h
Count1Resources();	short	resources.h
Count1Types();	short	resources.h
CountAppFiles();	void	segload.h
CountMItems();	short	menus.h
CountResources();	short	resources.h
CountTypes();	short	resources.h
create();	OSErr	files.h
Create();	OSErr	files.h
createresfile();	void	resources.h
CreateResFile();	void	resources.h
CTab2Palette();	void	palettes.h
CurResFile();	short	resources.h
Date2Secs();	void	osutils.h
Debugger();	void	types.h
debugstr();	void	types.h
DebugStr();	void	types.h
DeferUserFn();	OSErr	vmcalls.h
Delay();	void	osutils.h
DelComp();	void	quickdraw.h
DeleteMenu();	void	menus.h
DelMCEntries();	void	menus.h
DelMenuItem();	void	menus.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
DelSearch();	void	quickdraw.h
deltapoint();	long	toolutils.h
DeltaPoint();	long	toolutils.h
Dequeue();	OSErr	osutils.h
DetachResource();	void	resources.h
DialogSelect();	Boolean	dialogs.h
dibadmound();	OSErr	diskinit.h
DIBadMount();	short	diskinit.h
DiffRgn();	void	quickdraw.h
DIFormat();	OSErr	diskinit.h
DILoad();	void	diskinit.h
DirCreate();	OSErr	files.h
DisableItem();	void	menus.h
DiskEject();	OSErr	disks.h
DispMCInfo();	void	menus.h
DisposCCursor();	void	quickdraw.h
DisposCIcon();	void	quickdraw.h
DisposCTable();	void	quickdraw.h
DisposDialog();	void	dialogs.h
DisposeControl();	void	controls.h
DisposeMenu();	void	menus.h
DisposePalette();	void	palettes.h
DisposeRgn();	void	quickdraw.h
DisposeWindow();	void	windows.h
DisposGDevice();	void	quickdraw.h
DisposHandle();	void	memory.h
DisposPixMap();	void	quickdraw.h
DisposPixPat();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
DisposPtr();	void	memory.h
DIUnload();	void	diskinit.h
DIVerify();	OSErr	diskinit.h
dizero();	OSErr	diskinit.h
DIZero();	OSErr	diskinit.h
DlgCopy();	void	dialogs.h
DlgCut();	void	dialogs.h
DlgDelete();	void	dialogs.h
DlgPaste();	void	dialogs.h
DoVBLTask();	OSErr	retrace.h
dragcontrol();	void	controls.h
DragControl();	void	controls.h
draggrayrgn();	long	windows.h
DragGrayRgn();	long	windows.h
dragwindow();	void	windows.h
DragWindow();	void	windows.h
DrawlControl();	void	controls.h
DrawChar();	void	quickdraw.h
DrawControls();	void	controls.h
DrawDialog();	void	dialogs.h
DrawGrowIcon();	void	windows.h
DrawJust();	void	script.h
DrawMenuBar();	void	menus.h
DrawNew();	void	windows.h
DrawPicture();	void	quickdraw.h
drawstring();	void	quickdraw.h
DrawString();	void	quickdraw.h
DrawText();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
DriveStatus();	OSErr	disks.h
DTInstall();	OSErr	dtask.h
DTInstall();	OSErr	osutils.h
eject();	OSErr	files.h
Eject();	OSErr	files.h
EmptyHandle();	void	memory.h
EmptyRect();	Boolean	quickdraw.h
EmptyRgn();	Boolean	quickdraw.h
EnableItem();	void	menus.h
EndUpdate();	void	windows.h
Enqueue();	void	osutils.h
Environ();	void	osutils.h
equalpt();	Boolean	quickdraw.h
EqualPt();	Boolean	quickdraw.h
EqualRect();	Boolean	quickdraw.h
EqualRgn();	Boolean	quickdraw.h
equalstring();	Boolean	osutils.h
EqualString();	Boolean	osutils.h
EraseArc();	void	quickdraw.h
EraseOval();	void	quickdraw.h
ErasePoly();	void	quickdraw.h
EraseRect();	void	quickdraw.h
EraseRgn();	void	quickdraw.h
EraseRoundRect();	void	quickdraw.h
ErrorSound();	void	dialogs.h
EventAvail();	Boolean	events.h
ExitToShell();	void	segload.h
FillArc();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
FillCArc();	void	quickdraw.h
FillCOval();	void	quickdraw.h
FillCPoly();	void	quickdraw.h
FillCRect();	void	quickdraw.h
FillCRgn();	void	quickdraw.h
FillCRoundRect();	void	quickdraw.h
FillOval();	void	quickdraw.h
FillPoly();	void	quickdraw.h
FillRect();	void	quickdraw.h
FillRgn();	void	quickdraw.h
FillRoundRect();	void	quickdraw.h
FindBlock();	struct FindBlockStatus	script.h
findcontrol();	short	controls.h
FindControl();	short	controls.h
findditem();	short	dialogs.h
FindDItem();	short	dialogs.h
findwindow();	short	windows.h
FindWindow();	short	windows.h
FindWord();	void	script.h
FInitQueue();	void	files.h
Fix2SmallFract();	SmallFract	picker.h
FixMul();	Fixed	toolutils.h
FixRatio();	Fixed	toolutils.h
FixRound();	short	toolutils.h
FlashMenuBar();	void	menus.h
FlushEvents();	void	osevents.h
flushvol();	OSErr	files.h
FlushVol();	OSErr	files.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
FMSwapFont();	FMOutPtr	fonts.h
Font2Script();	short	script.h
FontMetrics();	void	fonts.h
FontScript();	short	script.h
ForeColor();	void	quickdraw.h
Form2Str();	FormatStatus	script.h
FormStr2X();	FormatStatus	script.h
FormX2Str();	FormatStatus	script.h
FrameArc();	void	quickdraw.h
FrameOval();	void	quickdraw.h
FramePoly();	void	quickdraw.h
FrameRect();	void	quickdraw.h
FrameRgn();	void	quickdraw.h
FrameRoundRect();	void	quickdraw.h
FreeAlert();	void	dialogs.h
FreeDialog();	void	dialogs.h
FreeMem();	long	memory.h
FrontWindow();	WindowPtr	windows.h
FSClose();	OSErr	files.h
fsdelete();	OSErr	files.h
FSDelete();	OSErr	files.h
fsopen();	OSErr	files.h
FSOpen();	OSErr	files.h
FSRead();	OSErr	files.h
fsrename();	OSErr	files.h
FSWrite();	OSErr	files.h
Get1IndResource();	Handle	resources.h
Get1IndType();	void	resources.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
get1namedresource();	Handle	resources.h
Get1NamedResource();	Handle	resources.h
Get1Resource();	Handle	resources.h
GetAlrtStage();	short	dialogs.h
GetAppFiles();	void	segload.h
GetAppFont();	short	script.h
GetApplLimit();	Ptr	memory.h
getappparms();	void	segload.h
GetAppParms();	void	segload.h
GetAuxCtl();	Boolean	controls.h
GetAuxWin();	Boolean	windows.h
GetBackColor();	void	quickdraw.h
GetCaretTime();	unsigned long	events.h
GetCCursor();	CCrsrHandle	quickdraw.h
GetCIcon();	CIconHandle	quickdraw.h
GetClip();	void	quickdraw.h
GetColor();	Boolean	picker.h
GetCPixel();	void	quickdraw.h
GetCRefCon();	long	controls.h
GetCTable();	CTabHandle	quickdraw.h
getctitle();	void	controls.h
GetCTitle();	void	controls.h
GetCtlAction();	ProcPtr	controls.h
GetCtlMax();	short	controls.h
GetCtlMin();	short	controls.h
GetCtlValue();	short	controls.h
GetCTSeed();	long	quickdraw.h
GetCurrentProcess();	OSErr	processes.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
GetCursor();	CursHandle	toolutils.h
GetCVariant();	short	controls.h
GetCWMgrPort();	void	windows.h
GetDateTime();	void	osutils.h
GetDblTime();	unsigned long	events.h
GetDCtlEntry();	DCtlHandle	devices.h
GetDefFontSize();	short	script.h
GetDeviceList();	GDHandle	quickdraw.h
GetDItem();	void	dialogs.h
GetDrvQHdr();	QHdrPtr	files.h
GetEntryColor();	void	palettes.h
GetEntryUsage();	void	palettes.h
GetEnvirons();	long	script.h
GetEOF();	OSErr	files.h
GetEvQHdr();	QHdrPtr	osevents.h
getfinfo();	OSErr	files.h
GetFInfo();	OSErr	files.h
getfnum();	void	fonts.h
GetFNum();	void	fonts.h
GetFontInfo();	void	quickdraw.h
getfontname();	void	fonts.h
GetFontName();	void	fonts.h
