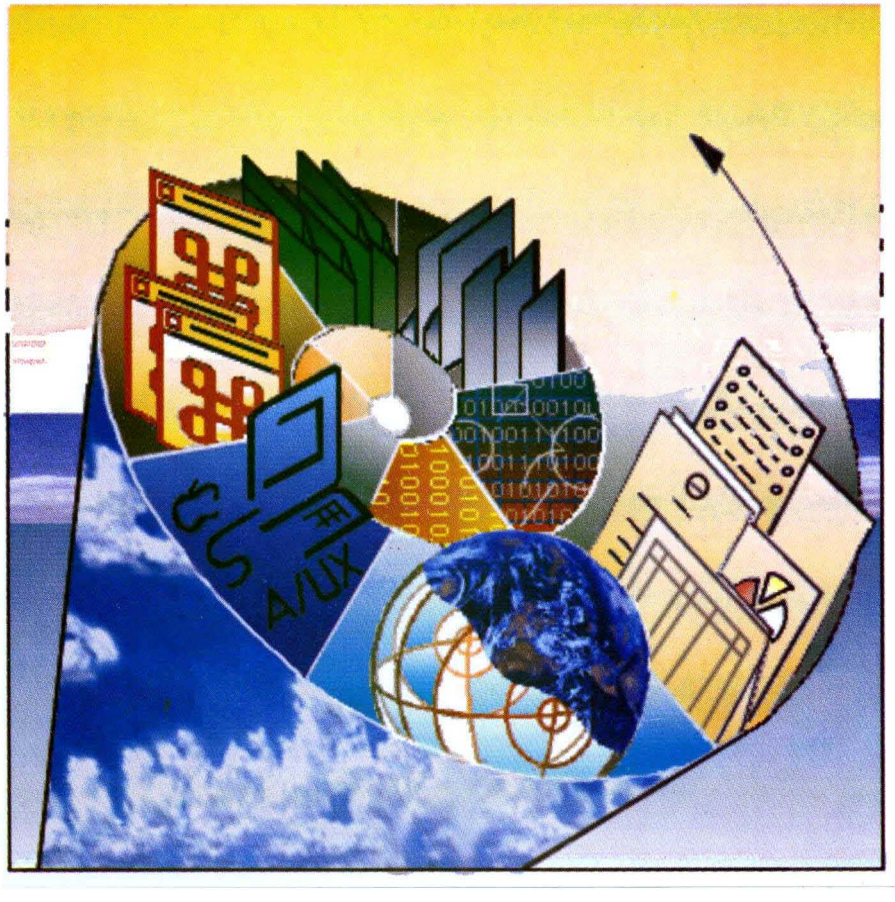
A/UX® Toolbox: Macintosh® ROM Interface



A/UX® Toolbox: Macintosh® ROM Interface

030-0787-A

APPLE COMPUTER, INC.

© 1990, Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc. Printed in the United States of America.

The Apple logo is a registered trademark of Apple Computer, Inc. Use of the "keyboard" 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.
20525 Mariani Ave.
Cupertino, California 95014
(408) 996-1010

Apple, the Apple logo, AppleLink, AppleShare, AppleTalk, A/UX, LaserWriter, LocalTalk, Macintosh, MacTCP, MPW, MultiFinder and SANE are registered trademarks of Apple Computer, Inc.

APDA, Finder, MacX, QuickDraw, ResEdit and SuperDrive are trademarks of Apple Computer, Inc.

Ethernet is a registered trademark of Xerox Corporation.

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

Microsoft is a registered trademark of Microsoft Corporation.

NuBus is a trademark of Texas Instruments.

POSTSCRIPT is a registered trademark, and Illustrator is a trademark of Adobe Systems, Incorporated.

UNIX is a registered trademark of AT&T.

Simultaneously published in the United States and Canada.

030-0787-A

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, Apple will replace the media or manual 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 THIS MANUAL, 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 reviewed this manual, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

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.

Contents

Figures and tables

Preface

This guide describes the A/UX® Toolbox, which gives you access from within A/UX to the User Interface Toolbox that is part of the Apple® Macintosh® computer. This guide also provides compatibility guidelines for programs intended to run under both the A/UX operating system and the standard Macintosh Operating System (OS).

What's in this guide

This preface describes the guide and the conventions used in it. Following the Preface, here's what you'll find inside:

- Chapter 1, "About the A/UX Toolbox," gives an overview of the A/UX Toolbox.
- 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 Checklist," summarizes the compatibility guidelines you must follow to write code that runs under both the Macintosh OS and A/UX.
- Chapter 5, "Inside A/UX Macintosh," describes in detail the differences between the standard libraries used in the User Interface Toolbox and Macintosh OS and the libraries provided with the A/UX Toolbox.
- Chapter 6, "File Systems and File Formats," describes the differences of file systems in A/UX and in the Macintosh OS, 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 all of the files shipped with the A/UX Toolbox and their functions.
- 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 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.

What you need to know

You need this guide if you are porting a Macintosh application to A/UX or developing a Macintosh-like application under A/UX. This guide assumes:

- You are an experienced C programmer. If not, see the standard C reference manuals listed in Appendix A, "Additional Reading."
- You are familiar with the standard Macintosh User Interface Toolbox and Operating System. If not, see *Inside Macintosh,* Volumes I through V, and the Macintosh programming guides listed in Appendix A, "Additional Reading."
- You are familiar with the A/UX development environment. If not, see *A/UX Essentials* and *Road Map to A/UX* for an introduction to the A/UX system and the manuals that describe it.

The technical terms used in this guide are defined in the glossaries in *Inside Macintosh* and *A/UX Essentials.*

Conventions used in this guide

A/UX guides follow specific conventions. 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	ESCAPE	SHIFT
COMMAND	LEFT ARROW	TAB
CONTROL	RETURN	UP ARROW
DOWN ARROW	RIGHT ARROW	

For example, suppose you enter

```
Applee
```

instead of

```
Apple
```

To erase the additional *e*, you would position the cursor (or insertion point) to the right of the word and press the DELETE key once.

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 an entry and press the RETURN key. The instruction

Enter `ls`

means "Type `ls` and press the RETURN key."

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

Term	Action
Choose	Activate a command in a menu. To choose a command from a pull-down menu, click once on the menu title while holding down the mouse button, and drag down until the command is highlighted. Then release the mouse button.
Click	Press and then immediately release the mouse button.
Drag	Position the pointer on an object, then press and hold down the mouse button while moving the mouse. Release the mouse button when the object reaches the desired position on the screen.
Enter	Type the letter or letters and press the RETURN key.
Press	Type a *single* key *without* pressing the RETURN key. Or position the pointer on an object and hold down the mouse button.
Select	Position the pointer on a selectable object and click the mouse button.
Type	Type an entry *without* pressing 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 the 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 type `date` and 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.

Font styles

Words that you must replace with a value appropriate to a particular set of circumstances appear in *italics.* For example, if you see
`cat` *filename*

replace the italicized word with the name of the 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.

A/UX command syntax

A/UX commands follow a specific command syntax. A typical A/UX command has this form:
`command` [*flag-option*] [*argument*]...

The following table outlines the elements of an A/UX command.

Element	Description
`command`	The command name.
flag-option	One or more optional arguments that modify the command. Most flag options have the form [-*opt*...], where *opt* is a letter representing an option. Most commands have one or more flag options.
argument	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 an argument that can be repeated any number of times.

For example, the `wc` command is used to count lines, words, and characters in a file. Here is the full syntax for that command, including all possible flag options and the optional argument *name*.

`wc [-c] [-l] [-w] [`*name*`...]`

Thus, you can enter

```
wc -w /Priscilla
```

to count all of the words in the file `/Priscilla`, where `wc` is the name of the command, `-w` is the flag option that instructs the command to count all of the words in the file, and the optional argument `/Priscilla` is the file to be searched.

Command reference notation

A/UX Command Reference, A/UX Programmer's Reference, and *A/UX System Administrator's Reference* contain references for commands, programs, and other related information. Material is organized within these references by section numbers. The standard A/UX cross-reference notation is

cmd (sect)

where *cmd* is the name of the command, file, or other facility; *sect* is the section number where the entry resides.

- Items followed by section numbers (1M), (7), or (8) are listed in *A/UX System Administrator's Reference.*
- Items followed by section numbers (1), (1C), (1G), (1N), and (6) are listed in *A/UX Command Reference.*
- Items followed by section numbers (2), (3), (4), and (5) are listed in *A/UX Programmer's Reference.*

For example,

`cat(1)`

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

References can be also called up on the screen. Use the `man` command to display pages from reference manuals, known as manual pages, directly on the screen. 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 shell prompt, or type `q` at any time to return immediately to your shell prompt.

Cross-referencing

An A/UX guide often refers to information discussed in another guide in the suite. The format for this type of cross-reference is "Chapter Title," *Name of Guide.*

For a complete description of A/UX guides, see *Road Map to A/UX.* It contains descriptions of each A/UX guide, part numbers, and ordering information for all the guides in the A/UX documentation suite.

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

The A/UX® Toolbox is a library that enables a program 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 take advantage of the standard Macintosh support tools to write A/UX programs that look and act like Macintosh programs.

For a detailed description of the User Interface Toolbox, see *Inside Macintosh,* Volumes I, IV, and V. For a detailed description of the Macintosh OS, see *Inside Macintosh,* Volumes II, IV, and V. You must understand how to write a Macintosh application program to use this manual and the A/UX Toolbox.

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 2.0.

Overview

The A/UX Toolbox bridges the Macintosh and A/UX 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 Release 2.0" makes clear, the new capabilities of the A/UX Toolbox remove many prior limitations.)
- 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) on the Macintosh. Because of differences between A/UX and the Macintosh OS, not all Macintosh ROM routines are available through the A/UX Toolbox. Release 2.0 has increased support for Macintosh ROM routines. Programs that are intended to run in both environments can use only the ROM routines common to both. Any Macintosh application that runs under the Macintosh OS MultiFinder™, does not do direct input and output operations, and does not call routines missing from the A/UX Toolbox, can run under A/UX Release 2.0. Chapter 5, "Inside A/UX Macintosh," gives details about all Macintosh managers and their support under the A/UX Toolbox; Table 5-1 in that chapter summarizes manager support.

The A/UX software includes the standard desk accessories that come with Macintosh system software for the Macintosh SE/30 and Macintosh II family of computers. The Chooser and the Control Panel now support a serial printer.

A/UX Release 2.0 also includes support for AppleTalk® communications software, running on both LocalTalk™ and Ethernet hardware.

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

Configuration requirements: none

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.

Prior to Release 2.0, if you were developing an A/UX Toolbox application using the `adb` or `sdb` debuggers, you needed a terminal that could communicate with your computer over either a serial line or a network. This terminal is no longer necessary when using the MacsBug debugger, which now is available in a version for use with A/UX. The `adb` and `sdb` debuggers require use of a terminal.

Contents of the A/UX Toolbox

The following list summarizes what types of files 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.

- `/mac/sys`
  This directory contains the system folders for startup and login. The System file provided with Release 2.0 of A/UX is almost identical in functionality to the System file provided with Release 6.0.5 of the Macintosh system software.
- `/mac/bin`
  This directory contains executables, including a few utilities for use when developing and running applications with the A/UX Toolbox. See Chapter 3, "A/UX Toolbox Utilities and Extensions," for descriptions of `fcnvt`, `settc`, `rez`, and `derez`.
- `/mac/src`
  This directory contains source code for sample applications, including `sample`, `qdsamp`, and the Sound Manager demo, `sndDemo`. The source material includes associated makefiles, which demonstrate how to compile and link an application, and 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 `cmdo` 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 or in UNIX program development. An example of the first kind is `maccrt0.o`, and examples of the second kind are in `libs.a`, `libposix.a`, and `termcap.a`.

- `/usr/include/mac`

  The C interface files that define the constants, types, and functions used by the A/UX Toolbox libraries are in this directory. 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 describes briefly the function of each file.

Standards compliance of A/UX Release 2.0

A/UX Release 2.0 is in compliance with the following standards:

- AT&T SVID, Issue 2—AT&T's System V Interface Definition
- IEEE POSIX 1003.1-1988—the Institute of Electrical and Electronics Engineers' (IEEE) Standard Portable Operating System Interface for Computer Environments (POSIX)
- ISO 9945-1—the International Standards Organization's designation that is equivalent to POSIX 1003.1-1988
- FIPS 151-1—the most current version of the Federal Information Processing Standard

For information about the A/UX POSIX environment and a guide to POSIX conformance, see *Programming Languages and Tools, Volume 1.*

New features in A/UX Release 2.0

A/UX Release 2.0 is a major enhancement that brings together a standard UNIX operating system and programming environment with Macintosh system capabilities, including the standard Macintosh Finder™ user interface. Macintosh applications with extensive graphics and sound capabilities, for example, can now run in a full UNIX environment. Macintosh applications can run together in the A/UX Toolbox environment, and transfer information between applications by using the clipboard, as in the Macintosh OS. For example, a user can run a graphics application and a word-processing application, alternately developing graphics and text and then copying graphics into the text. A user can run a 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 2.0 are listed in the general manuals. This section presents the major points of interest from a developer viewpoint.

A/UX Finder user interface

A/UX Release 2.0 supports the Finder user interface, and displays both Macintosh OS and A/UX applications and directories as icons. The A/UX ownership and permission status is shown in the icon display of files and folders. Icons show as bright or dim according to the access permissions of the user who logged on. Macintosh floppy disks are accessible from the A/UX Finder.

Applications of both kinds can be opened by double-clicking, and files can be moved by dragging. Files moved between the two environments, A/UX and A/UX Toolbox, are automatically converted. Chapter 6, "File Systems and File Formats," discusses the results of automatic conversion.

Compatibility requirements

For a Macintosh application binary to run in the A/UX Release 2.0 environment, it must meet certain requirements, which are briefly summarized below. 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. (In Release 1.1, 29-bit addressing was sufficient.)

- MultiFinder ready

  A Macintosh binary that does not run in a Macintosh OS multitasking (MultiFinder) environment will not run under Release 2.0.

- No calls on unsupported commands

  An application cannot call on commands that are not supported under the A/UX Toolbox.

- No direct input/output

  An application cannot issue instructions for direct control of hardware.

A special 24-bit environment is also furnished that provides a 24-bit test environment for developers converting applications to 32-bit clean and supports running older applications that have not been converted and orphan applications that never will be. The 24-bit environment is accessed by a special login.

Increased manager support

The level of support for managers in Release 2.0 of the A/UX Toolbox has been generally extended to the support provided by Release 6.0.5 of the Macintosh OS. Several managers that were not supported or were partially supported are now fully supported, and two new managers have been added.

- 32-Bit QuickDraw™ Manager (with Color QuickDraw)

  The 32-Bit QuickDraw Manager is fully supported.

- File Manager

  The File Manager has increased support. It now supports Toolbox access to the various A/UX file systems (BSD, SVFS, NFS) as well as those of the Macintosh OS (HFS, MFS, AppleShare®). Now that Release 2.0 supports the BSD file system, names on such volumes may now be much longer than with SVFS (up to 255 characters).

- Memory Manager

  The Memory Manager is fully supported.

- Notification Manager

  The new Notification Manager is fully supported.

- Serial Driver

  The Serial Driver is partially supported. The driver allows synchronous calls to serial I/O. This provides support for printer and plotter devices accessed by Toolbox calls and for applications such as AppleLink®.

- Slot Manager

  The Slot Manager is partially supported.

- Sound Manager

  The Sound Manager is partially supported. The Sound Manager provides continuous sampled sound as well as note and wavetable synthesizing. The System 6.0.5 Sound Manager does not have MIDI capability.

Additional enhanced support

A/UX Release 2.0 supports the following:

- MacsBug

  The MacsBug debugger is supported. For information, see "The MacsBug Debugger Under A/UX" in Chapter 3.

- Shared libraries

  Shared libraries are supported. Shared libraries offer the possibility of smaller program size, with less memory and disk space required. They also help in application maintenance, by providing the current library version for applications calling on them. A shared version of the standard C library is provided. For information, see "Shared Libraries" in *A/UX Programming Languages and Tools, Volume 1.*

- UNIX Shell windows

  UNIX Shell windows are provided in a manner similar to the older `term` program, but with enhancements to the user interface. The new program is called `CommandShell`.

- NFS 3.2 enhancement

  The NFS 3.2 enhancement to support byte-locking and special devices is provided.

- SuperDrive

  The Apple® SuperDrive peripheral, which supports floppy disks with capacities of 400 or 800 kilobytes, and 1.44 megabytes, is supported. Either HFS or A/UX disk formats may be used.

- Utility enhancements

  With the tighter integration of A/UX Release 2.0 and the Macintosh OS, several Release 1.1 utilities have become obsolete. The `toolboxdaemon` is obsolete in the new implementation, and so are the `mfs` and `hfx` utilities. These three are no longer furnished. The `fcnvt` utility has been enhanced and now supports conversion from and to six file formats. The resource compiler and decompiler utilities, `rez` and `derez`, have been enhanced. The `fcnvt`, `rez`, and `derez` utilities are discussed in Chapter 3, "A/UX Toolbox Utilities and Extensions," with more information on the last two in Appendix E, "Resource Compiler and Decompiler." File formats are discussed in Chapter 6, "File Systems and File Formats."

Ease-of-use extensions

In addition to the standard Macintosh user interface you can use in the A/UX environment, other extensions make A/UX easier for those not familiar with UNIX. Two of these enhancements are:

- Macintosh-style text editor

  Users may specify this TextEditor as their default editor. This editor is explained in *A/UX Essentials*.

- Commando

  The optional Commando facility provides a convenient help and prompt tool for using commands. Commando is taken from MPW™, the Macintosh Programmer's Workshop, and adapted to the A/UX environment.

Connectivity support

Release 2.0 includes the following enhancements:

- AppleTalk 2.0

  Printer access by LocalTalk or EtherTalk™ is supported.

- MacTCP™

  Macintosh network applications written to the Macintosh Transmission Control Protocol (MacTCP) programmer interface are supported.

- AFP client application

  The AppleTalk Filing Protocol (AFP) client application allows access to AppleShare volumes from the A/UX Finder desktop.

- Communications Toolbox

  The Communications Toolbox is supported.

Two optional implementations of the X Window system are available for Release 2.0, both described in *Getting Started with X Window System for A/UX*.

- X11 is a standard implementation of the X Window System developed at the Massachusetts Institute of Technology, which combines two releases of Version 11 (Releases 3 and 4). X11 provides a complete development environment. Users may switch between the X11 environment and the A/UX Finder environment.
- MacX™ provides a Finder-like environment for X11, with such features as pull-down menus, dialog boxes, and windows in which users may run X applications. MacX also implements Version 11. Users have a range of installation choices for the user environment seen within MacX. At one end, 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 MacX appearance the user chooses, the user may switch between the MacX environment and the A/UX Finder environment.

International character support

The use of international characters is supported as follows:

- The kernel can now handle 8-bit characters and alternate character sets.
- Printing of international characters is now supported.
- International keyboards are supported by appropriate character mapping.

A caution applies to use of international characters within the A/UX environment. In brief, if you have a Macintosh application that supports international characters, the application should run appropriately under A/UX. However, if you attempt to process international characters 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 a 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 characters.

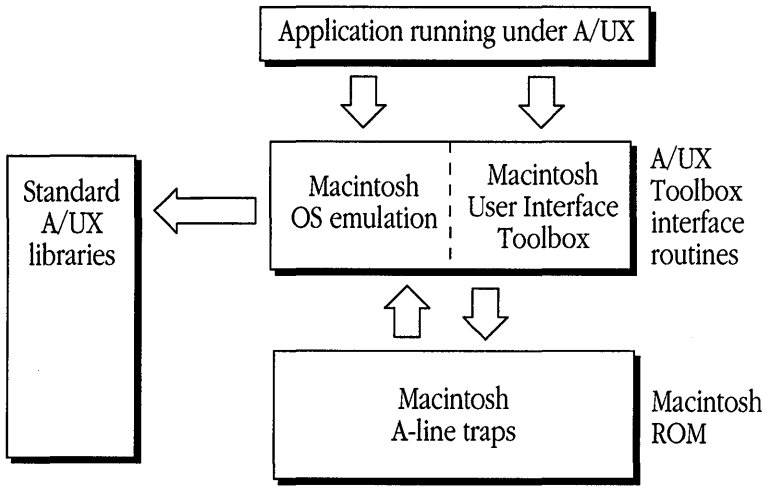
How the A/UX Toolbox works

The primary function of the A/UX Toolbox is to make available to programs running under A/UX the standard Macintosh OS support code described in *Inside Macintosh*. 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, either passes the call 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



Chapter 2 **Using the A/UX Toolbox**

This chapter describes the procedures for developing and running programs that use the A/UX Toolbox. It includes four kinds of information:

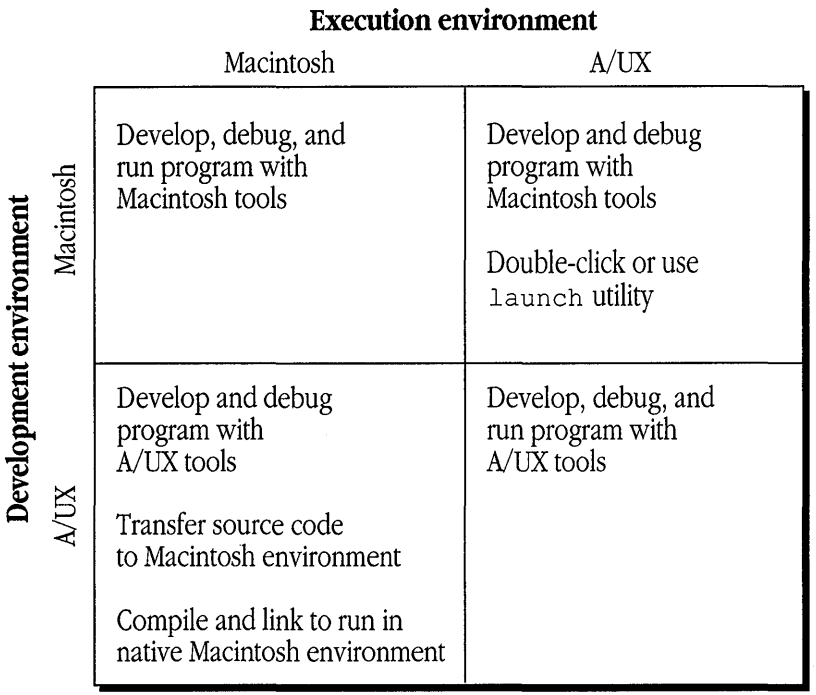
- an overview of how you can develop A/UX Toolbox applications under either the Macintosh OS or A/UX
- an outline of the procedures for porting a Macintosh application to A/UX
- an outline of the procedures for developing an A/UX Toolbox application under A/UX
- descriptions of the sample programs

For a detailed description of the utilities and special features of the A/UX Toolbox that support program development, see Chapter 3, "A/UX Toolbox Utilities and Extensions."

Application development environments

You can develop applications under either the Macintosh OS or A/UX. Through the A/UX Toolbox, you can 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.

■ **Figure 2-1** Application development and execution environments



Choice of the development environment depends on the preference of the developer. Generally speaking, a Macintosh application probably would be developed on the Macintosh side and (if a well-behaved Macintosh application) could be executed under the A/UX Finder with no changes (or perhaps a few, in certain cases). A typical case for development on the A/UX side might be that of adapting an existing UNIX application to run on the Macintosh side as well, and provide a Finder interface when doing so. Although such an application could be developed in either environment, as convenient, an experienced UNIX programmer might use the A/UX environment.

A/UX Release 2.0 supports two key phases of application development:

- Transferring a compiled and linked application from the Macintosh OS to A/UX and running it under A/UX.

  The section "Porting a Macintosh Application to A/UX" in this chapter outlines the procedures for transferring a binary file and running it under A/UX. Chapter 3, "A/UX Toolbox Utilities and Extensions," describes the utilities that support these procedures.

- Developing an application under A/UX that exploits the Macintosh user interface tools.

  The section "Developing an A/UX Toolbox Application" in this chapter outlines the procedures for developing an A/UX Toolbox program. The section "Building and Running the Demonstration Programs" in this chapter describes the sample programs and makefiles provided as examples. Chapter 3, "A/UX Toolbox Utilities and Extensions," describes the utilities that support these procedures.

Both Macintosh binary files transferred to A/UX and A/UX Toolbox programs developed under A/UX must meet the A/UX Toolbox compatibility requirements. With Release 2.0, these requirements are likely to be met by a most applications that meet the standards for current Macintosh OS applications. For details, see Chapter 4, "Compatibility Checklist."

A user can access files in either the A/UX or the A/UX Toolbox environment, and so can an application. You can transfer files between environments either from within an application or by dragging on the desktop. Files transferred between the two environments 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 environments (UNIX persmissions, Macintosh file structure, etc.), transferring a file results in automatic changes in information relating to that file in its new environment. These topics are described in Chapter 6, "File systems and File Formats."

Your application in the A/UX Finder environment

Applications must be 32-bit clean and must be MultiFinder friendly, or at the least, MultiFinder aware, to run under the A/UX Finder. This section discusses how to insure that an application is A/UX Finder friendly. *Programmer's Guide to MultiFinder,* which is available from APDA, provides detailed information on being Multifinder friendly. The points given here extend that information with reference to the A/UX environment and how to be A/UX Finder friendly.

There are certain applications that cannot run under A/UX because they violate A/UX requirements, by doing direct input/output operations, for example, or that rely on Macintosh calls that are not supported under A/UX. Information on these matters is given elsewhere; in particular, in Chapter 4, "Compatibility Checklist," and in Chapter 5, "Inside A/UX Macintosh." The information that follows here is not concerned with these special requirements, but with how, in general, an application can function well in the A/UX Finder environment.

- Use `WaitNextEvent`

  Please do not use `GetNextEvent`, which is very unfriendly to the A/UX kernel scheduler. Use `WaitNextEvent`, with timeouts and mouse regions, if at all possible. This allows the kernel scheduler to put things to sleep, improving the efficiency of CPU usage. `WaitNextEvent` also improves responsiveness to the user, because processes are penalized for accumulated CPU time.

- Do blocking operations only if unavoidable

  The `ui_setselect` call is helpful on this. See "Using the `ui_setselect` call," below, for information.

- Set the `'SIZE'` resource higher

  When setting the `size` resource, allow slightly more memory than for running under the Macintosh OS MultiFinder. Running the Macintosh OS memory management under A/UX requires some additional overhead.

Using the ui_setselect call

If you have an application, such as a terminal emulator, that normally blocks on select(2) for events, you can use the ui_setselect function to block in WaitNextEvent and allow other applications their due slices. Usage is similar to select(2).

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

```
ui_setselect(nfds,rdmask,wtmask,excptmask)   /*set masks*/
WaitNextEvent (...);
ui_setselect (0, 0, 0, 0 0;          /*clear select masks*/
```

The ui_setselect call causes WaitNextEvent to return a null event whenever a select(2) call would succeed. The old method, used with Release 1.1, which first called select and then GetNextEvent, will no longer work properly.

When you get a null event, you can call select with a timeout value of 0 to see if I/O is pending on any file descriptors.

The reason for calling ui_setselect a second time to clear the masks is that calling it to set the masks effectively adds another event type to the event mask for the WaitNextEvent call. Clearing the select 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 would still request an event. In such a case, a null event may be returned to indicate that select would break. For the ModalDialog example, this means the update event for the dialog would not be returned because it has a lower precedence than the select physical event, and the contents of the dialog would not be drawn.

The ui_setselect call is similar in function to select(2), as documented in *A/UX Programmer's Reference,* except for two differences,

- ui_setselect has no timeout argument
- the masks for ui_setselect are integers; for select, they are pointers to integers

Porting a Macintosh application to A/UX

The requirements and recommendations for developing a Macintosh application are given in *Inside Macintosh,* Volumes I through V, and are also presented in a less thorough but more accessible manner in some of the books described in Appendix A, "Additional Reading."

To run under the A/UX Finder, an application

- must be 32-bit clean
- must be at least MultiFinder aware, and preferably MultiFinder friendly
- must not call on Macintosh OS routines that are not supported under the A/UX Toolbox

The first two of these requirements are highly recommended for all Macintosh applications. Guidelines for these three requirments are given in Chapter 4, "Compatibility Checklist." Chapter 5, "Inside A/UX Macintosh," has details on the level of support for Macintosh OS routines.

Developing an A/UX Toolbox application

This section summarizes the procedures for developing an A/UX Toolbox application under A/UX. After you read this summary, see Chapter 3, "A/UX Toolbox Utilities and Extensions," for information on the special tools provided with the A/UX Toolbox for support of program development.

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 an application under the Macintosh OS, see "Additional Reading" at the end of this manual 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 outlines the tools provided for developing the two elements.

The sample programs 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" in this chapter for more information.

△ **Important** Shared libraries are now implemented in Release 2.0. This provides a convenient method of supporting future enhancements by using shared library code for routines subject to change and development. You can update and ship the library code, and the applications that call upon the shared library will automatically use the current code in the library without having to recompile. See "Shared Libraries, in *A/UX Programming Languages and Tools, Volume 1,* for information. △

Developing the source code

You can use the standard A/UX C development environment for developing and debugging an A/UX Toolbox application (see *A/UX Programming Languages and Tools, Volumes 1 and 2*). Use the standard C libraries (shared or non-shared) 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 the available header files), and check the library's entry in Chapter 5, "Inside A/UX Macintosh," for any warnings about the A/UX implementation of that library.

  Follow the general A/UX compatibility guidelines in Chapter 4, "Compatibility Checklist," and the general Macintosh programming guidelines in *Inside Macintosh,* Volumes I through V.

- Building the application

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

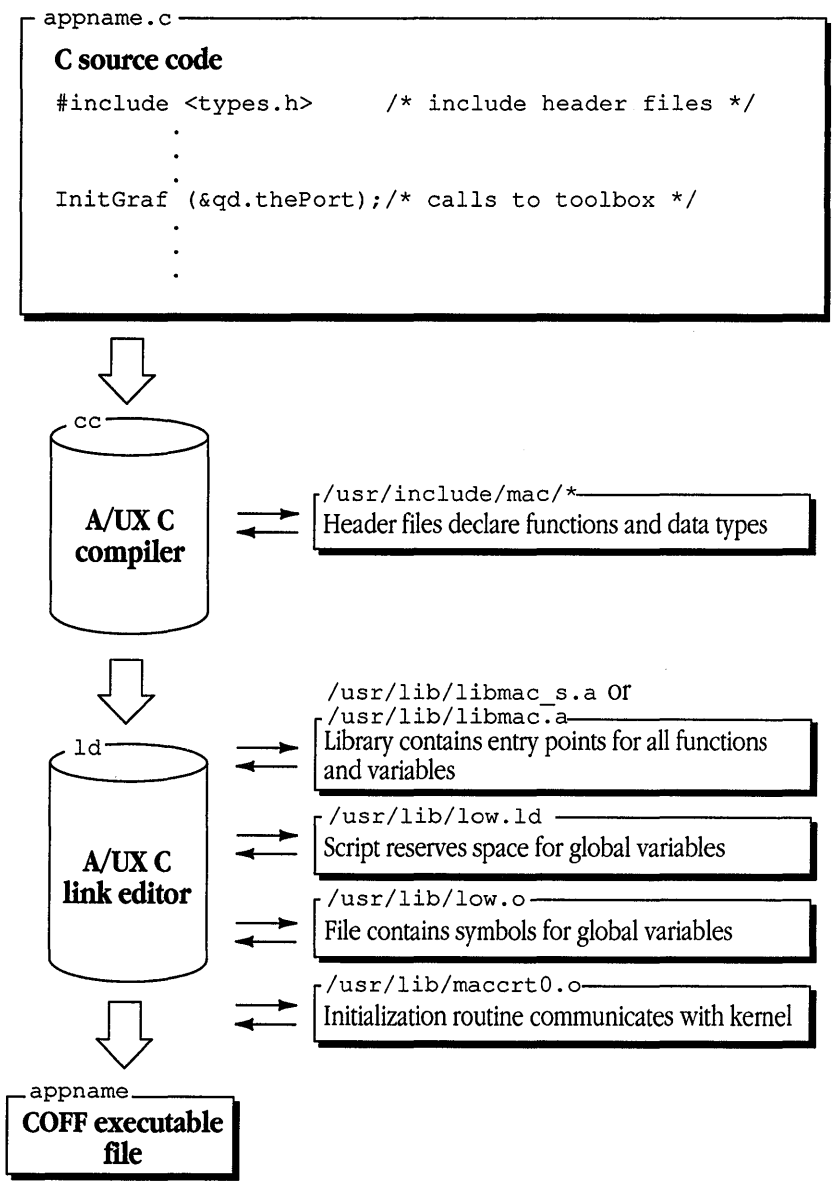
  - specifying the pathname for the include files
  - invoking a script to reserve space for the Macintosh global variables
  - linking to the files that contain the A/UX Toolbox routines, the symbols for Macintosh global variables, and the initialization routine

  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 COFF object file.