GetForeColor();	void	quickdraw.h
GetFormatOrder();	void	script.h
GetFPos();	OSErr	files.h
GetFrontProcess	OSErr	processes.h
GetFSQHdr();	QHdrPtr	files.h
GetGDevice();	GDHandle	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
GetGrayRgn();	RgnHandle	windows.h
GetHandleSize();	Size	memory.h
GetIcon();	Handle	toolutils.h
GetIndPattern();	void	toolutils.h
GetIndResource();	Handle	resources.h
getindstring();	void	toolutils.h
GetIndString();	void	toolutils.h
GetIndType();	void	resources.h
getitem();	void	menus.h
GetItem();	void	menus.h
GetItemCmd();	void	menus.h
GetItemIcon();	void	menus.h
GetItemMark();	void	menus.h
GetItemStyle();	void	menus.h
getitext();	void	dialogs.h
GetIText();	void	dialogs.h
GetKeys();	void	events.h
GetMainDevice();	GDHandle	quickdraw.h
GetMaskTable();	Ptr	quickdraw.h
GetMaxDevice();	GDHandle	quickdraw.h
GetMBarHeight();	short	script.h
GetMCEntry();	MCEntryPtr	menus.h
GetMCInfo();	MCTableHandle	menus.h
GetMenu();	MenuHandle	menus.h
GetMenuBar();	Handle	menus.h
GetMHandle();	MenuHandle	menus.h
GetMMUMode();	char	osutils.h
GetMouse();	void	events.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
getnamedresource();	Handle	resources.h
GetNamedResource();	Handle	resources.h
GetNewControl();	ControlHandle	controls.h
GetNewCWindow();	WindowPtr	windows.h
GetNewDialog();	DialogPtr	dialogs.h
GetNewMBar();	Handle	menus.h
GetNewPalette();	PaletteHandle	palettes.h
GetNewWindow();	WindowPtr	windows.h
GetNextDevice();	GDHandle	quickdraw.h
GetNextEvent();	Boolean	events.h
GetNextProcess	OSErr	processes.h
GetOSEvent();	Boolean	osevents.h
GetPalette();	PaletteHandle	palettes.h
GetPattern();	PatHandle	toolutils.h
GetPen();	void	quickdraw.h
GetPenState();	void	quickdraw.h
GetPhysical()	OSErr	vmcalls.h
GetPicture();	PicHandle	toolutils.h
GetPixel();	Boolean	quickdraw.h
GetPixPat();	PixPatHandle	quickdraw.h
GetPort();	void	quickdraw.h
GetProcessInformation();	OSErr	processes.h
GetPtrSize();	Size	memory.h
GetResAttrs();	short	resources.h
GetResFileAttrs();	short	resources.h
getresinfo();	void	resources.h
GetResInfo();	void	resources.h
GetResource();	Handle	resources.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
GetScrap();	long	scrap.h
GetScript();	long	script.h
GetString();	StringHandle	toolutils.h
GetStylHandle();	TEStyleHandle	textedit.h
GetStylScrap();	StScrpHandle	textedit.h
GetSubTable();	void	quickdraw.h
GetSysFont();	short	script.h
GetSysJust();	short	script.h
GetSysPPtr();	SysPPtr	osutils.h
GetTime();	void	osutils.h
GetTrapAddress();	long	osutils.h
GetVBLQHdr();	QHdrPtr	retrace.h
GetVCBQHdr();	QHdrPtr	files.h
getvinfo();	OSErr	files.h
GetVInfo();	OSErr	files.h
getvol();	OSErr	files.h
GetVol();	OSErr	files.h
GetVRefNum();	OSErr	files.h
GetWDInfo();	OSErr	files.h
GetWindowPic();	PicHandle	windows.h
GetWMgrPort();	void	windows.h
GetWRefCon();	long	windows.h
getwtitle();	void	windows.h
GetWTtitle();	void	windows.h
GetWVariant();	short	windows.h
GetZone();	THz	memory.h
GlobalToLocal();	void	quickdraw.h
GrafDevice();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
growwindow();	long	windows.h
GrowWindow();	long	windows.h
GZSaveHnd();	Handle	memory.h
HandAndHand();	OSErr	osutils.h
HandleZone();	THz	memory.h
HandToHand();	OSErr	osutils.h
HClrRBit();	void	memory.h
HCreate();	OSErr	files.h
HCreateResFile();	void	resources.h
HDelete();	OSErr	files.h
HGetFInfo();	OSErr	files.h
HGetState();	short	memory.h
HGetVol();	OSErr	files.h
HideControl();	void	controls.h
HideCursor();	void	quickdraw.h
HideDItem();	void	dialogs.h
HidePen();	void	quickdraw.h
HideWindow();	void	windows.h
HiliteColor();	void	quickdraw.h
HiliteControl();	void	controls.h
HiliteMenu();	void	menus.h
HiliteText();	void	script.h
HiliteWindow();	void	windows.h
HiWord();	short	toolutils.h
HLock();	void	memory.h
HNoPurge();	void	memory.h
HoldMemory();	OSErr	vmcalls.h
HomeResFile();	short	resources.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
HOpen();	OSErr	files.h
HOpenResFile();	short	resources.h
HOpenRF();	OSErr	files.h
HPurge();	void	memory.h
HRename();	OSErr	files.h
HRstFLock();	OSErr	files.h
HSetFInfo();	OSErr	files.h
HSetFLock();	OSErr	files.h
HSetRBit();	void	memory.h
HSetState();	void	memory.h
HSetVol();	OSErr	files.h
HSL2RGB();	void	picker.h
HSV2RGB();	void	picker.h
HUnlock();	void	memory.h
Index2Color();	void	quickdraw.h
InfoScrap();	PScrapStuff	scrap.h
InitAllPacks();	void	packages.h
InitApplZone();	void	memory.h
InitCPort();	void	quickdraw.h
InitCursor();	void	quickdraw.h
InitDateCache();	OSErr	script.h
InitDialogs();	void	dialogs.h
InitFonts();	void	fonts.h
InitGDevice();	void	quickdraw.h
InitGraf();	void	quickdraw.h
InitMenus();	void	menus.h
InitPack();	void	packages.h
InitPalettes();	void	palettes.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
InitPort();	void	quickdraw.h
InitProcMenu();	void	menus.h
InitResources();	short	resources.h
InitSDeclMgr();	OSErr	slots.h
InitUtil();	OSErr	osutils.h
InitWindows();	void	windows.h
InitZone();	void	memory.h
InsertMenu();	void	menus.h
InsertResMenu();	void	menus.h
InsetRect();	void	quickdraw.h
InsetRgn();	void	quickdraw.h
insmenuItem();	void	menus.h
InsMenuItem();	void	menus.h
InsTime();	void	timer.h
IntlScript();	short	script.h
InvalRect();	void	windows.h
InvalRgn();	void	windows.h
InvertArc();	void	quickdraw.h
InvertColor();	void	quickdraw.h
InvertOval();	void	quickdraw.h
InvertPoly();	void	quickdraw.h
InvertRect();	void	quickdraw.h
InvertRgn();	void	quickdraw.h
InvertRoundRect();	void	quickdraw.h
IsDialogEvent();	Boolean	dialogs.h
iucompstring();	short	packages.h
IUCompString();	short	packages.h
iudatepstring();	void	packages.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
IUDatePString();	void	packages.h
iudatesting();	void	packages.h
IUDateString();	void	packages.h
iuequalstring();	short	packages.h
IUEqualString();	short	packages.h
IUGetIntl();	Handle	packages.h
IULDateString();	void	script.h
IULTimeString();	void	script.h
IUMagIDString();	short	packages.h
IUMagString();	short	packages.h
IUMetric();	Boolean	packages.h
IUSetIntl();	void	packages.h
iutimepstring();	void	packages.h
IUTimePString();	void	packages.h
iutimestring();	void	packages.h
IUTimeString();	void	packages.h
KeyScript();	void	script.h
KeyTrans();	long	osutils.h
KillControls();	void	controls.h
KillIO();	OSErr	devices.h
KillPicture();	void	quickdraw.h
KillPoly();	void	quickdraw.h
LActivate();	void	lists.h
LAddColumn();	short	lists.h
LAddRow();	short	lists.h
LAddToCell();	void	lists.h
LaunchApplication	OSErr	processes.h
LaunchDeskAccessory	OSErr	processes.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
LAutoScroll();	void	lists.h
lcellsize();	void	lists.h
LCellSize();	void	lists.h
lclick();	Boolean	lists.h
LClick();	Boolean	lists.h
LClrCell();	void	lists.h
LDelColumn();	void	lists.h
LDelRow();	void	lists.h
LDispose();	void	lists.h
LDoDraw();	void	lists.h
ldraw();	void	lists.h
LDraw();	void	lists.h
LFind();	void	lists.h
LGetCell();	void	lists.h
LGetSelect();	Boolean	lists.h
Line();	void	quickdraw.h
LineBreak();	LineBreakCode	script.h
LineTo();	void	quickdraw.h
LLastClick();	Cell	lists.h
lnew();	ListHandle	lists.h
LNew();	ListHandle	lists.h
LNextCell();	Boolean	lists.h
LoadResource();	void	resources.h
LoadScrap();	long	scrap.h
LocalToGlobal();	void	quickdraw.h
LockMemory();	OSErr	vmcalls.h
LockMemoryContiguous();	OSErr	vmcalls.h
LongDate2Secs();	void	script.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
LongMul();	void	toolutils.h
LongSecs2Date();	void	script.h
LoWord();	short	toolutils.h
LRect();	void	lists.h
LScroll();	void	lists.h
LSearch();	Boolean	lists.h
LSetCell();	void	lists.h
LSetSelect();	void	lists.h
LSize();	void	lists.h
LUpdate();	void	lists.h
LwrText();	void	script.h
MakeITable();	void	quickdraw.h
MakeRGBPat();	void	quickdraw.h
MapPoly();	void	quickdraw.h
MapPt();	void	quickdraw.h
MapRect();	void	quickdraw.h
MapRgn();	void	quickdraw.h
MaxApplZone();	void	memory.h
MaxBlock();	long	memory.h
MaxMem();	Size	memory.h
MaxSizeRsrc();	long	resources.h
MeasureJust();	void	script.h
MeasureText();	void	quickdraw.h
MemError();	OSErr	memory.h
MenuChoice();	long	menus.h
MenuKey();	long	menus.h
menuselect();	long	menus.h
MenuSelect();	long	menus.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
MFFreeMem();	long	memory.h
MFMaxMem();	Size	memory.h
MFTempDisposHandle();	void	memory.h
MFTempHLock();	void	memory.h
MFTempHUnlock();	void	memory.h
MFTempNewHandle();	Handle	memory.h
MFTopMem();	Ptr	memory.h
ModalDialog();	void	dialogs.h
MoreMasters();	void	memory.h
Move();	void	quickdraw.h
MoveControl();	void	controls.h
MoveHHi();	void	memory.h
MovePortTo();	void	quickdraw.h
MoveTo();	void	quickdraw.h
MoveWindow();	void	windows.h
mrattr();	short	aux_rsrc.h
mrclose();	int	aux_rsrc.h
mrget();	Resource	aux_rsrc.h
mrgetnamed();	Resource	aux_rsrc.h
mrinfo();	int	aux_rsrc.h
mropen();	ResHandle	aux_rsrc.h
mrrel();	void	aux_rsrc.h
Munger();	long	toolutils.h
newcdialog();	DialogPtr	dialogs.h
NewCDialog();	DialogPtr	dialogs.h
newcontrol();	ControlHandle	controls.h
NewControl();	ControlHandle	controls.h
newcwindow();	WindowPtr	windows.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
NewCWindow();	WindowPtr	windows.h
newdialog();	DialogPtr	dialogs.h
NewDialog();	DialogPtr	dialogs.h
NewEmptyHandle();	Handle	memory.h
NewGDevice();	GDHandle	quickdraw.h
NewHandle();	Handle	memory.h
newmenu();	MenuHandle	menus.h
NewMenu();	MenuHandle	menus.h
NewPalette();	PaletteHandle	palettes.h
NewPixMap();	PixMapHandle	quickdraw.h
NewPixPat();	PixPatHandle	quickdraw.h
NewPtr();	Ptr	memory.h
NewRgn();	RgnHandle	quickdraw.h
newstring();	StringHandle	toolutils.h
NewString();	StringHandle	toolutils.h
newwindow();	WindowPtr	windows.h
NewWindow();	WindowPtr	windows.h
NGetTrapAddress();	long	osutils.h
NMInstall();	OSErr	notify.h
NMremove();	OSErr	notify.h
NoteAlert();	short	dialogs.h
NSetPalette();	void	palettes.h
NSetTrapAddress();	void	osutils.h
numtostring();	void	packages.h
NumToString();	void	packages.h
ObscureCursor();	void	quickdraw.h
OffsetPoly();	void	quickdraw.h
OffsetRect();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
OffsetRgn();	void	quickdraw.h
OpColor();	void	quickdraw.h
OpenASD();	FileHandle	asd.h
OpenCPort();	void	quickdraw.h
opendeskacc();	short	desk.h
OpenDeskAcc();	short	desk.h
opendriver();	OSErr	devices.h
OpenDriver();	OSErr	devices.h
OpenPicture();	PicHandle	quickdraw.h
OpenPoly();	PolyHandle	quickdraw.h
OpenPort();	void	quickdraw.h
openresfile();	short	resources.h
OpenResFile();	short	resources.h
openrf();	OSErr	files.h
OpenRF();	OSErr	files.h
openrfperm();	short	resources.h
OpenRFPerm();	short	resources.h
OpenRgn();	void	quickdraw.h
OpenSlot();	OSErr	slots.h
OpenWD();	OSErr	files.h
OSEventAvail();	Boolean	osevents.h
PackBits();	void	toolutils.h
PaintArc();	void	quickdraw.h
PaintBehind();	void	windows.h
PaintOne();	void	windows.h
PaintOval();	void	quickdraw.h
PaintPoly();	void	quickdraw.h
PaintRect();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
PaintRgn();	void	quickdraw.h
PaintRoundRect();	void	quickdraw.h
Palette2CTab();	void	palettes.h
paramtext();	void	dialogs.h
ParamText();	void	dialogs.h
ParseTable();	Boolean	script.h
PBAllocate();	OSErr	files.h
PBAllocContig();	OSErr	files.h
PBCatMove();	OSErr	files.h
PBClose();	OSErr	files.h
PBCloseWD();	OSErr	files.h
PBControl();	OSErr	devices.h
PBCreate();	OSErr	files.h
PBDelete();	OSErr	files.h
PBDirCreate();	OSErr	files.h
PBEject();	OSErr	files.h
PBFlushFile();	OSErr	files.h
PBFlushVol();	OSErr	files.h
PBGetCatInfo();	OSErr	files.h
PBGetEOF();	OSErr	files.h
PBGetFCBInfo();	OSErr	files.h
PBGetFInfo();	OSErr	files.h
PBGetFPos();	OSErr	files.h
PBGetVInfo();	OSErr	files.h
PBGetVol();	OSErr	files.h
PBGetWDInfo();	OSErr	files.h
PBHCopyFile();	OSErr	files.h
PBHCreate();	OSErr	files.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
PBHDelete();	OSErr	files.h
PBHGetDirAccess();	OSErr	files.h
PBHGetFInfo();	OSErr	files.h
PBHGetLogInInfo();	OSErr	files.h
PBHGetVInfo();	OSErr	files.h
PBHGetVol();	OSErr	files.h
PBHGetVolParms();	OSErr	files.h
PBHMapID();	OSErr	files.h
PBHMapName();	OSErr	files.h
PBHMoveRename();	OSErr	files.h
PBHOpen();	OSErr	files.h
PBHOpenDeny();	OSErr	files.h
PBHOpenRF();	OSErr	files.h
PBHOpenRFDeny();	OSErr	files.h
PBHRename();	OSErr	files.h
PBHRstFLock();	OSErr	files.h
PBHSetDirAccess();	OSErr	files.h
PBHSetFInfo();	OSErr	files.h
PBHSetFLock();	OSErr	files.h
PBHSetVol();	OSErr	files.h
PBKillIO();	OSErr	devices.h
PBLockRange();	OSErr	files.h
PBMountVol();	OSErr	files.h
PBOffLine();	OSErr	files.h
PBOpen();	OSErr	files.h
PBOpenRF();	OSErr	files.h
PBOpenWD();	OSErr	files.h
PBRead();	OSErr	files.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
PBRename();	OSErr	files.h
PBRstFLock();	OSErr	files.h
PBSetCatInfo();	OSErr	files.h
PBSetEOF();	OSErr	files.h
PBSetFInfo();	OSErr	files.h
PBSetFLock();	OSErr	files.h
PBSetFPos();	OSErr	files.h
PBSetFVers();	OSErr	files.h
PBSetVInfo();	OSErr	files.h
PBSetVol();	OSErr	files.h
PBStatus();	OSErr	devices.h
PBUnlockRange();	OSErr	files.h
PBUnmountVol();	OSErr	files.h
PBWrite();	OSErr	files.h
PenMode();	void	quickdraw.h
PenNormal();	void	quickdraw.h
PenPat();	void	quickdraw.h
PenPixPat();	void	quickdraw.h
PenSize();	void	quickdraw.h
PicComment();	void	quickdraw.h
pinrect();	long	windows.h
PinRect();	long	windows.h
Pixel2Char();	short	script.h
PlotCIcon();	void	quickdraw.h
PlotIcon();	void	toolutils.h
PmBackColor();	void	palettes.h
PmForeColor();	void	palettes.h
PopUpMenuSelect();	long	menus.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
PortionText ();	Fixed	script.h