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

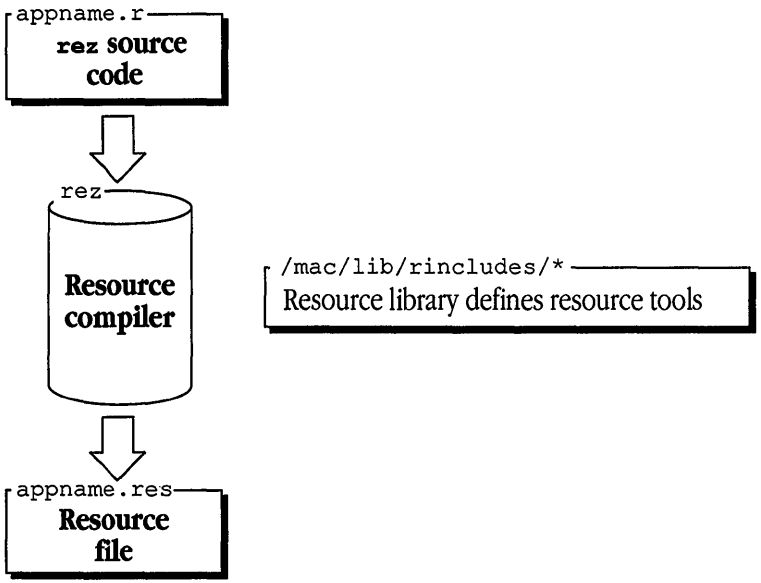


Developing the resource file

The A/UX Toolbox includes A/UX versions of the `rez`(1) and `derez`(1) tools, ported from MPW, for compiling and decompiling resources. `rez` creates an AppleSingle file to hold the compiled resources. The `fcnvt` utility can be used to make this an AppleDouble file.

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`. Figure 2-7 illustrates the resource development path.

■ **Figure 2-3** Developing a resource file under A/UX



ResEdit, which allows you to manipulate resources graphically and to copy resources between applications, now runs under A/UX (version 1.2 or later required). Additional tools and utilities for development of resources under the Macintosh OS are available from third party developers. Apple's Macintosh Programmer's Workshop offers several tools. See *Macintosh Programmer's Workshop 3.1 Reference* for a description of them. If you develop your resources in the Macintosh OS, you can transfer the compiled resources into a file of appropriate format for A/UX.

Building and running the sample programs

Source for sample programs is in `/mac/src/example`. The directory contains a makefile, and a C source file and a resource file for these applications:

`qdsamp.c`	QuickDraw sample program C source and resource file
`qdsamp.r`	
`sample.c`	Sample program C source and resource file; generic application
`sample.r`	

Another example is found in the `/mac/src/sndDemo` directory. The directory contains a makefile, C source file, resource file, C header file, and a sound demo resource.

The directory `/mac/src/examples` contains the C source files for `qdsamp` and `sample`, the uncompiled resource file for `sample`, and the makefile that builds those two programs. (See *Inside Macintosh,* Volume I, for a detailed description of resource files and Macintosh application development procedures.)

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

```
make sample %sample
```

The A/UX COFF file is `sample` and the resource information is in `%sample`.

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

The makefile provided with the demonstration 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 to your program.

Chapter 3 **A/UX Toolbox Utilities and Extensions**

This chapter discusses some special features of the A/UX Toolbox that support program development in A/UX. The features fall into four 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
- additional traps and routines for use in A/UX Toolbox applications
- environment variables for use during debugging

This chapter also describes strategies for writing applications that use both the Macintosh OS and A/UX in two ways:

- making A/UX system calls from within an application that is intended to be run under both the Macintosh OS and A/UX
- allowing an application that runs under A/UX to monitor both A/UX I/O activity and Macintosh events (events that would be handled by the Event Manager under the Macintosh OS)

Using the utilities

The A/UX Toolbox utilities discussed in this chapter are all in `/mac/bin`, which contains many other useful utilities and programs. They all have Commando interfaces that present the choices available when using them. Descriptions of the files in the `/mac/bin` directory can be found in `/FILES`. Some of the facilities available in `/mac/bin` are described here:

- Two utilities handle the transition between the standard A/UX environment and an A/UX Toolbox application environment.

  `startmac` provides the standard 32-bit A/UX Toolbox environment

  `startmac24` provides the 24-bit A/UX Toolbox environment

  The 24-bit environment is isolated from the rest of the system, and is provided for testing when converting 24-bit applications and for running obsolescent 24-bit utilities and tools, such as orphaned compilers.

- A utility is available for launching a Macintosh binary from the command line of CommandShell.

  `launch`(1) provides special options and capabilities for a Macintosh binary launched within the A/UX Finder. This utility is not necessary for ordinary launching, which can be done by double-clicking or opening an executable Macintosh binary. The utility is provided for convenience in launching from CommandShell and for use in application development. The `launch` utility functions only in the 32-bit environment.

- Three utilities support file conversion and setting of file attributes.

  `setfile` sets file creator and type, and other attributes

  `changesize` changes the value of the file's `'SIZE'` attribute. A slightly higher value than strictly necessary for the Macintosh MultiFinder environment is suggested for an application running under the A/UX finder. The additional space allows for overhead that the Memory Manager uses when dealing with the A/UX virtual memory environment on the application's behalf.

  `fcnvt` converts files from and to six formats. This utility is described in detail later in the next section.

- Two utilities are available for compiling and decompiling resources.

  `rez` compiles resources

  `derez` decompiles resources

These two utilities are described in detail later in this chapter, and extensive additional information is in Appendix E, "Resource Compiler and Decompiler." The descriptions in this chapter are also in Section 1 of *A/UX Command Reference*.

Information on utilities is available from many sources, including online and printed man pages and Commando dialogs. For the utilities `fcnvt`, `rez`, and `derez`, additional information follows.

fcnvt

Function

Convert a file from one format to another. The six formats available are AppleSingle, AppleDouble, Plain Triple, Plain Pair, BinHex 4.0, and MacBinary.

Syntax

`fcnvt` [`-i` input-format] [`-o` output-format] [`-f`] input-file output-file

`fcnvt` [`-i` input-format] `-s` [`-f`] input-file output-file

`fcnvt` [`-i` input-format] `-d` [`-f`] input-file output-file

`fcnvt` [`-i` input-format] `-t` [`-f`] input-file output-file

`fcnvt` [`-i` input-format] `-p` [`-f`] input-file output-file

`fcnvt` [`-i` input-format] `-b` [`-f`] input-file output-file

`fcnvt` [`-i` input-format] `-m` [`-f`] input-file output-file

Description

`fcnvt` converts a file (*input-file*) from one file format to another (*output-file*). The command-line options and their meanings are:

`-i` *input-format*

Specify the file format of the file to be converted. If an input-file format is not specified, the AppleSingle format is assumed. Supported formats are:

`single`
AppleSingle (see the `-s` flag option)

`double`
AppleDouble (see the `-d` flag option)

`triple`
Plain Triple (see the `-t` flag option)

`pair`
Plain Pair (see the `-p` flag option)

`hex`
BinHex 4.0 (see the `-b` flag option)

`bin`
MacBinary (see the `-m` flag option)

`-o` *output-format*

Specify the output-file format. The formats are the same as the input-file formats listed above. If an output-file format is not specified, the input file is converted to AppleSingle format.

`-s`

Create an AppleSingle-format output file. This format is the default. AppleSingle combines both the resource and data forks into a single file and is most useful when the resource fork seldom or never changes or when a file has no data fork. The use of AppleSingle format is very inefficient when both the resource and data forks are frequently expanded.

`-d`

Create an AppleDouble-format output file. The AppleDouble format is the same as AppleSingle format except that the data fork is kept in a separate file. The resource file is prefixed with a `%` character.

`-t`

Convert the input file into Plain Triple file format. This format is used by the `macget` and `macput` public-domain file-transfer programs. Three files are created with descriptive suffixes attached. The files *output-file*`.info`, *output-file*`.data`, and *output-file*`.rsrc` contain the input file's identification information, data fork, and resource fork, respectively.

`-p`
Convert the input file into Plain Pair file format. This option is the same as the `-t` option except that *output-file*`.info` is not created.

`-b`
Create a BinHex 4.0-format output file. The input-file is encoded into ASCII characters, permitting ASCII transfer of a binary file.

`-m`
Create a MacBinary-format output file. This format is commonly used when transferring files using XMODEM, XMODEM7, Kermit, and CompuServe A or B protocols.

`-f`
Allow `fcnvt` to overwrite an existing file with the new file of the same name. If you specify an output filename that is the same as an existing filename without specifying the `-f` flag option, `fcnvt` takes no action and returns an error message.

AppleSingle provides a safeguard for executable Macintosh object files, in contrast to AppleDouble format—the AppleDouble pair must be kept in the same directory in order to function.

`fcnvt` can also be used to help complete the transfer of a file from the native Macintosh environment to A/UX. As described in Chapter 6, in the sub-section titled "Automatic Conversion," when a Macintosh file is placed in the A/UX environment it is automatically converted to one of three formats: AppleSingle, AppleDouble, or Plain. The resultant format depends on various conditions. `fcnvt` may then be used to convert to a different format.

If the initial transfer is made using a terminal emulator program, the file created is likely to be in a text-only format, BinHex 4.0 format, or MacBinary format, if not just a copy of the resource fork of the Macintosh file. In any of those cases, `fcnvt` can be used to convert the file to another format, assuming you know the starting format.

Note that file transfers made using terminal emulators are likely to strip away the Macintosh type and creator attributes for the file. (Each of these attributes is one four-character string.) These attributes may be restored, once you know what they are supposed to be, by using `setfile`, which is described in this chapter (and in the man pages).

FILES

```
/mac/bin/fcnvt
```

SEE ALSO

Chapter 6, "File Systems and File Formats," has additional details of the AppleSingle, AppleDouble, and Plain file formats.

---

rez

Function

Compile resources.

Syntax

`rez` [*-option...*] [*resource-description-file...*]

Description

`rez` creates a resource file according to a series of statements in the resource description language developed for Macintosh resources. Appendix E, "Resource Compiler and Decompiler," describes the resource description language in detail.

The parameter *resource-description-file* represents the names of one or more files containing resource descriptions. If no filenames are specified, `rez` accepts keyboard input.

The data used to build the resource file can come directly from one or more resource description files, from other text files (through `#include` and `#read` directives in the resource description file), and from other resource files (through the `include` directive in the resource description file). The type declarations for standard Macintosh resources are contained in the files `types.r` and `systypes.r` in the directory `/mac/lib/rincludes`.

`rez` includes macro processing, full expression evaluation, and built-in functions and system variables.

`rez` never sends output to standard output. By default, `rez` writes to an AppleDouble header file named `rez.Out` in the current directory. You can specify a different output file with the `-o` option.

If no errors or warnings are detected, `rez` runs silently. Errors and warnings are written to standard error.

`rez` returns one of the following status values:

0 No errors
1 Error in parameters
2 Syntax error in file
3 I/O or program error

You may specify one or more of the following options for *-option*:

`-align (word)`
`-align (long word)`

Align resources on word or long word boundaries (without the `-align` option, resources are byte-aligned). This may allow the Resource Manager to load these resources faster. The `-align` option is ignored when the `-a` option is in effect.

`-a[ppend]`

Append the output of `rez` to the output file instead of replacing the output file.

▲ Warning `rez` overwrites any existing resource of the same type and ID without a warning message. `rez` cannot append resources to a resource file in which the `Read Only` bit is set. Also, `rez` does not replace a resource file that has a protection bit set unless the `-ov` option is specified. ▲

`-c[reator]` *creator-expr*

Set the creator of the output file to *creator-expr*. The default value is `'????'`.

`-d[efine]` *macro*[`=`*data*]
: Define the macro variable *macro* to have the value *data* (see "Preprocessor Directives" in Appendix E for a description of macro variables). If *data* is omitted, then *macro* is set to the null string (note that this still means that *macro* is defined). The `-d` option is the same as writing

  `#define` *macro*[*data*]

  at the beginning of the input. The `-d` option may be repeated any number of times.

`-i` *pathname*(*s*)
: Search the specified *pathname* (*s*) for include files. You may specify more than one pathname. The paths are searched in the order they appear in the command line.

  Resource include files are provided with the A/UX Toolbox at this pathname:

  `/mac/lib/rincludes`

`-o` *output-file*
: Place the output in *output-file*. Specify the name of the associated data file; `rez` automatically prefixes a percent sign to the name of the header file containing the resources. The default output file is `rez.Out`.

`-ov`
: Override the protected bit when replacing resources with the `-a` option.

`-p[rogress]`
: Write version and progress information to diagnostic output.

`-rd`
: Suppress warning messages if a resource type is redeclared.

`-ro`
: Set the `mapReadOnly` flag in the resource map.

`-s` *pathname*(*s*)
: Search the specified *pathname* (*s*) for resource include files.

`-t[ype]` *type-expr*
: Set the type of the output file to *type-expr*. The default value is `'APPL'`.

`-u[ndef]` *macro* "Undefine" the macro variable *macro* to remove the definition; this is the opposite of the `-d` option. (See "Preprocessor Directives" in Appendix E for a description of macro variables.) This is the same as writing

`#undef` *macro*

at the beginning of the input. It is meaningful to "undefine" only the preset macro variables.

Example

```
rez -i /mac/lib/rincludes -o sample sample.r
```

Generate a resource file for `sample`, based on the descriptions in `sample.r` and the include files in `/mac/lib/rincludes`. Place the output in an AppleDouble header file named `%sample`.

---

derez

Function Decompile a resource file.

Syntax `derez` [*-option...*] *resource-file* [*resource-description-file...*]

Description `derez` creates a text representation (resource description) of a compiled resource file according to the resource type declarations in the resource description file.

The parameter *resource-file* specifies the name of the file containing the compiled resources. You must specify a file; `derez` never reads standard input. The file may be either an AppleSingle or an AppleDouble file. (Do not prefix the filename with a percent sign.)

The parameter *resource-description-file* specifies one or more files containing the type declarations used by the resource file. `derez` can provide more meaningful output if you provide the type declarations.

The type declarations in the resource description file follow the same format as that used by the resource compiler, `rez`. The type declarations for standard Macintosh resources are contained in the files `types.r` and `systypes.r` in the directory `/mac/lib/rincludes`. If you do not specify a resource description file, the output consists of `data` statements giving the resource data in hexadecimal form, without any additional format information.

The resource description is written to standard output.

If the output of `derez` is used as input to `rez`, with the same resource description files, it produces the same resource file that was originally given to `derez`. When `derez` does not have the appropriate resource description information, it is not able to run a declaration backward. It produces a `data` statement instead of the appropriate `resource` statement.

`derez` ignores all `include` (but not `#include`), `read`, `data`, and `resource` statements found in the resource description file. (It still parses these statements for correct syntax.) Appendix E, "Resource Compiler and Decompiler," describes the format of resource type declarations.

The resource description consists of `resource` and `data` statements that can be understood by `rez` (see Appendix E, "Resource Compiler and Decompiler").

If no errors or warnings are detected, `derez` runs silently. Errors and warnings are written to standard error. See `intro`(3) in *A/UX Programmer's Reference*.

`derez` returns one of the following status values:

0 No errors
1 Error in parameters
2 Syntax error in file
3 I/O or program error

You may specify one or more of the following options for *-option*:

`-c[ompatible]` Generate output that is backward compatible with `rez` in Release 1.0.

`-d[efine]` *macro*`[=`*data*`]`

Define the macro variable *macro* to have the value *data* (see "Preprocessor Directives" in Appendix E for a description of macro variables). If *data* is omitted, then *macro* is set to the null string (note that this still means that *macro* is defined). The `-d` option is the same as writing

```
#define macro[data]
```

at the beginning of the input. The `-d` option may be repeated any number of times.

`-e[scape]`

Do not escape characters that are normally escaped (such as `\0xff`). Instead, print these characters as extended Macintosh characters. Normally characters with values between 0x20 and 0xD8 are printed as Macintosh characters. With the `-e` option, however, all characters (except null, newline, tab, backspace, formfeed, vertical tab, and rubout) are printed as characters, not as escape sequences. *Note:* Not all fonts have defined all the characters.