PortSize ();	void	quickdraw.h
PostEvent ();	OSErr	osevents.h
PPostEvent ();	OSErr	osevents.h
PrClose ();	void	printing.h
PrClose ();	void	printtraps.h
PrCloseDoc ();	void	printing.h
PrCloseDoc ();	void	printtraps.h
PrClosePage ();	void	printing.h
PrClosePage ();	void	printtraps.h
PrCtlCall ();	void	printing.h
PrCtlCall ();	void	printtraps.h
PrDlgMain ();	Boolean	printing.h
PrDlgMain ();	Boolean	printtraps.h
PrDrvrClose ();	void	printing.h
PrDrvrClose ();	void	printtraps.h
PrDrvrDCE ();	Handle	printing.h
PrDrvrDCE ();	Handle	printtraps.h
PrDrvrOpen ();	void	printing.h
PrDrvrOpen ();	void	printtraps.h
PrDrvrVers ();	short	printing.h
PrDrvrVers ();	short	printtraps.h
PrError ();	short	printing.h
PrError ();	short	printtraps.h
PrGeneral ();	void	printing.h
PrGeneral ();	void	printtraps.h
PrimeTime ();	void	timer.h
PrintDefault ();	void	printing.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
PrintDefault();	void	printtraps.h
PrJobDialog();	Boolean	printing.h
PrJobDialog();	Boolean	printtraps.h
PrJobInit();	TPPrDlg	printing.h
PrJobInit();	TPPrDlg	printtraps.h
PrJobMerge();	void	printing.h
PrJobMerge();	void	printtraps.h
PrNoPurge();	void	printing.h
PrNoPurge();	void	printtraps.h
PrOpen();	void	printing.h
PrOpen();	void	printtraps.h
PrOpenDoc();	TPPrPort	printing.h
PrOpenDoc();	TPPrPort	printtraps.h
PrOpenPage();	void	printing.h
PrOpenPage();	void	printtraps.h
ProtectEntry();	void	quickdraw.h
PrPicFile();	void	printing.h
PrPicFile();	void	printtraps.h
PrPurge();	void	printing.h
PrPurge();	void	printtraps.h
PrSetError();	void	printing.h
PrSetError();	void	printtraps.h
PrStlDialog();	Boolean	printing.h
PrStlDialog();	Boolean	printtraps.h
PrStlInit();	TPPrDlg	printing.h
PrStlInit();	TPPrDlg	printtraps.h
PrValidate();	Boolean	printing.h
PrValidate();	Boolean	printtraps.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
pt2rect();	void	quickdraw.h
Pt2Rect();	void	quickdraw.h
ptinrect();	Boolean	quickdraw.h
PtInRect();	Boolean	quickdraw.h
ptinrgn();	Boolean	quickdraw.h
PtInRgn();	Boolean	quickdraw.h
PtrAndHand();	OSErr	osutils.h
PtrToHand();	OSErr	osutils.h
PtrToXHand();	OSErr	osutils.h
PtrZone();	THz	memory.h
pttoangle();	void	quickdraw.h
PtToAngle();	void	quickdraw.h
PurgeMem();	void	memory.h
PurgeSpace();	void	memory.h
PutScrap();	long	scrap.h
QDError();	short	quickdraw.h
RamSDClose();	void	serial.h
RamSDOpen();	OSErr	serial.h
Random();	short	quickdraw.h
ReadASD();	long	asd.h
ReadDateTime();	OSErr	osutils.h
ReadLocation();	void	script.h
RealColor();	Boolean	quickdraw.h
RealFont();	Boolean	fonts.h
ReallocHandle();	void	memory.h
RecoverHandle();	Handle	memory.h
RectInRgn();	Boolean	quickdraw.h
RectRgn();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
ReleaseResource();	void	resources.h
relstring();	short	osutils.h
RelString();	short	osutils.h
Rename();	OSErr	files.h
ResError();	short	resources.h
ReserveEntry();	void	quickdraw.h
ResetAlrtStage();	void	dialogs.h
ResrvMem();	void	memory.h
Restart();	void	osutils.h
RestoreEntries();	void	quickdraw.h
RGB2CMY();	void	picker.h
RGB2HSL();	void	picker.h
RGB2HSV();	void	picker.h
RGBBackColor();	void	quickdraw.h
RGBForeColor();	void	quickdraw.h
RGetResource();	Handle	resources.h
RmveResource();	void	resources.h
RmvTime();	void	timer.h
RsrcMapEntry();	long	resources.h
RsrcZoneInit();	void	resources.h
rstfLock();	OSErr	files.h
RstFLock();	OSErr	files.h
SameProcess	OSErr	processes.h
SaveEntries();	void	quickdraw.h
SaveOld();	void	windows.h
SCalcSPointer();	OSErr	slots.h
SCalcStep();	OSErr	slots.h
ScalePt();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
SCardChanged();	OSErr	slots.h
SckCardStat();	OSErr	slots.h
ScreenRes();	void	toolutils.h
ScrollRect();	void	quickdraw.h
SDeleteSRTRec();	OSErr	slots.h
Secs2Date();	void	osutils.h
SectRect();	Boolean	quickdraw.h
SectRgn();	void	quickdraw.h
SeedCFill();	void	quickdraw.h
SeedFill();	void	quickdraw.h
SeekASD();	long	asd.h
SelectWindow();	void	windows.h
SelIText();	void	dialogs.h
SendBehind();	void	windows.h
SerClrBrk();	OSErr	serial.h
SerGetBuf();	OSErr	serial.h
SerHShake();	OSErr	serial.h
SerReset();	OSErr	serial.h
SerSetBrk();	OSErr	serial.h
SerSetBuf();	OSErr	serial.h
SerStatus();	OSErr	serial.h
SetA5();	long	osutils.h
SetApplBase();	void	memory.h
SetApplLimit();	void	memory.h
SetCCursor();	void	quickdraw.h
SetChooserAlert();	Boolean	devices.h
SetClientID();	void	quickdraw.h
SetClikLoop();	void	textedit.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
SetClip();	void	quickdraw.h
SetCPixel();	void	quickdraw.h
SetCRefCon();	void	controls.h
setctitle();	void	controls.h
SetCTitle();	void	controls.h
SetCtlAction();	void	controls.h
SetCtlColor();	void	controls.h
SetCtlMax();	void	controls.h
SetCtlMin();	void	controls.h
SetCtlValue();	void	controls.h
SetCurrentA5();	long	osutils.h
SetCursor();	void	quickdraw.h
SetDAFont();	void	dialogs.h
SetDateTime();	OSErr	osutils.h
SetDeskCPat();	void	windows.h
SetDeviceAttribute();	void	quickdraw.h
SetDItem();	void	dialogs.h
SetEmptyRgn();	void	quickdraw.h
SetEntries();	void	quickdraw.h
SetEntryColor();	void	palettes.h
SetEntryUsage();	void	palettes.h
SetEnvirons();	OSErr	script.h
SetEOF();	OSErr	files.h
SetEventMask();	void	osevents.h
setfinfo();	OSErr	files.h
SetFInfo();	OSErr	files.h
setflock();	OSErr	files.h
SetFLock();	OSErr	files.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
SetFontLock();	void	fonts.h
SetFPos();	OSErr	files.h
SetFractEnable();	void	fonts.h
SetFrontProcess	OSErr	processes.h
SetFScaleDisable();	void	fonts.h
SetGDevice();	void	quickdraw.h
SetGrowZone();	void	memory.h
SetHandleSize();	void	memory.h
setItem();	void	menus.h
SetItem();	void	menus.h
SetItemCmd();	void	menus.h
SetItemIcon();	void	menus.h
SetItemMark();	void	menus.h
SetItemStyle();	void	menus.h
setitext();	void	dialogs.h
SetIText();	void	dialogs.h
SetMCEntries();	void	menus.h
SetMCInfo();	void	menus.h
SetMenuBar();	void	menus.h
SetMenuFlash();	void	menus.h
SetOrigin();	void	quickdraw.h
SetPalette();	void	palettes.h
SetPenState();	void	quickdraw.h
SetPort();	void	quickdraw.h
SetPortBits();	void	quickdraw.h
SetPortPix();	void	quickdraw.h
SetPt();	void	quickdraw.h
SetPtrSize();	void	memory.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
SetRect();	void	quickdraw.h
SetRectRgn();	void	quickdraw.h
SetResAttrs();	void	resources.h
SetResFileAttrs();	void	resources.h
setresinfo();	void	resources.h
SetResInfo();	void	resources.h
SetResLoad();	void	resources.h
SetResPurge();	void	resources.h
SetScript();	OSErr	script.h
SetStdCProcs();	void	quickdraw.h
SetStdProcs();	void	quickdraw.h
setstring();	void	toolutils.h
SetString();	void	toolutils.h
SetStylHandle();	void	textedit.h
SetStylScrap();	void	textedit.h
SetSysJust();	void	script.h
SetTagBuffer();	OSErr	disks.h
SetTime();	void	osutils.h
SetTrapAddress();	void	osutils.h
SetupAIFFHeader	OSErr	soundinput.h
SetupSndHeader	OSErr	soundinput.h
setvol();	OSErr	files.h
SetVol();	OSErr	files.h
SetWinColor();	void	windows.h
SetWindowPic();	void	windows.h
SetWordBreak();	void	textedit.h
SetWRefCon();	void	windows.h
setwttitle();	void	windows.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
SetWTitle();	void	windows.h
SetZone();	void	memory.h
SExec();	OSErr	slots.h
sfgetFile();	void	packages.h
SFGetFile();	void	packages.h
SFindBigDevBase();	OSErr	slots.h
SFindDevBase();	OSErr	slots.h
SFindSInfoRecPtr();	OSErr	slots.h
SFindSRsrcPtr();	OSErr	slots.h
SFindStruct();	OSErr	slots.h
sfpgetFile();	void	packages.h
SFPGetFile();	void	packages.h
sfpputfile();	void	packages.h
SFPPutFile();	void	packages.h
sfputfile();	void	packages.h
SFPutFile();	void	packages.h
SGetBlock();	OSErr	slots.h
SGetCString();	OSErr	slots.h
SGetDriver();	OSErr	slots.h
SGetsRsrc();	OSErr	slots.h
SGetsRsrcInfo();	OSErr	slots.h
SGetTypesRsrc();	OSErr	slots.h
shieldcursor();	void	toolutils.h
ShieldCursor();	void	toolutils.h
ShowControl();	void	controls.h
ShowCursor();	void	quickdraw.h
ShowDItem();	void	dialogs.h
ShowHide();	void	windows.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
ShowPen();	void	quickdraw.h
ShowWindow();	void	windows.h
ShutDwnInstall();	void	shutdown.h
ShutDwnRemove();	void	shutdown.h
SInitPRAMRecs();	OSErr	slots.h
SInitSRsrcTable();	OSErr	slots.h
SInsertSRTRec();	OSErr	slots.h
SIntInstall();	OSErr	slots.h
SIntRemove();	OSErr	slots.h
SizeControl();	void	controls.h
SizeResource();	long	resources.h
SizeWindow();	void	windows.h
SlopeFromAngle();	Fixed	toolutils.h
SlotVInstall();	OSErr	retrace.h
SlotVRemove();	OSErr	retrace.h
SmallFract2Fix();	Fixed	picker.h
SndRecord	OSErr	soundinput.h
SndRecordToFile	OSErr	soundinput.h
SNextSRsrc();	OSErr	slots.h
SNextTypeSRsrc();	OSErr	slots.h
SOffsetData();	OSErr	slots.h
SpaceExtra();	void	quickdraw.h
SPBBytesToMilliseconds	OSErr	soundinput.h
SPBCloseDevice	OSErr	soundinput.h
SPBGetDefaultDevice	OSErr	soundinputpriv.h
SPBGetDeviceInfo	OSErr	soundinput.h
SPBGetIndexedDevice	OSErr	soundinput.h
SPBGetRecordingStatus	OSErr	soundinput.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
SPBMillisecondsToBytes	OSErr	soundinput.h
SPBOpenDevice	OSErr	soundinput.h
SPBPauseRecording	OSErr	soundinput.h
SPBRecord	OSErr	soundinput.h
SPBRecordToFile	OSErr	soundinput.h
SPBResumeRecording	OSErr	soundinput.h
SPBSetDefaultDevice	OSErr	soundinputpriv.h
SPBSetDeviceInfo	OSErr	soundinput.h
SPBSignInDevice	OSErr	soundinput.h
SPBSignOutDevice	OSErr	soundinput.h
SPBStopRecording	OSErr	soundinput.h
SPrimaryInit();	OSErr	slots.h
SPtrToSlot();	OSErr	slots.h
SPutPRAMRec();	OSErr	slots.h
SReadByte();	OSErr	slots.h
SReadDrvName();	OSErr	slots.h
SReadFHeader();	OSErr	slots.h
SReadInfo();	OSErr	slots.h
SReadLong();	OSErr	slots.h
SReadPBSize();	OSErr	slots.h
SReadPRAMRec();	OSErr	slots.h
SReadStruct();	OSErr	slots.h
SReadWord();	OSErr	slots.h
SRsrcInfo();	OSErr	slots.h
SSearchSRT();	OSErr	slots.h
SSetsRsrcState();	OSErr	slots.h
StackSpace();	long	memory.h
Status();	OSErr	devices.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
StdArc();	void	quickdraw.h
StdBits();	void	quickdraw.h
StdComment();	void	quickdraw.h
StdGetPic();	void	quickdraw.h
stdline();	void	quickdraw.h
StdLine();	void	quickdraw.h
StdOval();	void	quickdraw.h
StdPoly();	void	quickdraw.h
StdPutPic();	void	quickdraw.h
StdRect();	void	quickdraw.h
StdRgn();	void	quickdraw.h
StdRRect();	void	quickdraw.h
stdtext();	void	quickdraw.h
StdText();	void	quickdraw.h
StdTxMeas();	short	quickdraw.h
StillDown();	Boolean	events.h
StopAlert();	short	dialogs.h
Str2Form();	FormatStatus	script.h
String2Date();	String2DateStatus	script.h
String2Time();	String2DateStatus	script.h
stringtonum();	void	packages.h
StringToNum();	void	packages.h
stringwidth();	short	quickdraw.h
StringWidth();	short	quickdraw.h
StripAddress();	Ptr	memory.h
stuffhex();	void	quickdraw.h
StuffHex();	void	quickdraw.h
subpt();	void	quickdraw.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
SubPt ();	void	quickdraw.h
SUpdateSRT ();	OSErr	slots.h
SVersion ();	OSErr	slots.h
SwapMMUMode ();	void	osutils.h
SysBeep ();	void	osutils.h
SysEnviron ();	OSErr	osutils.h
SysError ();	void	errors.h
SystemClick ();	void	desk.h
SystemEdit ();	Boolean	desk.h
SystemEvent ();	Boolean	desk.h
SystemMenu ();	void	desk.h
SystemTask ();	void	desk.h
SystemZone ();	THz	memory.h
TEActivate ();	void	textedit.h
TEAutoView ();	void	textedit.h
TECalText ();	void	textedit.h
teclick ();	void	textedit.h
TEClick ();	void	textedit.h
TEContinuousStyle ();	Boolean	textedit.h
TECopy ();	void	textedit.h
TECustomHook ();	void	textedit.h
TECut ();	void	textedit.h
TEDeactivate ();	void	textedit.h
TEDelete ();	void	textedit.h
TEDispose ();	void	textedit.h
TEFromScrap ();	OSErr	textedit.h
TEGetHeight ();	long	textedit.h
TEGetOffset ();	short	textedit.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
TEGetPoint();	struct Point	textedit.h
TEGetScrapLen();	long	textedit.h
TEGetStyle();	void	textedit.h
TEGetText();	CharsHandle	textedit.h
TEIdle();	void	textedit.h
TEInit();	void	textedit.h
TEInsert();	void	textedit.h
TEKey();	void	textedit.h
TENew();	TEHandle	textedit.h
TENumStyles();	long	textedit.h
TEPaste();	void	textedit.h
TEPinScroll();	void	textedit.h
TEReplaceStyle();	void	textedit.h
TEScrapHandle();	Handle	textedit.h
TEScroll();	void	textedit.h
TESelView();	void	textedit.h
TESetJust();	void	textedit.h
TESetScrapLen();	void	textedit.h
TESetSelect();	void	textedit.h
TESetStyle();	void	textedit.h
TESetText();	void	textedit.h
testcontrol();	short	controls.h
TestControl();	short	controls.h
TestDeviceAttribute();	Boolean	quickdraw.h
TEStylInsert();	void	textedit.h
TEStylNew();	TEHandle	textedit.h
TEStylPaste();	void	textedit.h
TEToScrap();	OSErr	textedit.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
TEUpdate();	void	textedit.h
TextBox();	void	textedit.h
TextFace();	void	quickdraw.h
TextFont();	void	quickdraw.h
TextMode();	void	quickdraw.h
TextSize();	void	quickdraw.h
TextWidth();	short	quickdraw.h
TickCount();	unsigned long	events.h
ToggleDate();	ToggleResults	script.h
Tokenize();	TokenResult	script.h
TopMem();	Ptr	memory.h
trackbox();	Boolean	windows.h
TrackBox();	Boolean	windows.h
trackcontrol();	short	controls.h
TrackControl();	short	controls.h
trackcontrol();	short	controls.h
TrackGoAway();	Boolean	windows.h
trackcontrol();	short	controls.h
Transliterate();	OSErr	script.h
UnholdMemory();	OSErr	vmcalls.h
UnionRect();	void	quickdraw.h
UnionRgn();	void	quickdraw.h
Unique1ID();	short	resources.h
UniqueID();	short	resources.h
UnloadScrap();	long	scrap.h
UnloadSeg();	void	segload.h
UnlockMemory();	OSErr	vmcalls.h
unmountvol();	OSErr	files.h