`-i` *pathname* (*s*)

Search the specified *pathname* (*s*) for include files. You may specify more than one pathname. The paths are searched in the order they appear on the command line.

Resource include files are provided with the A/UX Toolbox at this pathname:

```
/mac/lib/rincludes
```

`-m[axstringsize]` *n*

Set the maximum string size to *n,* which must be in the range 2–120. This setting controls how wide strings can be in the output.

`-o[nly]` *type-expr*`[(`*ID1*`[:`*ID2*`])]`
`-o[nly]` *type-expr*`[(`*resource-name*`)]`

Read only resources of resource type *type-expr.* If *ID,* a range of ID numbers, or a *resource-name* is given, read only those resources for the given type. This option may be repeated.

Note: The placeholder *type-expr* is an expression, so straight single quotes (') might be needed. If *ID*, a range of ID numbers, or a *resource-name* is given, the entire option parameter must be quoted. A type-expression may have any of these forms:

type

"'*type*' (*ID*) "

"'*type*' (*ID*:*ID*) "

"'*type*' (∂"*name*∂") "

Note: The ∂ character, obtained by pressing OPTION-D, is necessary for proper interpretation of the following double quote (") character.

This option can be used as shown here:

```
derez -only "'MENU'(1:128)" ...
```

See also "Examples" later in this section.

Note: The `-only` option cannot be specified together with the `-skip` option.

`-o[nly]` *type* Read only resources specified by *type*. This is a simpler version of the preceding option. No quotes are needed to specify a literal type as long as it starts with a letter. Do not use escape characters or other special characters. This option can be used as shown here:

```
derez -only MENU ...
```

`-p` Display progress and version information.

`-rd` Suppress warning messages if a resource type is redeclared.

`-s[kip]` *type-expr*[(*ID1*[:*ID2*])]
`-s[kip]` *type-expr*[(*resource-name*)]
Skip resources of type *type-expr* in the resource file. For example, you can save execution time by skipping `'CODE'` resources. If *ID*, a range of ID numbers, or *resource-name* is given, skip only those resources for the given type. The `-s` option may be repeated any number of times.

Note: The placeholder *type-expr* is an expression, so straight single quotes (') might be needed. If an *ID*, a range of ID numbers, or *resource-name* is given, the entire option parameter must be quoted. See the note under `-only` *type-expr* earlier in this section.

`-s[kip]` *type* Skip resources specified by *type*. This is a simpler version of the `-s` option. No quotes are needed to specify a literal as long as it starts with a letter.

`-u[ndef]` *macro*

"Undefine" the macro variable *macro* to remove definition; it is the opposite of the `-d` option). This is the same as writing

`#undef` *macro*

at the beginning of the input file (see "Preprocessor Directives" in Appendix E for a description of macro variables). It is meaningful to "undefine" only the preset macro variables. This option may be repeated.

Examples

```
derez -i /mac/lib/rincludes sample types.r > sample.r
```

Decompile the resource file `%sample`, by using the definitions in the file `/mac/lib/rincludes/types.r` and putting the output in the file `sample.r`. If `derez` has access to the type definitions, it generates more meaningful output.

```
derez -o MENU -i /mac:lib:rincludes sample types.r
```

Display all of the `'MENU'` resources in `%sample`. The type definition for `'MENU'` resources is in the file `types.r`.

A/UX Toolbox variables

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

- `dontForeground`
  This variable specifies whether or not the program runs in the background. If set to 1, the program runs in the background.

  To set `dontForeground` to 1, include this line in your program:

  ```
  int dontForeground = 1;
  ```

- `noCD`

  This variable sets the current directory. If 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.

  To set `noCD` to 1, include this line in your program:

  ```
  int noCD = 1;
  ```

Additional traps and routines

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. The first is described below and the second under "Using `select` to Monitor both A/UX I/O Activity and Macintosh Events."

AUXDispatch trap

The AUXDispatch trap is a multipurpose call that supports some A/UX-specific extensions to the A/UX Toolbox. The call should only be invoked after using SysEnviron to determine that the application is running under A/UX.

The definitions for the AUXDispatch trap are in the header file aux.h, found in /usr/include/mac. (See "Definitions for A/UXDispatch" in Appendix F.) The header file provides a syntax compatible with MPW C version 3.0.

AUXDispatch takes 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:

AUX_HIGHEST — Return the highest available selector (for support of future releases, which may provide more selectors).

With this selector, the value of the pointer *p* is not used.

AUX_GET_ERRNO — Get a pointer to errno, which is linked to your program through the standard C library.

AUXDispatch puts the address of errno in the address you specify with the pointer *p.*

AUX_GET_PRINTF — Get a pointer to the printf(3S) routine, which is linked to your program through the standard C library.

AUXDispatch puts the address of printf in the address you specify with the pointer *p.*

`AUX_GET_SIGNAL`

Gets a pointer to the `signal`(3) routine, which is linked to your program through the standard C library.

`AUXDispatch` puts the address of `signal` in the address you specify with the pointer *p*.

`AUX_GET_TIMEOUT`

Return a time period, in clock ticks, when the next Macintosh device driver will need processor time through the `SystemTask` routine. See "The Desk Manager" in *Inside Macintosh,* Volume I, for a description of `SystemTask`.

With this selector, the value of the pointer *p* is not used.

`AUX_SET_SELRECT`

Set a rectangle that the user interface device driver will use to monitor mouse movements for the `select`(2N) system call. For an explanation of using the `select` call, see the section "Using `select` to Monitor Both A/UX I/O Activity and Macintosh Events," later in this chapter.

With this selector, the pointer *p* points to the specified rectangle.

`AUX_CHECK_KIDS`

Check for the existence of child processes, returning 1 if child processes exist for the specified process, 0 if not.

With this selector, you specify the process to be checked for child processes by passing a pointer to the process ID in the pointer *p*.

`AUX_POST_MODIFIED`

Post an event, with modifiers. With this selector, pass a pointer to the Event Record in the pointer *p*.

`AUX_FIND_EVENT`

Searches event queue for an event. With this selector, pass a pointer to a `FindEvent` struct (mask and pointer to an Event Record) in the pointer *p*.

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 (see `environ`(5) in *A/UX Programmer's Reference*). This section lists both the environment variables used by the A/UX Toolbox and their functions.

`TBCORE` If this variable is set, 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. A typical fatal error during development of A/UX Toolbox applications is an unimplemented A-line trap.

`TBRAM` If this variable is set, 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 path of the directory that contains Macintosh system files. The default setting is `/mac/lib/SystemFiles`.

`TBTRAP` If this variable is set, the system writes debugging information to standard error every time an A-line trap is executed.

`TBWARN` If this variable is set, 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 wish to set `TBWARN` in `.login` or `.profile`.

Making A/UX system calls

This section describes a strategy for using A/UX system calls in an application that will be built under the Macintosh OS and result in a binary file that can be executed in both environments.

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 could 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 `SysEnviron` to determine which system the application is running under.

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.**

   Here is how to do it:

   - 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);
     }
     ```

   - Compile the program.

   - 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.**

   You need to check the value of bit 9 in the global variable `HWCfgFlag`, a 16-bit word at memory location 0x0B22. See *Macintosh Technical Note #37,* available through APDA, for a description of `HWCFgFlag`; see "Information Sources" in Appendix A for the address of APDA).

   If bit 9 is 1, your application is running under A/UX. Use the appropriate A/UX code, including any assembly-language routines that you have created to replace A/UX system calls.

   If bit 9 is 0, your application is running under the Macintosh OS. Use the appropriate Macintosh OS or User Interface Toolbox calls.

Using `select` to monitor both 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 communication 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, simply 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 of a specified rectangle. (Ordinarily, mouse motion without the pressing or releasing of the mouse button is not an event.) First, issue the `AUXDispatch` call, by 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 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 under "Event Manager, Toolbox" in Chapter 5).

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 later required). MacsBug comes with a reference manual. See "Information sources" in Appendix A for information on APDA.

MacsBug does not underly the entire system, as when used with the Macintosh OS. Pressing the hardware programmer switch when A/UX is running brings you into an A/UX environment. To get into MacsBug, you use a key command.

To install MacsBug, place it here:

```
/mac/sys/System Folder/MacsBug
```

This is the equivalent of placing it in the System folder under the Macintosh OS. MacsBug will automatically install itself at the next login or reboot.

MacsBug will be invoked when the system encounters an exception error. MacsBug can be accessed from an application whenever desired, by pressing this key command:

Command	Result
`Cmd-Cntl-I`	Interrupt into MacsBug

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

The following command kills the A/UX Toolbox and reboots into A/UX.

Command	Meaning
`Cmd-Cntl-E`	Reboot

This command can be useful to get out of a hung system. It does a general tidying up of the mess before logging you out. If you press this key combination when MacsBug is not installed, a similar reboot takes place, but not so tidy.

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

Command	Result
`g`	Go; continue from current location
`rs`	Restart; equivalent to logout
`rb`	Reboot; equivalent to logout
`es`	Exit to shell; kills current application
	This is not a good way to exit. Your get out leaving Low-memory globals in a strange state, and after a while strange thing start to happen to your other applications.
`dm curapname`	Display current application name
	The current application may not be what you think it is; this command is worth trying before you kill the current application.

Chapter 4 **Compatibility Checklist**

To run the same code under both the Macintosh OS and the A/UX environments, you must make provisions for a number of compatibility issues:

- differences between the Macintosh OS and A/UX execution environments
- differences between the C compilers used in A/UX and 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

This chapter provides a checklist of the compatibility issues. For details about the A/UX implementation of the Macintosh ROM code, see Chapter 5, "Inside A/UX Macintosh," 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 A/UX.

In A/UX, the kernel arbitrates all access to hardware, including memory allocation. Only the kernel can use the hardware instructions of the MC680x0 microprocessor. Table 4-1 provides information on support for privileged instructions.

Unlike the current Macintosh OS, system software Release 6.0.5, A/UX uses virtual memory.

This section lists the compatibility problems that result from the differences between the Macintosh OS and A/UX execution environments. This section augments the Macintosh programming guidelines provided in *Inside Macintosh,* Volumes I through V. 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 V.

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 SysEnvirons or check the variable `CPUFlag`, described in Appendix D, "Low-Memory Global Variables." Eventually, this information will be available by use of the Gestalt facility.

32-bit address violations

A/UX uses all 32 bits of an address, but the Macintosh OS formerly 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. For present and future Macintosh OS applications, being 32-bit clean is now required 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 special 24-bit environment is provided by means of a special login so that programmers upgrading an application to 32-bit clean may test in both environments.)

To have 32-bit clean applications, use the Memory Manager in a safe manner. Programming according to the brief guidelines below helps 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.
- Check that a handle marked purgeable is not empty before using that handle.

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

- Creating or using fake handles

  A fake handle is one that was not made 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.

- Direct access to the flag bits on relocatable blocks

  The Macintosh OS stores 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 only 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 "Memory Manager" in Chapter 5.

- 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.

- 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 (see *Inside Macintosh,* Volume V).

Privileged microprocessor instructions

The A/UX Toolbox is run by an A/UX process in MC680x0 User Mode. Therefore, most privileged processor instructions are not available within an A/UX Toolbox application. 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	

[continued]

■ **Table 4-1** Privileged microprocessor instructions within the A/UX Toolbox [continued]

Instruction	Register and addressing modes supported
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 supported
STOP	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 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 (see *Inside Macintosh,* Volumes II and V) and the Time Manager to schedule routines that require precise timing (see *Inside Macintosh,* Volume IV). 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,* 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. See Chapter 3, "A/UX Toolbox Utilities and Extensions," for a strategy on including A/UX system calls in applications that are intended to run under both the Macintosh OS and A/UX.

Newline characters

The A/UX Toolbox supports the transfer of files between the Macintosh OS and A/UX. 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, 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 in the Macintosh side of the system and is now on the A/UX side will find the expected newline characters.

Automatic newline conversion happens for Macintosh OS files identified as text files, and for A/UX files that are determined to be text or shell script files.

See the section, "Text Files," and "Automatic conversion," in Chapter 6, "File Systems and File Formats," 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 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 in the Macintosh side of the environment, then the correct newline character will be present.

Why mention this at all, if automatic conversion takes care of it? Because programmers 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. A resource file contains text and non-text matter. No conversion is done for resource files. A binary file transferred between the two environments has its original definition of newline when running in the new environment, which may affect both the output of text and the processing of input text. An application intended to execute in both environments may need to determine its execution environment and select the desired newline character accordingly.

File Manager

The A/UX File Manager almost fully supports the Macintosh OS File Manager. For details about support, 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 the Macintosh OS Memory Manager and does it within a virtual memory environment. Virtual memory does have practical limits and performance limits. In general, performance degrades to an unacceptable level unless all of the memory that is actively being used fits into physical memory.

Differences in C compilers

This section lists the known differences between the A/UX C compiler 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 compiler generates warnings when it sees zero-length arrays, which appear frequently in the A/UX Toolbox header files.

- Newline characters

  The newline character is return (ASCII value 0x0D) in MPW C and line feed (ASCII value 0x0A) in A/UX C.

- Pascal function types

  MPW C has an `extern pascal` function type 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_s.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.*

  However, if you want to create your own definition functions or filter functions (see *Inside Macintosh,* Volume I, for explanations of definition functions and filter functions), you must generate your own assembly-language glue.

  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.

- 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 `int` 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 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 alphabetic list of all calls in the C interface libraries, "Calls in Alphabetic Order," is available in Appendix F. That list can be consulted to determine whether an alternate version of a call is available, because lowercase and mixed-case versions of a call name sort together.

Chapter 5 **Inside A/UX Macintosh**

This chapter describes the differences between the A/UX Toolbox and 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, however, not *Inside Macintosh* order.

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 through the A/UX Toolbox as they work in the standard Macintosh OS. This chapter describes the small differences between the two versions.

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.

Table 5-1 summarizes the status of the various ROM libraries at the time A/UX Release 2.0 was distributed.

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

ROM library	Supported?
32-Bit QuickDraw with Color QuickDraw	Yes
Apple Desktop Bus	No
AppleTalk Manager	Partially
Binary-Decimal Conversion Package	Yes
Color Manager	Yes
Color Picker Package	Yes
Control Manager	Yes
Deferred Task Manager	No
Desk Manager	Yes
Device Manager	Yes
Dialog Manager	Yes
Disk Driver	Yes

[continued]

■ **Table 5-1** The Status of User Interface Toolbox and Macintosh OS libraries in the A/UX Toolbox [continued]

ROM library	Implemented?
Disk Initialization Package	Yes
Event Manager, Operating System	Partially
Event Manager, Toolbox	Yes*
File Manager	Partially
Floating-Point Arithmetic and Transcendental Functions Packages	Yes*
Font Manager	Yes
Gestalt facility	Partially
International Utilities Package	Yes
List Manager Package	Yes
Memory Manager	Yes
Menu Manager	Yes
Notification Manager	Yes
Package Manager	Yes
Palette Manager	Yes
Printing Manager	Yes
Resource Manager	Yes
Scrap Manager	Yes
Script Manager	Partially
SCSI Manager	No
Segment Loader	Partially
Serial Driver	Partially
Shutdown Manager	Yes*
Slot Manager	Partially
Sound Manager	Partially
Startup Manager	Not needed
Standard File Package	Yes
System Error Handler	Yes*

[continued]

■ **Table 5-1** The Status of User Interface Toolbox and Macintosh OS libraries in the A/UX Toolbox [continued]

ROM library	Implemented?
TextEdit	Yes
Time Manager	Yes*
Utilities, Operating System	Partially
Utilities, Toolbox	Yes
Vertical Retrace Manager	Partially
Window Manager	Yes

* All calls are implemented, but functionality is not identical. 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 defines, types, and functions provided by the library. Appendix F, "C Interface Library," contains an expanded version of the header files, with comments and function parameters added. 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, IV, and V, for a description of QuickDraw and Color QuickDraw, supplemented by the APDA document on 32-Bit QuickDraw. See "32-Bit QuickDraw" in Appendix F for the A/UX C interface.

Apple Desktop Bus

The A/UX Toolbox does not support the Macintosh OS Apple Desktop Bus.

AppleTalk Manager

AppleTalk printing operations are supported under A/UX. AppleTalk network calls are available at the program level, but not fully supported at the network level: there is no session protocol or data stream.

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 Macintosh binary calls. The equivalent A/UX C header files are in the library `/usr/include/at`.

See *Inside Macintosh,* Volumes II and V, 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 V, 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 in A/UX is that a control's variant code is not stored in the `contrlDefProc` 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.