<i>Call</i>	<i>Return type</i>	<i>Header file</i>
UnmountVol();	OSErr	files.h
UnpackBits();	void	toolutils.h
UpdateResFile();	void	resources.h
UpdtControl();	void	controls.h
UpdtDialog();	void	dialogs.h
uprstring();	void	osutils.h
UprString();	void	osutils.h
UprText();	void	script.h
UseResFile();	void	resources.h
ValidDate();	short	script.h
ValidRect();	void	windows.h
ValidRgn();	void	windows.h
VInstall();	OSErr	retrace.h
VisibleLength();	long	script.h
VRemove();	OSErr	retrace.h
WaitMouseUp();	Boolean	events.h
WaitNextEvent();	Boolean	events.h
WakeUpProcess	OSErr	processes.h
WriteASD();	long	asd.h
WriteLocation();	void	script.h
WriteParam();	OSErr	osutils.h
WriteResource();	void	resources.h
XorRgn();	void	quickdraw.h
ZeroScrap();	long	scrap.h
ZoomWindow();	void	windows.h

Index

32-bit addressing 4-3
32-Bit QuickDraw 5-4
 C header file for F-5

A

accessing resource data E-33
access privileges 6-5
alarm routine 5-28, 5-30
Alias Manager 5-5
align type E-25 to E-26
A-line traps
 and UNIX device drivers C-4
 C header file for F-30
 handling C-4
Apple Desktop Bus 5-5
AppleDouble-format files
 filename conventions for 6-20
 magic number for 6-19
 maximizing efficiency of 6-19
 overview of 6-9 to 6-10, 6-19
Apple Event Manager 5-5
AppleSingle-format files 6-16 to 6-18
 entry ID field 6-17
 filename conventions for 6-20
 header contents 6-16 to 6-18
 home file system field 6-17
 length field 6-18
 magic number for 6-17

 maximizing efficiency of 6-18
 number of entries field 6-17
 offset field 6-18
 overview of 6-9 to 6-10
 version number for 6-17
Apple Sound Chip (ASC), systems
 containing 5-23
AppleTalk (communications software) 1-2
AppleTalk Manager 5-5
application development
 environments 2-2
application portability 4-7
array type E-27
ASC, systems containing 5-23
AUXDispatch trap 3-4
 C header file for F-17
A/UX files. *See also* files
 file structure 6-9
 simple A/UX format 6-13
A/UX Finder
 developing applications for 2-4
 entry ID value for "Finder info" field
 6-17 to 6-19
 file information and Segment
 Loader 5-19
A/UX Release 3.0
 connectivity support 1-5
 Finder user interface 1-3

 increased manager support 1-4
 new features in 1-3 to 1-5
A/UX system calls 3-8
A/UX Toolbox
 access to Macintosh ROM routines 1-2
 code compatibility provided by 1-2
 configuration requirements 1-2
 contents 1-7 to 1-8
 environment variables 3-6 to 3-7
 initialization of C-2 to C-3
 overview of functions in 1-8
 utilities 3-2 to 3-3
 variables 3-3