Deferred Task Manager

The Deferred Task Manager is not implemented in the A/UX Toolbox. A call to the `DTInstall` routine generates an `unimplemented trap` message.

Desk Manager

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

See *Inside Macintosh,* Volume I, for a description of the Desk Manager. See "Desk Manager" in Appendix F for the A/UX C interface.

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 A/UX.

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 "Video 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 after an error, it ignores the `resumeProc` 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 returns (`\r`) in C.

- `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 and V, for a description of the Dialog Manager. See "Dialog Manager" in Appendix F for the A/UX C interface.

Disk Driver

The Disk Driver is supported.

See *Inside Macintosh,* Volumes II and IV, 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,* Volume II, 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.

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. See *Inside Macintosh,* Volumes II and IV, for a description of the Operating System Event Manager. See "Event Manager, Operating System" in Appendix F for the A/UX C interface. Also, see "`A/UX Dispatch` 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 the Operating System Event Manager routines for manipulating the queue. The `AUXDispatch` call `AUX_FIND_EVENT` may be used 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`. An application cannot depend on a patch to `PostEvent` to alert it to mouse and key keyboard events. The `A/UXDispatch` 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 A/UX than under the Macintosh OS. See *Inside Macintosh,* Volumes I 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 multitasking environments such as the A/UX Finder. Do not use `GetNextEvent`. The call takes this syntax:

```
Boolean WaitNextEvent (mask,*event,sleep,mouseRgn)
      unsigned short mask;
      struct EventRecord *event;
      unsigned long sleep
      RgnHandle mouseRgn;
```

The parameters `mask` and `event` are the same as in the `GetNextEvent` call (described in *Inside Macintosh,* Volume I). `mouseRgn` specifies a screen region. Screen regions are used to keep track of mouse movements for such purposes as changing cursor appearance as appropriate for the current location of the cursor. The application receives a mouse-moved event (one of the `App4` events in the Macintosh OS) only when the mouse strays outside the specified region. For more information on the `WaitNextEvent` call, see *Programmer's Guide to MultiFinder,* available through the APDA (see Appendix A "Additional Reading").

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

- The global variable `Ticks` is not incremented in A/UX. To find out the current tick count, call the `TickCount` routine. The `-t` option to the `launch`(1) utility creates and maintains the `Ticks` variable, but this option absorbs a lot of processor time.
- The `GetNextEvent` and `WaitNextEvent` functions do not support FKEY resources. This means they do not perform the special functions tied to COMMAND-SHIFT-*numberkey* combinations, such as ejecting the disk in internal drive 0 when the user presses COMMAND-SHIFT-1.
- Because the global variables `KeyThresh` and `KeyRepThresh` are ignored, an application cannot change key-repeat characteristics.
- Journaling is not supported.

File Manager

The A/UX Toolbox File Manager supports the Macintosh OS File Manager.

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

The underlying support for the File Manager is provided by UFS, an implementation of the BSD 4.2 (Berkeley System Distribution) file 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 HFS name is 32 characters, and longer names brought into the Macintosh OS environment are truncated.

For information on the Macintosh OS File Manager, see Chapter 19, "The File Manager," in *Inside Macintosh,* Volume IV.

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

See *Inside Macintosh,* Volume II, 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 is identical to the Macintosh OS Font Manager.

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

Gestalt facility

The A/UX Toolbox Gestalt facility provides limited support for the Macintosh OS Gestalt. The Gestalt facility is under development at present, and subject to change. The following environmental selector is available and may 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 is available, use the `TrapAvailable` function. The Gestalt trap address is $A1AD. For example, use the following:

```
FUNCTION GestaltAvailable : Boolean;
CONST
      _Gestalt = $A1AD;
BEGIN
      _GestaltAvailable := TrapAvailable(_Gestalt)
END;
```

Documentation on the Macintosh OS Gestalt is in development. See "Gestalt facility" in Appendix F for the A/UX C interface.

International Utilities Package

The A/UX Toolbox fully supports the Macintosh OS International Utilities Package.

See *Inside Macintosh,* Volumes I and V, 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.

The Memory Manager is 32-bit clean and expects to serve applications that are 32-bit clean. If you need to run an application that uses the older form of 24-bit addressing, there is a special 24-bit environment available in Release 2.0 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 Chapter 1, "The Memory Manager," in *Inside Macintosh,* Volume II, which is intended to be read in conjunction with related chapters in Volumes I and V. 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 (see *Inside Macintosh,* Volume I, for a description of definition functions), you must provide assembly-language routines to transform the parameters into Pascal format for compatibility with the ROM (see Chapter 4, "Compatibility Checklist," and Appendix C, "Implementation Notes").

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 the *Programmer's Guide to MultiFinder* for information on 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 interfacing 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 V, for a description of the Palette Manager. See "Palette Manager" in Appendix F for the A/UX C interface.

Printing Manager

The A/UX Toolbox Printing Manager is identical to the Macintosh OS Printing Manager. Release 2.0 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.

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 and V, for a description of the Resource Manager. See "Resource Manager" in Appendix F for the A/UX C interface. Related information on the resource compiler, `rez`, and resource decompiler, `derez`, is available under those titles in Chapter 3. Appendix E, "Resource Compiler and Decompiler," provides further information.

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

- Resource files

  A/UX files in AppleDouble format store resources and data in separate files. See Chapter 2, "Using the A/UX Toolbox," and Chapter 6, "File Systems and File Formats," for descriptions of Macintosh file formats in A/UX. Be careful to keep both files together when copying, renaming, or otherwise manipulating files.

- Write permission

  Your application might not have write permission in the directories containing the System Folder (typically `/mac/lib/System Files`) or the application.

- Case-sensitive filenames

  Unlike the standard Macintosh OS, A/UX differentiates between uppercase and lowercase in filenames. Be careful with the filenames in `OpenResFile` and `CreateResFile`.

- Search paths

  The standard Macintosh 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. The alternative search paths are not supported in 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, whose address appears in Appendix A, "Additional Readings."

  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,* Volume IV.

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 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 is 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,* Volume V, 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.

You can reproduce the functions of the SCSI Manager in A/UX by writing your own routines that use A/UX system calls. 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 A/UX.

For an application that is intended to run only under A/UX, you can write an A/UX device driver. See *Building A/UX Device Drivers.* A program that uses an A/UX device driver is not portable to the Macintosh OS.

Segment Loader

Applications in the standard Macintosh development environment are written in segments, which are loaded individually as needed to use memory 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 and IV, 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 under both environments may 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.
- A Macintosh binary 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 A/UX. The Segment Loader routines not listed here are implemented exactly as described in *Inside Macintosh.*

`UnloadSeg`	Performs normally for binary applications that are launched; stubbed out for native A/UX applications
`ExitToShell`	Issues an `exit`(2) call
`_Chain`	Not implemented
`_Launch`	Not implemented

`_LoadSeg`	Performs normally for 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 applications.

Summary of the Segment Loader

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

`FinderName` is always the null string.

Serial Driver

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

- Asynchronous I/O is not supported.
- Three baud rates are not supported: 3600, 7200, and 57600. These rates are mapped to 2400, 4800, and 19200, respectively. This affects control calls 8 (`SerReset`) and 13 (`baudRate`).
- Control call 9 (`SerSetBuf`) has no effect. When called, it just returns.
- Event message posting is not supported. This affects control calls 10 and 14 (`SerHShake`). If the `evts` field in the `SerShk` record is non-zero, 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. They are

`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`. `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.) `Ser_cts` indicates the current status of the CTS signal; TRUE indicates CTS ON (High). `Ser_inflow` is TRUE if input is currently blocked due to flow control. `Ser_outflow` is TRUE iF output is blocked due to flow control. The data structure is:

```
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 line (drives it high) for the serial line denoted by *fd*.

`ioctl(fd,TCCLRDTR,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 board must support these eight calls in the driver's ioctl routine if the Serial Manager is to work properly. The second argument to the ioctl routine is the *cmd* argument that contains the ioctl command, such as `TCRESET`.

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, but not necessarily with identical behavior.

`ShutDwnInstall` and `ShutDwnRemove` perform as described in *Inside Macintosh,* Volume V. Both `ShutDwnPower` and `ShutDwnStart` first execute any routines in the queue and then exit to the user's login shell.

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:

```
SRsrcInfo
SNextRsrc
SNextTypesRsrc
SReadDrvrName
```

- Specialized routines are supported except for one routine:

```
SReadByte
SReadWord
SReadLong
SGetcString
SGetBlock
SFindStruct
SReadStruct
SReadInfo
SReadPRAMRec
SPutPRAMRec
SReadFHeader
SCkCardStatus
SFindDevBase
SDeleteSRTRec          Not supported
SPtrToSlot
```

- Advanced routines are supported as follows:

```
InitSDeclMgr           Not supported
SPrimaryInit           Not supported
SCardChanged
SExec                  Not supported
SOffsetData
SReadPBSize
SCalcStep
InitsRsrcTable         Not supported
InitPRAMRecs           Not supported
SSearchSRT
SUpdateSRT
SCalcSPointer
SGetDriver             Not supported
SFindsInfoRecPtr       Not supported
SFindsRsrcPtr
```

- Assembly-language routine selectors are supported as follows:

`sReadByte`	
`sReadWord`	
`sReadLong`	
`sGetcString`	
`sGetBlock`	
`sFindStruct`	
`sReadStruct`	
`sReadInfo`	
`sReadPRAMRec`	
`sPutPRAMRec`	
`sReadFHeader`	
`sNextRsrc`	
`sNextTypesRsrc`	
`sRsrcInfo`	
`sDisposePtr`	Not supported
`sCkCardStatus`	
`sReadDrvrName`	
`sFindDevBase`	
`InitSDeclMgr`	Not supported
`sPrimaryInit`	Not supported
`sCardChanged`	
`sExec`	Not supported
`sOffsetData`	
`InitPRAMRecs`	Not supported
`sReadPBSize`	
`sCalcStep`	
`InitsRsrcTable`	Not supported
`sSearchSRT`	
`sUpdateSRT`	
`sCalcsPointer`	
`sGetDriver`	Not supported
`sPtrToSlot`	
`sFindsInfoRecPtr`	Not supported
`sFindsRsrcPtr`	
`sDeleteSRTRec`	Not supported

See *Inside Macintosh,* Volume V, 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, as documented in the Technical Note of October 2, 1988, mentioned below. 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, as in shell scripts.

The exceptions to full support for the Sound Manager are

- only one sampled channel, instead of two.
- maximum sampling rate is 22K, rather than 44K.
- no tickle commands.
- for the `_SndAddModifier` trap, no `addMod` command for synthesizer modules.

  The reason for this is 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.

MIDI synthesizers are not supported in Macintosh OS 6.0.5 or A/UX Release 2.0.

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 may happen under the Macintosh OS or A/UX.

The sample folder contains demonstration and sample programs that use the Sound Manager.

The description of the Sound Manager in *Inside Macintosh,* Volume V, has been replaced by the Technical Note titled "The Sound Manager," by Jim Reekes, dated October 2, 1988. See "Sound Manager" in Appendix F for the A/UX C interface.

Support details

Here are the details on trap support.

- Commands sent normally only by the Sound Manager are supported with one exception.

`nullCommd`	
`initCmd`	Supported for note, wave, and sampled synthesizers
`freeCmd`	
`quietCmd`	
`flushCmd`	

- Sync controls are fully supported.

`waitCmd`
`pauseCmd`
`resumeCmd`
`callBackCmd`
`syncCmd`
`emptyCmd`

- This `noteSynths` only command is supported.

`timbreCmd`

- The `waveTableSynth` only commands are supported.

`waveTableCmd`
`phaseCmd`

- The `sampledSynth` only commands are supported.

`soundCmd`
`bufferCmd`
`rateCmd`
`continueCmd`

- Synthesizer resource IDs with `SndNewChannel` are partially supported. (No Midi commands.)

`noteSynth`	
`waveTableSynth`	
`sampledSynth`	
`midiSynthIn`	Not supported
`midiSynthOut`	Not supported

- Init options for `SndNewChannel`, sampled synthesizer only, are partially supported.

`initChanLeft`	Ignored, defaults to mono
`initChanRight`	Ignored
`initSRate22k`	
`initSRate44k`	Not supported, defaults to 22K
`initMono`	
`initStereo`	Not supported, defaults to mono

- Init options for `SndNewChannel`, wave table synthesizer only, are supported.

```
initChan0
initChan1
initChan2
initChan3
```

The Raw Sound Driver

The Raw Sound Driver is available to A/UX without calling upon the Sound Manager (for use in shell scripts, and so forth), as

```
/dev/snd/raw
```

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

```
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 using the `echo` command:

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

The sampling rate of the Raw Sound Driver is 22K by default. To change the rate, use an ioctl to the same device. The ioctl sends a structure to the raw sound driver. 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 22K maximum rate. The value of `$00010000`, meaning `$1.0000`, preserves the default rate. Here is an example setting the rate to 7K. Calculate the multiplier:

7K/22K = .318 (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.

```
#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 almost identical to the Macintosh OS Standard File 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 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, 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 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 is identical to the Macintosh OS TextEdit. 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, and V, 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 IV, 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 queueing and trap vector routines are fully functional.
- The miscellaneous utilities `Delay`, `SysBeep`, and `Environs` are fully functional.
- 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 when the program started running.

Because setting the system clock through either the `date`(1) command or the `stime`(2) call requires A/UX superuser status, you cannot change the clock setting through 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`, `SysBeep`, and `Environs` are fully functional.

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

The `StripAddress` routine always returns the pointer unchanged.

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,* Volume II, are implemented in A/UX. None of the Vertical Retrace Manager routines described in *Inside Macintosh,* Volume V, are implemented.

The A/UX Vertical Retrace Manager routines are implemented using 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 in 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.

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

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

Chapter 6 **File Systems and File Formats**

This chapter describes how the file systems in A/UX and in the Macintosh OS differ. Users and applications may access files from either the A/UX environment or from the A/UX Toolbox that supports a Macintosh OS environment, as convenient, and files may be transferred between the two environments by users and by applications, without any special requirements. The chapter describes the results of the automatic conversion that is done when files cross the boundary between the two environments, and the formats used for storing Macintosh files in A/UX.

File systems

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

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

One theme of these general file systems is 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 (SVFS, UFS, NFS, and so on). The Macintosh OS file system provides equivalent functionality for files in volumes under its control. The discussion in this chapter will stay at the high-level view, except for discussion 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 designated by slash (/), the root directory. The one volume may contain multiple file systems. A file system, in this technical sense, is a combination of routines for manipulating files together with associated data structures and 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 interested in having such a structure.

In the UNIX design, subordinate file systems may be added to or removed from the one volume only by 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, or was, associated with a different physical device. (As the Macintosh OS developed in scope, it became possible to partition a large storage device into more than one volume.) 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 may 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 (A/UX) or folders (Macintosh OS). Directories and folders are functionally equivalent. A directory or folder may 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 specific filename. (Pathnames for a directory or folder end with a directory or folder.) In A/UX, the full pathname starts with the root volume (`/`); in the Macintosh OS, 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 OS pathname uses the colon (`:`). Here is an example of each type:

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

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

A Macintosh filename consists of any sequence of 1 to 31 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 the Berkeley File System, which is now the default system accessed via the Toolbox, A/UX filenames may consist of 1 to 255 eight-bit characters (slashes excluded).

When comparing filenames, the A/UX file system distinguishes between upper-case and lower-case versions of a character; the Macintosh file system does not distinguish 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 may be used in an A/UX filename, practical considerations suggest that a space should never be used. For example, suppose a user saves a text file under the name `my report` and attempts to access it under A/UX 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 file name or the space, as follows:

```
vi "my report"
vi my\ report
```

Because blanks are 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.

Access permissions

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

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

What happens to the file permissions when an A/UX file is transferred to the Macintosh OS file system? Conversely, when a file is transferred from the Macintosh OS to the A/UX file system, how are the permissions assigned? In brief, UNIX (A/UX) permissions and AppleShare permissions are mapped from and to each other, as appropriate.

Table 6-1 shows how A/UX permissions are mapped to Macintosh OS.

■ **Table 6-1** A/UX permissions mapped to Macintosh OS permissions

A/UX Permissions			Macintosh OS Permissions		
Read	**Write**	**Execute**	**Folders**	**Files**	**Changes**
No	No	No	No	No	No
No	No	Yes	No	No	No
No	Yes	No	No	No	No
No	Yes	Yes	No	No	Yes
Yes	No	No	No	No	No
Yes	No	Yes	Yes	Yes	No
Yes	Yes	No	No	No	No
Yes	Yes	Yes	Yes	Yes	Yes

Under the A/UX Finder, Macintosh OS (AppleShare) permissions may be set. Table 6-2 shows how these Macintosh permissions are mapped to A/UX. Setting access permissions on the root (/) volume from the A/UX Finder is not supported.

■ **Table 6-2** Macintosh OS permissions mapped to A/UX permissions

Macintosh OS Permissions			A/UX Permissions		
Folders	**Files**	**Changes**	**Read**	**Write**	**Execute**
No	No	No	No	No	No
No	No	Yes	No	Yes	Yes
No	Yes	No	Yes	No	Yes
No	Yes	Yes	Yes	Yes	Yes
Yes	No	No	Yes	No	Yes
Yes	No	Yes	Yes	Yes	Yes
Yes	Yes	No	Yes	No	Yes
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—but no special repository of information about the file. However, 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 it 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 is for display 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 in later sections. Here is a brief summary:

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

Text files

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

- Lines are terminated by return characters (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:

- Lines are terminated by line feeds (ASCII value 0x0A).
- The text file has no associated type or creator information. When such a file enters the A/UX Finder environment, it receives a file type and creator based on the rules described in the next section, "Automatic Conversion." If all goes well, an A/UX text file will get a file type of `'TEXT'` and a creator that is user selected.

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 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 returns. 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 terminations will be line feeds. All this is essentially transparent to the user.

Automatic conversion

When a file is transferred from one file system to the other, as by dragging, 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, creator, and flag setting.

The process is summarized in Table 6-3.

■ **Table 6-3** Automatic conversion of Macintosh files

These attributes:			
Type	**Creator**	**Flag setting**	**Result in this 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 6-3 shows, there are three ways to ensure conversion to Apple Double format.

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

The first way allows programs (such as CommandShell) to have their own icon while ensuring AppleDouble format. The second and third ways allow files that may have INITs to be AppleDouble. 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. Access permissions are lost. Text files and shell files have their line termination characters automatically translated from line feed to return characters, as described under "Text files," earlier in this chapter.

Mounting and unmounting floppy disks

Under A/UX (or any UNIX system), the file system can recognize a file structure on a floppy disk and grant appropriate access only if the disk is mounted. The mounting 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.

Under the Macintosh file system, users are accustomed to inserting and removing floppy disks as convenient or desirable. 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 does not require an unmount operation.

To prevent grave problems, 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 the previous section, "Automatic Conversion," individual Macintosh files may be placed under A/UX, after which the file becomes some variety of A/UX file.

An application may, via 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.

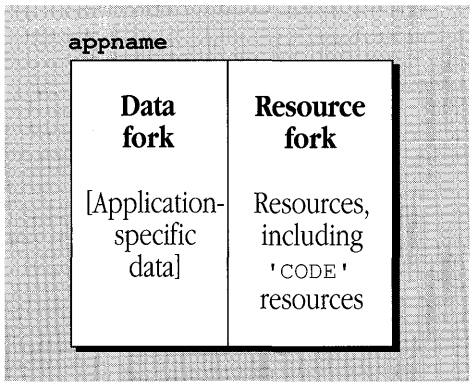
In short, when an application is running under the A?UX Toolbox, all floppy disk files that it "sees" (can access) are on Macintosh volumes. It does not "see" any A/UX files on floppy disks except by access through the A/UX file system.

Storing files in the Macintosh OS and in A/UX

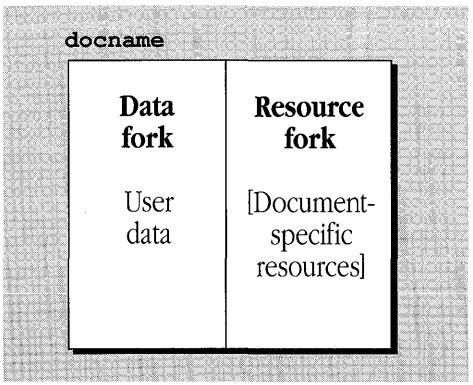
In the Macintosh OS, a file consists of two forks: a data fork and a resource fork. In general, the data fork contains user data, such as the text in a word-processing document, and the resource fork contains resources used by the application. Resources include commonly used structures such as screen dialogs 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 might 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, the file that holds an executable application may contain only resources, with an empty data fork. Figure 6-1 illustrates the elements of a file in 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



Macintosh application in Macintosh OS



Document created by Macintosh application in the Macintosh OS

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 with UNIX utilities, remember to move both files.

The internal formats of AppleSingle and AppleDouble files are discussed in "AppleSingle and AppleDouble Format Internals," in the next section of 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 an application from having to consider these details.

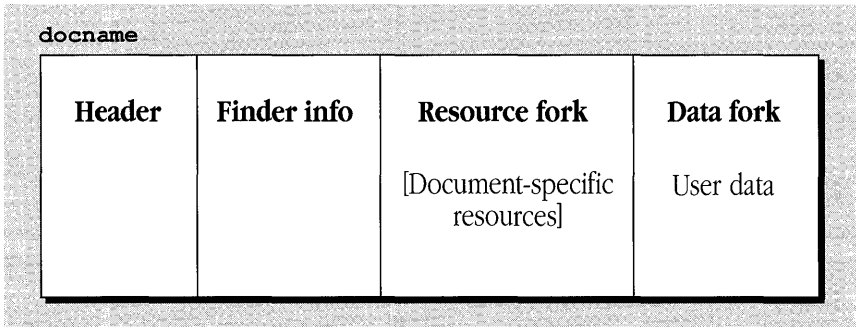
An A/UX file, standing alone, remains a Plain file but is recognized as an AppleDouble data file. (This allows A/UX Toolbox applications to access files created by conventional UNIX utilities, such as text editors.) An A/UX Toolbox application processing a Plain file may cause the creation of a header 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," earlier in this chapter. If the combination of file type and creator is changed to anything other than a 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 might 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.

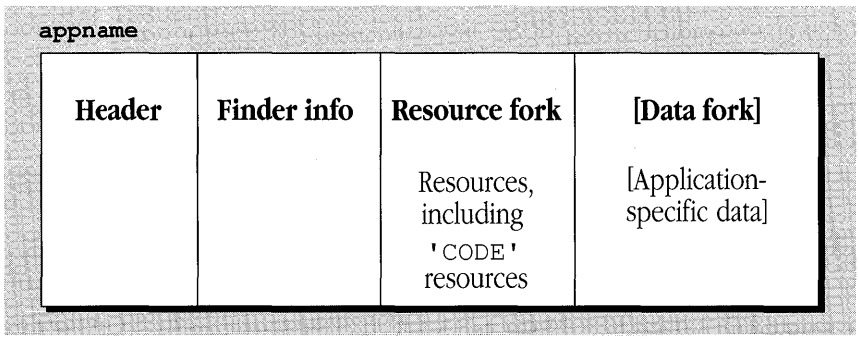
An AppleDouble data file is accompanied by a header file containing the file attribute information. The header file can—but need not—contain resources. An AppleDouble header file can exist without an associated data file.

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.

■ **Figure 6-2** Typical contents of an AppleSingle file

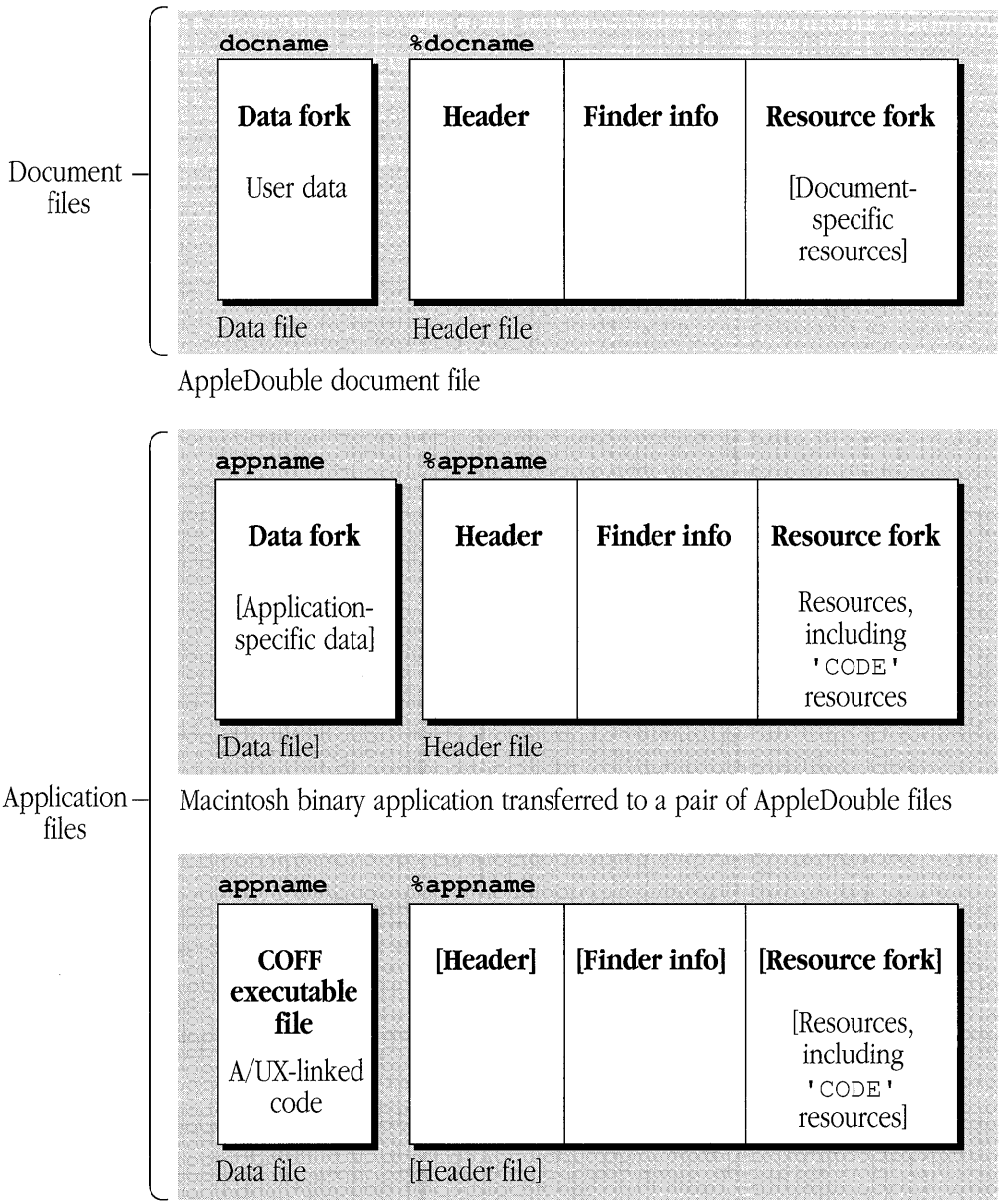


AppleSingle document file



Macintosh binary application transferred to an A/UX AppleSingle file

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

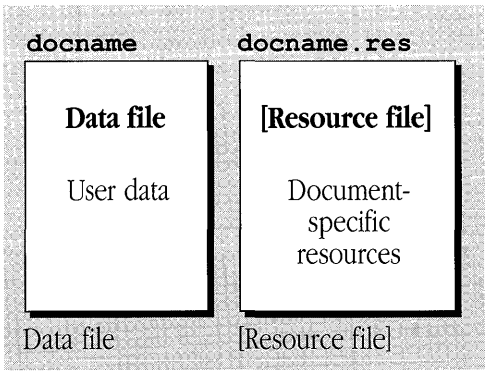


When you compile and link an application under A/UX, the result is a standard executable Common Object File Format (COFF) file. The Macintosh OS will consider the COFF file to be an AppleDouble file. As mentioned above, so long as the type remains `'COFF'` and the creator `'A/UX'`, no unnecessary header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

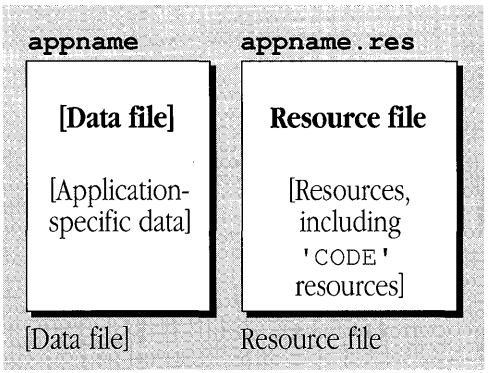
If you have used a general-purpose utility to transfer files from the Macintosh OS to A/UX, you might also have Macintosh files stored in a simple A/UX format. The `kermit`(1C) utility, for example, transfers the two forks of a Macintosh file separately into a pair of A/UX files that follow neither AppleSingle 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 Section 1C of *A/UX Command Reference* for a description of `kermit`.) For compatibility with other tools, the A/UX Toolbox file-conversion utility, `fcnvt`(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.

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



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

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 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 utilities for converting files and manipulating their formats:

- The `fcnvt`(1) utility converts files among AppleSingle format, AppleDouble format, and four other formats.
- The `settc`(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.

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 the 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. Macintosh 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. A/UX is the foreign file system 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 6-4 describes the contents of an AppleSingle file header.

■ **Table 6-4** 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. The fields follow these conventions:

- 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 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 ...
VAX/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. It 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 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 comment
Icon, B&W	5	Standard Macintosh black-and-white icon
Icon, color	6	Reserved for Macintosh color icon
File Info	7	File information: attributes, and so on
Finder Info	9	Standard Macintosh Finder Info

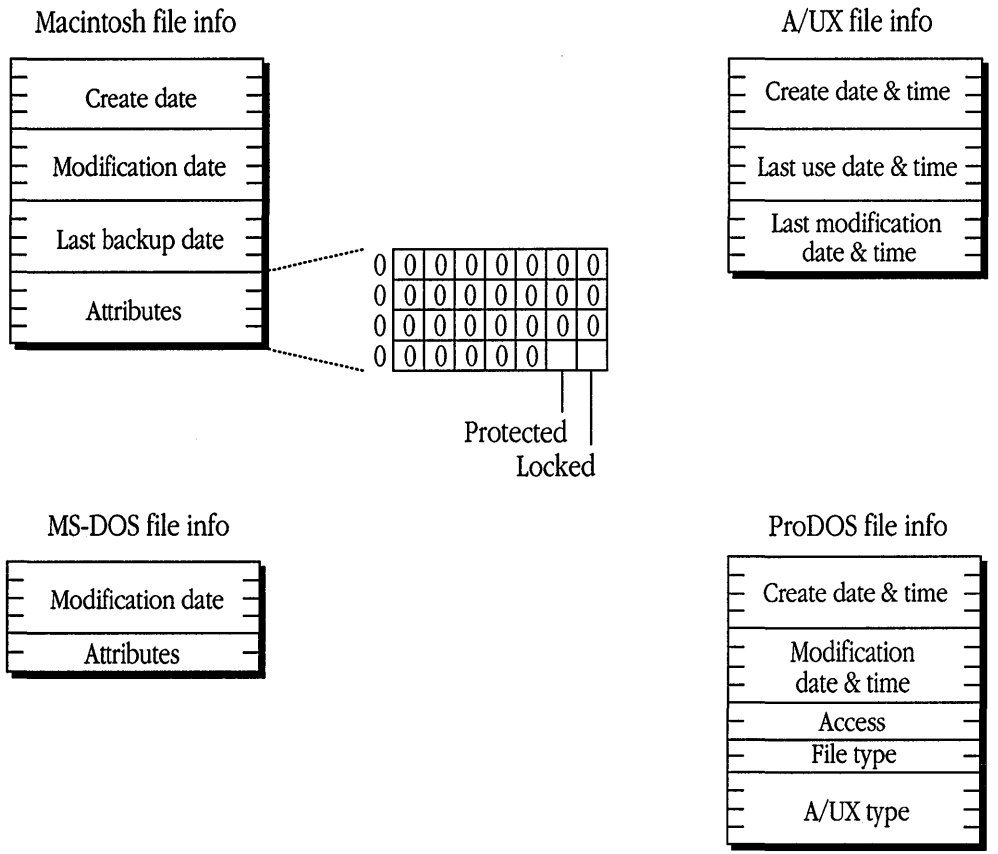
  Apple reserves the range of entry IDs from 0 to 0x7FFFFFFF. The rest of the range is available for other definitions. Apple does not arbitrate the use of the rest of the range.

  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 field consists of 16 bytes of Finder Info followed by 16 bytes of extended Finder Info (the fields `ioFlFndrInfo` followed by `ioFlXFndrInfo`, as returned by the `PBGetCatinfo` call). These fields contain what we have been calling "extended file attribute information." See *Inside Macintosh,* Volume IV, 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 any subfield whose value is unknown, except that you should set the `fdType` and `fdCreator` subfields. Values should be set using standard File Manager calls such as `SetFInfo` and `PBSetCatInfo`.

■ **Figure 6-5** Formats for File Info entries



- **Offset:** This field, an unsigned 32-bit number, shows the offset of the beginning of the entry's data from the beginning of the file.
- **Length:** This field, an unsigned 32-bit number, 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.

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 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

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

- Embedded spaces transfer. Such filenames are legal but cause problems in A/UX because A/UX commands consider a space to delimit a filename. Getting rid of the embedded spaces is preferable.
- Use character substitution to replace any illegal characters 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 of up to 255 characters.

AppleSingle format

The overall AppleSingle format does not specify an algorithm for deriving the AppleSingle filename from the file's "real" name as stored on a native Macintosh volume. Applications and file systems 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.

AppleDouble format

The general strategy for AppleDouble format filenames is to derive the data filename from the file's original Macintosh name and then to derive the header filename from the data filename. 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 filename, the general considerations apply.
- For an AppleDouble header filename, prefix a single percent sign (%) to the AppleDouble data filename. If necessary, truncate the last character to keep the filename within the legal length range. This is the convention followed by A/UX. The result is that the two files are kept together in a single subdirectory.

Appendix A **Additional Reading**

This appendix lists books and other information sources that will help you use the A/UX Toolbox. The first section lists information sources, the second section lists manuals that provide necessary information, and the third section lists supplementary information that may be useful.

Information sources

For programmers and developers who work on Apple equipment, the Apple Programmers and Developers Association (APDA) provides a wide range of technical products and documentation from Apple and other suppliers. For information about APDA, contact

Apple Computer, Inc.
Apple Developer Channels
20525 Mariani Avenue, Mailstop 33G
Cupertino, CA 95014-6299
1-800-282-APDA or 1-800-282-2732
Fax: 408-562-3971
Telex: 171-576
AppleLink: DEV.CHANNELS

Apple offers two developer programs, Apple Associates and Apple Partners. The Associates program is for those who are involved in development, but not engaged in commercial development, such as researchers, MIS professionals, consultants, and distributors. The Partners program is for those engaged in commercial development of hardware and software and in associated endeavors, such as VARs, developers, book authors, software publishers, OEMs, DSVARs, and contract programmers. Both programs provide information and support. For further information, write to

Apple Developer Programs
Apple Computer, Inc.
20525 Mariani Avenue, Mailstop 51W
Cupertino, CA 95014-6299

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, Mailstop 51M
Cupertino, CA 95014-6299

Required references

This list contains books that you will need for developing software under the A/UX system or interfacing hardware to a Macintosh computer running the A/UX system. *Road Map to A/UX*, listed below, provides a detailed description of each A/UX book published by Apple Computer, Inc.

A/UX Command Reference. Apple Computer, Inc., 1990. A collection of reference pages for A/UX user commands and games. This document corresponds to Sections 1 and 6 of the traditional UNIX user manual.

A/UX Essentials. Apple Computer, Inc., 1990. A user's introduction to A/UX Release 2.0.

A/UX Programmer's Reference. Apple Computer, Inc., 1990. A collection of reference pages 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.

A/UX Programming Languages and Tools, Volumes 1 and 2. Apple Computer, Inc., 1987 and 1990. A description of the A/UX C and Fortran languages and the libraries and tools used for program development and maintenance.

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

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

Programmer's Guide to MultiFinder. APDA, 1988. A guide to writing applications compatible with MultiFinder; applicable to the A/UX Finder.

Road Map to A/UX. Apple Computer, Inc., 1990. A guide to the features of A/UX and to the A/UX documentation.

Supplementary references

This list contains some useful books available for developing software under, or interfacing hardware to, the A/UX system. The list is not exhaustive. You can find many other excellent books on various aspects of developing under System V UNIX, developing with shell language, and making use of BSD features. Similarly, you can find other useful books on developing under the Macintosh Toolbox and Macintosh OS. The Macintosh Programmer's Workshop (MPW) references document a UNIX-like development environment running under the Macintosh OS.

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

Macintosh Technical Notes. Apple Computer, Inc., 1984–1990. A set of technical bulletins distributed at no charge by Apple Computer, Inc., to all affiliated developers. Available through APDA at Apple Computer, Inc.

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 OS. Later editions have tracked developments of the Macintosh OS.

Designing Cards and Drivers for Macintosh II and Macintosh SE. Addison-Wesley, 1987. A general reference for developing expansion cards and device drivers for the Macintosh II and Macintosh SE.

Harbison, Samuel P., and Guy L. Steele, Jr. *C: A Reference Manual.* Prentice-Hall, Inc., 1984. A standard reference book for the C language with the Western Electric extensions used in most UNIX operating system environments.

Hogan, Thom. *The Programmer's Apple Mac Sourcebook.* Microsoft Press, 1989. Reference tables and charts for Macintosh OS system software and Macintosh hardware, gathered and collated from eleven source volumes: *Inside Macintosh,* Volumes I through V, *Programmer's Introduction to the Macintosh Family, Technical Introduction to the Macintosh Family, Designing Cards and Drivers for Macintosh II and Macintosh SE, Inside Macintosh X-Ref, Hypercard Script Language,* and several hardware reference books. Includes data formats, resource formats, data structures, Pascal calls by manager, and much additional information, conveniently collated.

Inside Macintosh X-Ref. Addison-Wesley, 1988. A key to eight of the Addison-Wesley books that document the Macintosh: *Inside Macintosh,* Volumes I through V, *Programmer's Introduction to the Macintosh Family, Technical Introduction to the Macintosh Family,* and *Designing Cards and Drivers for Macintosh II and Macintosh SE.* 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 Dennis M. Ritchie. *The C Programming Language.* Prentice-Hall, Inc., 1984. An update of the original, official C manual, with tutorial information.

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.

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.

Macintosh Programmer's Workshop 3.0 C Reference. APDA, 1988. A description of the 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 for A/UX are closely linked.

Macintosh Programmer's Workshop 3.0 Pascal Reference. APDA, 1988. A description of the Pascal Compiler and tools.

Macintosh Programmer's Workshop 3.0 Reference. APDA, 1988. A full description of how to use the MPW program preparation tools.

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, Inc., 1985. A description of the MC68020 CPU in detail for hardware and software engineers.

MC68030 32-Bit Microprocessor User's Manual. Motorola, Inc., 1987. A description of the MC68030 CPU in detail for hardware and software engineers.

MC68851 Paged Memory Management Unit User's Manual. Motorola, Inc., 1985. A description of the PMMU in detail for hardware and software engineers.

MC68881 Floating-Point Coprocessor User's Manual. Motorola, Inc., 1985. A description of the instruction set and addressing conventions used by the MC68881 floating-point coprocessor, which is used in the Macintosh II.

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 Macintosh Toolbox and Macintosh OS features.

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

The large text file `/FILE` contains an annotated list of files in A/UX Release 2.0. This file may be explored for further information about the contents of directories listed here.

The list of files that follows is not exhaustive, but 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 online and printed man 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 doing login with the CommandShell and 24-bit CommandShell, files for executing the Commando function, and several utilities, some of which are discussed in Chapter 3.
`changesize`	Change `'SIZE'` attribute (Chapter 3)
`rez`	Resource compiler (Chapter 3, Appendix E)
`derez`	Resource decompiler (Chapter 3, App. E)
`fcnvt`	File conversion (Chapter 3)
`launch`	Launch a Macintosh binary application
`setfile`	Set file creator and type (Chapter 3)
`startmac`	Provide the A/UX Macintosh environment
`startmac24`	Provide the 24-bit A/UX Macintosh environment
`TextEditor`	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 dialogs
`/mac/lib/rincludes`	Contains resource header files (Chapter 3, 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 (Chapter 2)
`sndDemo`	Sound demonstration example application sources, resource, sound file, and makefile (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`	Access for 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
`errors.h`	System Error Handler
`events.h`	Toolbox Event Manager
`files.h`	File Manager
`fonts.h`	Font Manager
`gestalt.h`	Gestalt facility
`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
`printtraps.h`	Print traps
`quickdraw.h`	32-Bit QuickDraw with Color QuickDraw
`resources.h`	Resource Manager
`retrace.h`	Vertical Retrace Manager
`romdefs.h`	Definitions for ROMs
`scrap.h`	Scrap Manager
`script.h`	Script Manager
`segload.h`	Segment Loader
`serial.h`	Serial Driver
`shutdown.h`	Shutdown Manager
`slots.h`	Slot Manager
`sm.h`	Sound Manager
`strings.h`	String conversion routines
`sysequ.h`	Low-memory equates
`textedit.h`	TextEdit
`timer.h`	Time Manager
`toolutils.h`	Toolbox Utilities
`traps.h`	List of Macintosh traps
`types.h`	Type definitions
`video.h`	Video Driver
`windows.h`	Window Manager
`/usr/lib/libmr.a`	Declarations and routines for reading Macintosh resources

Appendix C **Implementation Notes**

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 it if you are writing an application that contains assembly-language routines or deviates from recommended Macintosh programming practices.

This appendix covers three areas:

- setting up the environment to run an A/UX Toolbox application
- running an A/UX Toolbox application
- 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 Toolbox application cannot be launched without support of the A/UX Finder.

A "MultiFinder aware" Macintosh OS application will execute, but a "MultiFinder friendly" application performs better. Developers should aim at that standard. A checklist of requirements for applications is in Chapter 4, "Compatibility Checklist."

Running an A/UX Toolbox application

The A/UX kernel contains a special user-interface device driver, `/dev/uinter0`, that handles communication between an A/UX Toolbox application and the kernel. The driver provides ioctl functions. (The "Serial Driver" discussion in Chapter 5 has an example of 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 instead of 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 the 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 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 4.2 BSD 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.
7. Calls the application's main routine.

A-line traps

The primary function of the A/UX Toolbox is to make available to programs running in 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 0xAFFF.

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 instead of 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's 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 may install its own patches to the tables.

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:

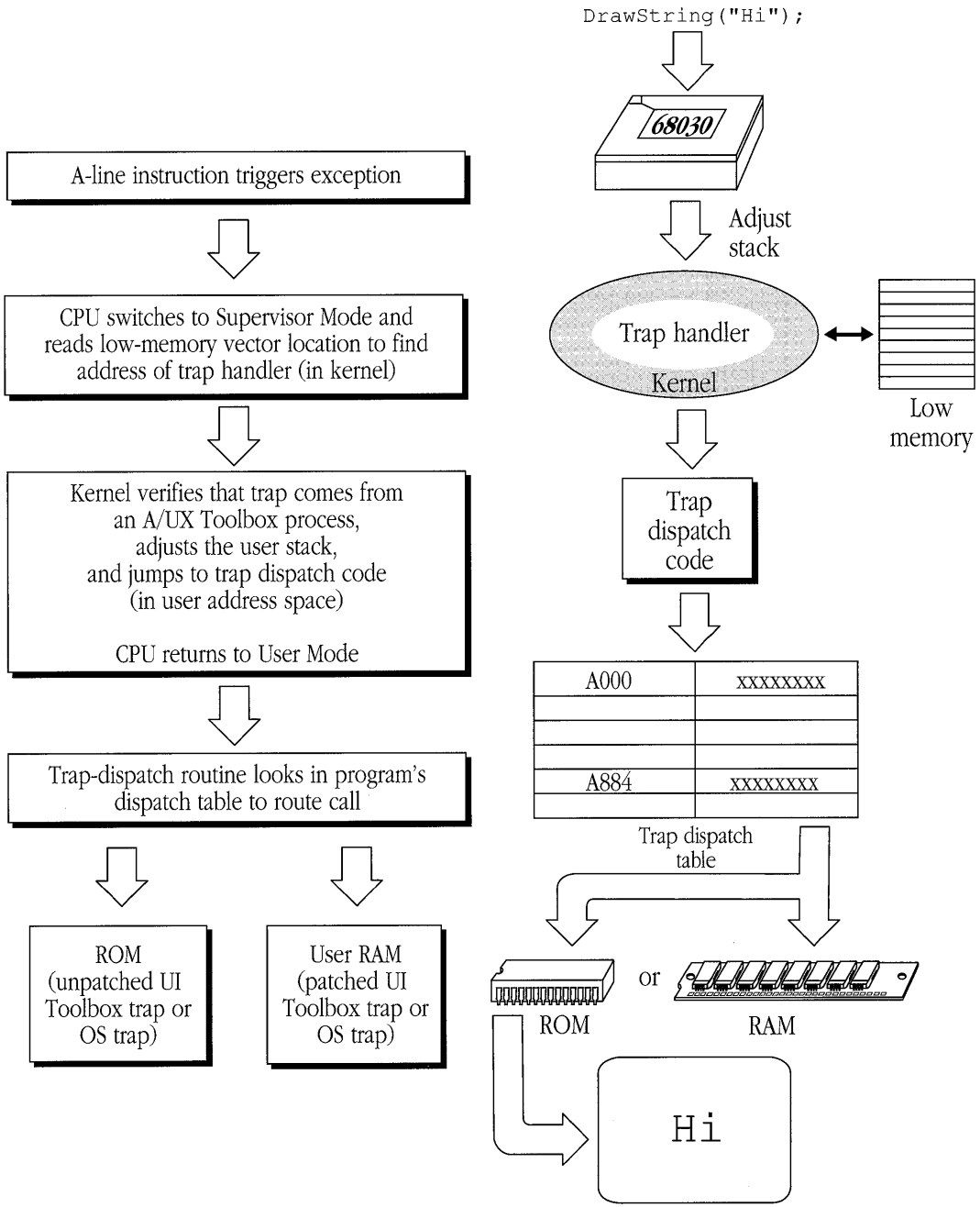
- ROM

  If the trap has no A/UX alternative, the table points to the ROM code.

- User RAM

  If the trap represents 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. Either version can be used 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 (see *Inside Macintosh,* Volumes III, IV, and V). To make room for these global variables, an A/UX Toolbox application compiled in 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 in 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 compiler.

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 values are passed, how function results are returned, and how registers are used. The remaining sections in this appendix describe 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 this release of the A/UX Toolbox include two versions of all routines that take strings or QuickDraw point values or 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 your own 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, strings are normally stored as an array of characters, of any length, terminated by the null byte ('\0'). In Pascal, strings start 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" 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` (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 left to right and are pushed onto the stack in the order they are evaluated. For example, with the function `foo(a,b)`, `a` is pushed first and then `b`.

Parameters in C functions are evaluated right to left and are pushed onto the stack in the order 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 types `char` or `unsigned char` are pushed as bytes. (This 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 types `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 QuickDraw point values

The QuickDraw point values are are a 4-byte structure, which the Pascal language always passes by value rather than by pointer, unless the structure is declared as a VAR. (This 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 QuickDraw point values put the address of the structure on the stack.

Returning function results

A/UX C functions return pointer values in register 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 types `char` or `unsigned char` are returned as bytes. (This 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 types `short` or `unsigned short` are returned as 16-bit values. `int` and `long` values and the remaining enumerated types are returned as 32-bit values. Pointers are returned as 32-bit addresses. Arrays may not 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 greater than 4 bytes, the caller pushes a pointer to a result space before pushing any parameters. If the structure is 4 bytes or less, 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 *Inside Macintosh,* Volumes III, IV, and V, in appendixes titled "Global Variables."

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 is follow by the 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
ABusDCE	0x2DC	Pointer to AppleTalk DCE
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]
BusErrVct	0x08	Bus error vector
CaretTime	0x2F4	Caret blink ticks [long]
ColLines	0x0C22	Screen vertical pixels[word]
CPUFlag	0x12F	$00=68000, $01=68010, $02=68020 (old ROM inits to $00)
CQDGlobals	0x0CCC	QuickDraw global extensions [long]
CrsrBusy	0x8CD	Cursor locked out? [byte]
CrsrDevice	0x89C	Current cursor device [long]
CrsrObscure	0x8D2	Cursor obscure semaphore [byte]
CrsrState	0x8D0	Cursor nesting level [word]
CrsrVis	0x8CC	Cursor visible? [byte]
CurApName	0x910	Name of application [STRING[31]]
CurApRefNum	0x900	Reference number of application's resource file [word]
CurDirStore	0x398	Save directory across calls to Standard File [long]
CurJTOffset	0x934	Current jump table offset [word]

[continued]

■ **Table D-1** General global variables [continued]

Name	Reference	Description
CurPageOption	0x936	Current page 2 configuration [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]
DSDrawProc	0x334	Alternate SysError draw procedure [pointer]
DSErrCode	0xAF0	Last system error alert ID
DskErr	0x142	Disk routine result code [word]
DSWndUpdate	0x15D	GNE not to paint behind DS AlertRect? [byte]
EjectNotify	0x338	Eject notify procedure [pointer]
ExpandMem	0x2B6	Pointer to expanded memory block
GetParam	0x1E4	System parameter scratch [20 bytes]
GZMoveHnd	0x330	Moving handle for grow zone [handle]
GZRootHnd	0x328	Root handle for grow zone [handle]
GZRootPtr	0x32C	Root pointer for grow zone [pointer]
HeapEnd	0x114	End of heap [pointer]
HiHeapMark	0xBAE	Highest address used by a zone below the stack pointer [long]
HiliteMode	0x938	Used for color highlighting
HiliteRGB	0x0DA0	RGB of hilite color [6 bytes]
HpChk	0x316	Heap check RAM code [pointer]