B

\ (backslash), escape character in
 resource descriptions E-50
Berkeley UNIX file system (UFS) 1-4,
 5-11
Binary-Decimal Conversion Package 5-6
 C header file for F-36
binary files, transferring to A/UX 6-10
bitstring type E-22
Boolean type E-23
{ } (braces) in type declarations E-31
byte type E-22

C

- C and Pascal language conventions
 - compared C-7 to C-10
- case-sensitive filenames 5-17
- C compilers 4-10
- Chain routine 5-20
- changesize utility 3-3
- change statement E-9
- character sets 4-9
- character type E-23
- C header files F-2 to F-4. *See also individual libraries*
- 'cicn' resource E-37
- C interface libraries F-2 to F-4
- clKwREr error 5-29
- Color Manager 5-6
- Color Picker Package 5-6
 - C header file for F-14
- Color QuickDraw 5-4
 - C header file for F-5
- , (comma), element separator in arrays E-28
- COMMAND-CONTROL-E 3-12
- COMMAND-CONTROL-I 3-12
- commands, resource compilation
 - derez E-3, E-6
 - echo E-9
 - escaping in derez E-51
 - rez E-2 to E-51
- comments
 - entry ID value for 6-17
 - in resource descriptions E-6
- compatibility between UNIX and Macintosh OS 4-2 to 4-11
- compilers
 - C 4-10
 - rez E-2 to E-51
- Control Manager 5-6
 - C header file for F-15
- conversions between C and Pascal 4-11
- converting between file types 3-3
- cstring type E-24
- CurPageOption (Segment Loader) 5-20

D

- Data Access Manager 5-7
- data files, filename conventions for 6-9, 6-20
- data fork, entry ID value for 6-17
- data statement E-10
- data-type statement E-21
- date command 5-29
- dbx debugger 3-13
- debuggers 1-6
 - dbx 3-13
 - MacBug 3-11 to 3-13, 5-27
- Deferred Task Manager 5-7
 - C header file for F-17
- define directive E-40
- Delay utility 5-28, 5-29
- delete statement E-11
- derez resource decompiler E-2 to E-51
- derez utility 3-3
- Desk (Accessory) Manager 5-7
 - C header file for F-18
- Desktop Manager 5-7
- developing applications
 - creating resource files 2-9, E-4
 - resource files 2-9 to 2-10
 - summary 2-6 to 2-10
 - writing source code 2-7 to 2-8
- device drivers 4-6, 5-7 to 5-8
 - and A-line traps C-4
- Device Manager 5-7 to 5-8
 - C header file for F-18
- Dialog Manager 5-8
 - C header file for F-19
- Disk Driver 5-9
 - C header file for F-21
- Disk Initialization Package 5-9
 - C header file for F-21, F-36
- \$\$ (dollar-sign) functions E-33 to E-39, E-46 to E-49
- dontForeground variable 3-3
- DoVBLTask function 5-30

E

- Edition Manager 5-9
- entry ID field (AppleSingle-format files) 6-17
- escape characters in resource descriptions
 - \ (backslash) E-50
 - ∂ (OPTION-D) E-9
- Event Manager, Operating System 5-9 to 5-10
 - C header file for F-22
- Event Manager, Toolbox 5-10
 - C header file for F-22
- events, monitoring 3-6
- examples of resource code 2-10 to 2-11
 - numeric escape sequences E-51
 - resource definition E-17
 - resource description file E-6
 - resource type statement E-30
 - using labels E-36 to E-39
- expressions in resource descriptions E-44

F

- fcnvT utility 3-3, 6-13
- "file info" field
 - entry ID value for 6-17
 - structure of entries in 6-18
- File Manager 5-11
 - C header file for F-23
- filenames
 - AppleDouble file conventions 6-20
 - AppleSingle file conventions 6-20
 - case-sensitivity of 5-17, 6-4
 - compatibility problems for blanks embedded in 6-4
 - overview of 6-3 to 6-4
- file permissions 6-4 to 6-5
- files
 - AppleDouble-format 6-9 to 6-10, 6-19, 6-20
 - AppleSingle-format 6-9 to 6-10, 6-16 to 6-18, 6-20
 - A/UX 6-9 to 6-10, 6-13

- formatting strategies of A/UX
 - Toolbox 6-14
 - Macintosh OS file structure 6-8 to 6-9
 - resource 5-17, E-5
 - simple A/UX format 6-13
 - standard type declaration for E-3
 - file systems
 - access privileges 6-5
 - automatic conversion between UNIX
 - and Macintosh OS files 6-15
 - defined 6-2
 - extended file attributes 6-6
 - file permissions 6-4 to 6-5
 - foreign file system defined 6-14
 - home file system defined 6-14
 - implementation in A/UX and
 - Macintosh OS 6-2
 - mounting and unmounting floppy
 - disks 6-7 to 6-8
 - overall organization (A/UX) 6-2 to 6-3
 - text files 6-6 to 6-7
 - fill type E-25
 - Finder. *See* A/UX Finder
 - Floating-Point Arithmetic Package 5-11
 - C header file for F-36
 - floppy disks, mounting 6-15
 - folders, file permissions for 6-5 to 6-6
 - Font Manager 5-12
 - C header file for F-28
 - foreign file system, defined 6-16
 - functions in resource descriptions E-46
- G**
- gestaltAUXVersion 3-9
 - Gestalt Manager 5-12 to 5-13
 - GetDateTime utility 5-29
 - GetNextEvent routine 2-4, 5-10
 - global variables, Macintosh C-6, D-1
 - to D-7
 - glue routines C-6
 - Graphics Devices Manager 5-13
- H**
- hardware access 4-6 to 4-7
 - header files (AppleDouble-format)
 - filename conventions for 6-20
 - magic number in 6-19
 - overview of 6-9 to 6-10, 6-19
 - Help Manager 5-14
 - HFS 1-4
 - hierarchical file system (HFS) 1-4
 - home file system
 - defined 6-16
 - field for 6-17
- I**
- icons, entry ID value for 6-17
 - identifiers in preprocessor directives E-39
 - ifdef directive E-41
 - ifndef directive E-41
 - if-then-else directives E-41
 - include directive E-40
 - include statement E-12
 - integer type E-22
 - international character support 4-9
 - International Utilities Package 5-14
 - C header file for F-36
- J**
- journaling 5-10
 - jump table F-30
- K**
- kermit utility 6-13
 - KeyRepThresh global variable,
 - compatibility problems with 5-10
 - KeyThresh global variable,
 - compatibility problems with 5-10
- L**
- labels E-32 to E-39
 - in arrays E-34
 - built-in functions for E-33
 - limitations of E-35
 - language conventions, differences in 4-11
 - Launch routine 5-20
 - launch utility 3-2
 - libraries implemented in A/UX Toolbox
 - 5-2 to 5-4
 - List Manager Package 5-14
 - available through Package
 - Manager 5-15
 - C header file for F-29
 - literals in resource descriptions E-43
 - LoadSeg routine 5-20
 - logout 5-22
 - longint type E-22
 - low-memory global variables C-6, D-1
 - to D-7
- M**
- Macintosh events, monitoring 3-6
 - Macintosh file system (MFS) 1-4
 - Macintosh OS
 - file structure 6-8 to 6-9
 - interface with A/UX Toolbox 1-8
 - utilities implemented in A/UX
 - Toolbox 5-28 to 5-29
 - Macintosh traps F-30
 - /mac/lib/rincludes directory E-3
 - MacsBug debugger 3-11 to 3-13, 5-27
 - magic number
 - for AppleDouble-format files 6-19
 - for AppleSingle-format files 6-17
 - Memory Manager 5-14
 - C header files for F-31
 - importance of using 4-3
 - Menu Manager 5-15
 - C header file for F-34
 - MFS 1-4
- N**
- Network File System (NFS) 1-4
 - newline character 4-7 to 4-8
 - compatibility problems with 5-8
 - NFS 1-4
 - noCD variable 3-3

Notification Manager 5-15
 C header file for F-36
"not in ROM" routines C-6
(number sign) in preprocessor
 directives E-39
numbers in resource descriptions E-43
numeric escape sequences in resource
 descriptions E-51
numeric types E-21

O

Operating System Utilities 5-28 to 5-29
 C header file for F-57
operators in resource descriptions E-45

P

Package Manager 5-15
 C header file for F-36
Palette Manager 5-15
 C header file for F-38
Pascal, passing small structures in C-9
Pascal and C language conventions
 compared C-7 to C-10
Pascal function type 4-10
patched Toolbox calls 5-31 to 5-33
pathnames 6-3
permissions, file 6-4 to 6-5
pict.r file E-3
Picture Utilities Package 5-16
point type E-25
(pound sign) in preprocessor
 directives E-39
Power Manager 5-16
'ppat' resource E-36
PPC Toolbox 5-16
PPostEvent 5-10, 5-32, 5-34
preprocessor directives E-6, E-39 to E-42
 for assigning variables E-40
 for conditional processing E-41
 include E-40
 print E-42
print directive E-42

Printing Manager 5-16
 C header file for F-39
print traps F-40
privileged microprocessor instructions
 4-4 to 4-6
privileges, access 6-5
privileges, file 6-5
Process Manager 5-16
 and desk accessories 5-7
 C header file for F-41
ProcPtr parameters 5-8
pstring type E-24

Q

QuickDraw 5-4

R

Raw Sound Driver 5-25 to 5-26
ReadDateTime utility 5-29
read statement E-15
real name, entry ID value for 6-17
rect type E-25
resource compilation using rez E-51
resource description files E-2 to E-7
 comments in E-6
 preprocessor directives in E-6, E-39
 to E-42
 structure of E-5
 type declarations in E-5
resource description statements E-7
 to E-39
 align type E-25
 array type E-27
 Boolean type E-23
 change statement E-9
 character type E-23
 data statement E-10
 data-type statement E-21
 delete statement E-11
 expressions in E-44
 fill type E-25
 functions in E-46
 include statement E-12

literals in E-43
numbers in E-43
numeric escape sequences in E-51
numeric types E-22
operators in E-44
point type E-25
read statement E-15
rect type E-25
resource statement E-16
separators in arrays E-28
special terms in E-8
string type E-24
switch statement E-29
syntax of E-7 to E-8, E-43 to E-51
terminators in arrays E-28
type statement E-20
variables in E-46
resource fork, entry ID value for 6-17
Resource Manager 5-17 to 5-18
 C header files for F-42
 differences between
 environments 5-17
 resource statement E-16
resources (Macintosh) 2-9. *See also*
 examples of resource code
 attributes of E-13
 data statements in E-16 to E-17
 preprocessor directives in E-39 to E-42
 symbol definitions in E-31
 symbolic names in E-19
 type declaration files for E-3
 type declarations in E-5
RestoreA5 routine 5-29
rez resource compiler E-2 to E-51. *See*
 also resource description files;
 resource description statements
rez utility 3-3
ROM definitions F-17

S

sample programs, Toolbox 2-10 to 2-11
SANE (Standard Apple Numeric
 Environment) 5-11

- Scrap Manager 5-18
 - C header file for F-45
 - Script Manager 5-18
 - C header file for F-45
 - scripttypes.r file E-3
 - SCSI Manager 5-18 to 5-19
 - search paths, compatibility issues
 - with 5-17
 - Segment Loader 5-19 to 5-20
 - C header file for F-48
 - CurPageOption setting 5-20
 - routines different in A/UX 5-20
 - select system call 3-6
 - ;(semicolon), element terminator in arrays E-28
 - separators in arrays E-28
 - Serial Driver 5-20 to 5-22
 - C header file for F-49
 - differences in A/UX 5-20 to 5-21
 - SetDateTime utility 5-29
 - setfile utility 3-2
 - setitimer routine 5-28, 5-30
 - SetUpA5 routine 5-29
 - Shutdown Manager 5-22
 - C header file for F-49
 - SIGNALRM signal 5-28, 5-30
 - 'SIZE' resource 2-4
 - sleep routine 5-28, 5-30
 - Slot Manager 5-22 to 5-23
 - C header file for F-50
 - declarations F-17
 - Sound Manager 5-23 to 5-26
 - C header files for F-52
 - special terms in resource description statements E-8
 - Standard Apple Numeric Environment (SANE) 5-11
 - Standard File Package 5-27
 - C header file for F-36
 - startmac utility 3-2
 - startmac24 utility 3-2
 - stime call 5-29
 - string type E-24
 - strings
 - converting between Pascal and C F-54
 - in resource descriptions E-49
 - types of E-23
 - StripAddress routine 5-28
 - structures, passing C-9
 - SVFS 1-4
 - switch statement E-29
 - symbolic names E-19
 - of resource description statements E-7
 - syntax of resource description statements E-7 to E-8
 - SysBeep routine 5-28
 - SysError system call 5-27
 - system calls 3-8
 - System Error Handler 5-27
 - C header file for F-54
 - System V file system (SVFS) 1-4
 - systypes.r file E-3
- T**
- TBCORE variable 3-7
 - TBRAM variable 3-7
 - TBSYSTEM variable 3-7
 - TBTRAP variable 3-7
 - TBWARN variable 3-7
 - terminators in arrays E-28
 - TextEdit 5-27
 - C header file for F-54
 - 32-bit addressing 4-3
 - 32-Bit QuickDraw 5-4
 - C header file for F-5
 - Time Manager 5-28
 - C header file for F-56
 - time operations 4-7, 5-29
 - Toolbox Utilities 5-29
 - C header file for F-58
 - Transcendental Functions Package 5-11
 - to 5-12
 - C header file for F-36
- U**
- UFS 1-4, 5-11
 - uinter0 device driver C-3
 - ui_setselect call 2-5 to 2-6
 - undef directive E-40
 - UnloadSeg routine 5-20
 - unsupported Toolbox calls 5-34
 - user-interface device driver C-2
 - utilities, Macintosh 5-28
 - Delay 5-28
 - fcvt 6-13
 - kermit 6-13
 - Utilities, Operating System 5-28 to 5-29
 - C header file for F-57
 - Utilities, Toolbox 5-29
 - C header file for F-58
- V**
- variables, A/UX Toolbox 3-3
 - variables in resource descriptions E-46
 - version number, for AppleSingle-format files 6-17
 - Vertical Retrace Manager 5-29
 - C header file for F-60
 - Video Driver
 - C header file for F-60
 - virtual memory, limits 4-9
- W, X, Y, Z**
- WaitNextEvent routine 2-4 to 2-5
 - using select after 2-5
 - Window Manager 5-30
 - C header file for F-61
 - 'WIND' resource type, example E-18
 - wstring type E-24

The Apple Publishing System

A/UX Toolbox: Macintosh ROM Interface was written, edited, and composed on a desktop publishing system using Apple Macintosh computers, an AppleTalk network system, Microsoft Word, and QuarkXPress. Line art was created with Adobe Illustrator. Proof pages were printed on Apple LaserWriter printers. Final pages were output directly to 70-mm film on an Electrocomp 2000 Electron Beam Recorder. PostScript, the LaserWriter page-description language, was developed by Adobe Systems Incorporated.

Text and display type are Apple's corporate font, a condensed version of ITC Garamond®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier, a fixed-width font.

Writer: Tom Berry

Developmental Editor: Paul Dreyfus and Silvio Orsino

Design Director: Lisa Mirski

Art Director: Joyce Zavarro

Production Editor: Debbie McDaniel

Special thanks to Winston Hendrickson, Michael Hinkson, and Kelly King.

Additional thanks to Eric Castle, Tim Dierks, Jim Mullin, and Kent Sandvik.