IAZNotify	0x33C	World swaps notify procedure [pointer]
IntlSpec	0xBA0	Pointer to extra international data [long]
JAllocCrsr	0x88C	Vector to routine that allocates cursor
JFetch	0x8F4	Fetch-a-byte routine for drivers [pointer]
JGNEFilter	0x29A	GetNextEvent filter procedure [pointer]
JIODone	0x8FC	IODone entry location [pointer]
JSetCCrsr	0x890	Vector to routine that sets color cursor [long]

[continued]

■ **Table D-1** General global variables [continued]

Name	Reference	Description
JStash	0x8F8	Stash-a-byte routine for drivers [pointer]
KbdType	0x21E	Keyboard model number [byte]
Key1Trans	0x29E	Keyboard translator procedure [pointer]
Key2Trans	0x2A2	Numeric keypad translator procedure [pointer]
KeyMap	0x174	Bitmap of the keyboard [2 longs]
KeypadMap	0x17C	Bitmap for numeric pad (18bits) [long]
KeyRepThresh	0x190	Key repeat speed [word]
KeyThresh	0x18E	Threshold for key repeat [word]
LastTxGDevice	0x0DC4	Copy of TheGDevice set up for fast text measuring [long]
Lo3Bytes	0x31A	Constant $00FFFFFF [long]
LoaderPBlock	0x93A	Parameter block for ExitToShell [10 bytes]
LoadTrap	0x12D	Trap before launch? [byte]
MainDevice	0x8A4	The main screen device [long]
MaskBC	0x31A	Memory Manager byte count mask [long]
MaskHandle	0x31A	Memory Manager handle mask [long]
MaskPtr	0x31A	Memory Manager pointer mask [long]
MBState	0x172	Current mouse button state [byte]
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]
MMDefFlags	0x326	Default zone flags [word]
MMU32bit	0x0CB2	Boolean reflecting current machine MMU mode [byte]
MMUFluff	0x0CB3	Fluff byte forced by reducing MMU mode to 32 bits [byte]
MonkeyLives	0x100	Monkey lives if >= 0 [word]
Mouse	0x830	Processed mouse coordinates [long]
NewCrsrJTbl	0x88C	Location of new cursor jump vectors
NMIFlag	0x0C2C	Flag for NMI debounce [byte]
OneOne	0xA02	Constant $00010001 [long]
PCDeskPat	0x20B	Desktop pattern (Top bit only! Others are in use.)
PortAUse	0x290	Bit 7: 1 = Port A not in use, 0 = Port A in use

[continued]

■ **Table D-1** General global variables [continued]

Name	Reference	Description
PortBUse	0x291	Port B use, same format as PortAUse
PortList	0x0D66	List of grafPorts
QDColors	0x8B0	Handle to default colors [long]
QDErr	0x0D6E	QuickDraw error code [word]
QDExist	0x8F3	QuickDraw is initialized [byte]
RestProc	0xA8C	Resume procedure (old name for ResumeProc) [pointer]
ResumeProc	0xA8C	Resume procedure from InitDialogs [pointer]
RGBBlack	0x0C10	The black field for color [6 bytes]
RGBWhite	0x0C16	The white field for color [6 bytes]
RndSeed	0x156	Random seed/number [long]
ROM85	0x28E	High bit is 0 for ROM version $75 (sic) and later [word]
ROMBase	0x2AE	ROM base address [pointer]
ROMMapHndl	0xB06	Handle of ROM resource map [long]
RowBits	0x0C20	Screen horizontal pixels [word]
SaveSegHandle	0x930	Segment 0 handle [handle]
ScrapCount	0x968	Validation byte [word]
ScrapEnd	0x980	End of scrap variables
ScrapHandle	0x964	Memory scrap [handle]
ScrapInfo	0x960	Scrap length (old name for ScrapSize) [long]
ScrapName	0x96C	Pointer to scrap name [pointer]
ScrapSize	0x960	Scrap length [long]
ScrapState	0x96A	Scrap state [word]
ScrapTag	0x970	Scrap filename [STRING[15]]
ScrapVars	0x960	Scrap Manager variables [32 bytes]
Scratch8	0x9FA	Scratch [8 bytes]
Scratch20	0x1E4	Scratch [20 bytes]
ScrDmpEnb	0x2F8	Screen dump enabled? [byte]
ScrDmpType	0x2F9	FF dumps screen, FE dumps front window [byte]
ScreenBytes	0x0C24	Total screen bytes [long]
ScreenRow	0x106	RowBytes of screen [word]
ScrHRes	0x104	Screen horizontal dots/inch [word]

[continued]

■ **Table D-1** General global variables [continued]

Name	Reference	Description
ScrnBase	0x824	Screen base [pointer]
ScrVRes	0x102	Screen vertical dots/inch [word]
SdVolume	0x260	Global volume (sound) control [byte]
SegHiEnable	0xBB2	0 to disable MoveHHi in LoadSeg [byte]
SerialVars	0x2D0	Asynchronous driver variables [16 bytes]
SEvtEnb	0x15C	Enable SysEvent calls from GNE [byte]
SFSaveDisk	0x214	Last vRefNum seen by Standard File [word]
SInfoPtr	0x0CBC	Pointer to Slot Manager information [long]
SlotQDT	0x0D04	Pointer to slot queue table
SoundActive	0x27E	Sound is active? [byte]
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[2 four-bit]
SPConfig	0x1FB	Configuration bits: 4-7 A, 0-3 B (see SPPortA, SPPortB)
SPFont	0x204	Default application font number minus 1 [word]
SPKbd	0x206	Keyboard repeat threshold in 4/60ths [2 four-bit]
SPMisc1	0x20A	Miscellaneous [1 byte]
SPMisc2	0x20B	Mouse scaling, sys startup disk, menu blink values [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 [byte]
SPVolCtl	0x208	Volume control [byte]
SrcDevice	0x8A0	Source device for stretchBits [long]
StkLowPt	0x110	Lowest stack as measured in VBL task [pointer]
SwitcherTPtr	0x286	Switcher's switch table
SysEvtBuf	0x146	System event queue element buffer [pointer]
SysEvtMask	0x144	System event mask [word]
SysParam	0x1F8	System parameter memory [20 bytes]
SysVersion	0x15A	Version number of RAM-based system [word]

[continued]

■ **Table D-1** General global variables [continued]

Name	Reference	Description
SysZone	0x2A6	System heap zone [pointer]
TableSeed	0x0D20	Seed value for color table ID's [long]
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]
UnitNtryCnt	0x1D2	Count of entries in unit table [word]
UTableBase	0x11C	Unit I/O table [pointer]
VBLQueue	0x160	VBL queue header [10 bytes]
VidMode	0x0C2E	Video mode (4=four-bit color) [byte]
VidType	0x0C2D	Video board type ID [byte]
WMgrCPort	0x0D2C	Window Manager color port
WWExist	0x8F2	Window Manager initialized? [byte]

■ **Table D-2** Window Manager global variables

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 global variables

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]
TEWdBreak	0xAF6	Default word break routine [pointer]
WordRedraw	0xBA5	Used by TextEdit RecalDraw [byte]

■ **Table D-4** Resource Manager global variables

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	Autoload feature [word]
ResReadOnly	0xA5C	Read-only flag [word]
RomMapInsert	0xB9E	Necessary to link map to ROM resources? [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]

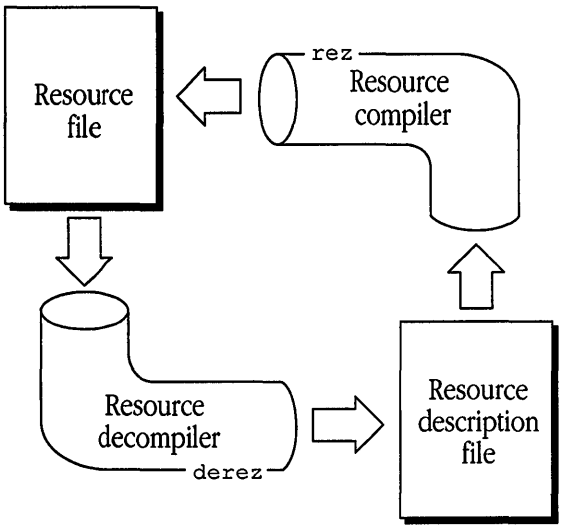
Appendix E **Resource Compiler and Decompiler**

This appendix explains how to build resources with the resource compiler, `rez`, and how to use the resource decompiler, `derez`. See Chapter 3, "A/UX Toolbox Utilities and Extensions," for the command-line syntax for `rez` and `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. (See "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 may 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

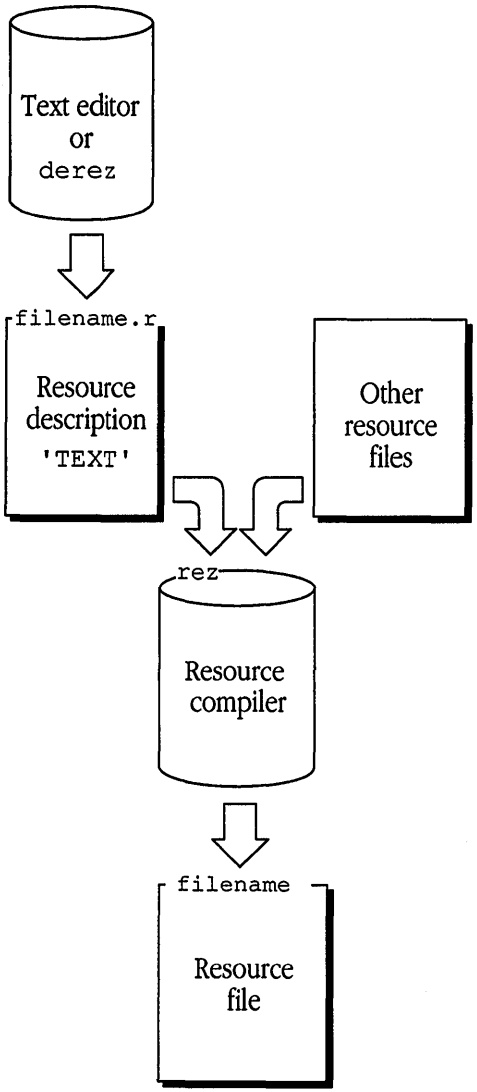
Four text files, `types.r`, `systypes.r`, `scripttypes.r`, and `pict.r`, contain resource declarations for standard resource types. These files are located in the directory `/mac/lib/rincludes`. They contain definitions for the following types:

`types.r`	Type declarations for the most common Macintosh resource types (`'ALRT'`, `'DITL'`, `'MENU'`, and so on)
`systypes.r`	Type declarations for `'DRVR'`, `'FOND'`, `'FONT'`, `'FWID'`, `'INTL'`, and `'NFMT'`, plus many others
`scripttypes.r`	Type declarations for resource descriptions specific to the Script Manager
`pict.r`	Type declarations for `'PICT'` resources for debugging `PICT`s

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:

`include`	Include resources from another file.
`read`	Read data file and include it as a resource.
`data`	Specify raw data.
`type`	Declare resource type descriptions for subsequent `resource` statements.
`delete`	Delete existing resources.
`change`	Change the type, ID, name, or attributes of existing resources.
`resource`	Specify data for a resource type declared in a previous `type` statement.

The section "Resource Description Statements" 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. 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/rincludes` 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: `include`, `read`, `data`, `type`, `delete`, `change`, and `resource`.

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 quotation marks, like this: `bitstring '['` *length* `']'`

In this case, the brackets are typed literally. They 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
resource-type	*long-expression*
resource-name	*string*
resource-ID	*word-expression*
ID-range	*ID*[*:ID*]

◆ *Note:* The placeholder *expression* is defined later in this appendix in "Expressions."

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

`include`—include resources from another file

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

Syntax and description

`include` statements can have the following forms:

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

Read the resource of type *resource-type* with the specified *resource-name, ID,* or range of ID numbers in *filename.* If both the resource name and the resource ID are omitted, read all resources of the type *resource-type* in *filename.* If *resource-type* is omitted, read all the resources in *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 *filename.*

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

Read all resources of type *resource-type1* and include them as resources of *resourcetype2.*

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

Read the resource of type *resource-type1* with the specified *resource-name1, ID,* or range of ID numbers in *filename,* and include it as a resource of *resource-type2* with the specified ID. You can optionally specify a *resource-name2* and new resource *attributes.* Resource attributes are defined in the next section "Resource Attributes."

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

`$$Type`	Type of resource from include file
`$$ID`	ID of resource from include file
`$$Name`	Name of resource from include file
`$$Attributes`	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 as a numeric expression (see 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:

Default	Alternative	Meaning
`appheap`	`sysheap`	The keyword specifies whether the resource is to be loaded into the application heap or the system heap.This attribute is meaningless if the resource is used only in A/UX.
`nonpurgeable`	`purgeable`	Purgeable resources can be automatically purged by the Memory Manager.
`unlocked`	`locked`	Locked resources cannot be moved by the Memory Manager.
`unprotected`	`protected`	Protected resources cannot be modified by the Resource Manager.
`nonpreload`	`preload`	Preloaded resources are placed in the heap as soon as the Resource Manager opens the resource file.
`unchanged`	`changed`	The keyword tells the Resource Manager whether a resource has been changed. `rez` does not allow you to set this bit, but `derez` 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

Read *filename* and write 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* (as defined in the preceding section). 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`.

`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

Read the data found in *data-string* and write 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 statement

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

reads the data string shown and writes it as a `'PICT'` resource with resource ID 128.

- *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 cannot be disassembled for some other reason.

`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

Cause 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 listed below are data types.

type-specification can be 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 in the following sections.

- *Note:* Several of these types require additional fields. The exact syntax is given in the following sections.

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 later in this appendix under "Expressions."

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`

  `decimal`

  `octal`

  `binary`

  `literal`

You can supply numeric data as hexadecimal, decimal, octal, binary, or literal data.

- *numeric-type* is one of the following:

`bitstring['`*length*`']`	Bitstring of *length* bits (maximum 32).
`byte`	Byte (8-bit) field. This is the same as `bitstring[8]`.
`integer`	Integer (16-bit) field. This is the same as `bitstring[16]`.
`longint`	Long integer (32-bit) field. This 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 these ranges:

Type	Maximum	Minimum
`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 Boolean type declaration

```
type  'DONE' {
      boolean = false;
};
```

declares a *symbolic-value* of FALSE.

Type `boolean` declares a 1-bit field; this is 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; this is 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:

`[hex] string`	Plain string (contains no length indicator or termination character). The optional `hex` prefix tells `derez` to display it as a hex-string. `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). `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 `pstring` (two leading bytes contain the length information). A `wstring` can contain up to 65,535 characters. `wstring[`*n*`]` contains *n* characters and is *n* + 2 bytes long.

`cstring` C string (a trailing null byte marks the end of the string). `cstring[n]` contains $n-1$ characters and is n bytes long. A `cstring` 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 $1 \leq$ *length* ≤ 2147483647 for `string` and `cstring`, and in the range $1 \leq$ *length* ≤ 255 for `pstring`, and in the range $1 \leq$ *length* ≤ 65535 for `wstring`.

◆ *Note:* You cannot assign the value of a literal to a *string-type.*

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 may 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 `cstring` is a termination indicator and may confuse `derez` 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 are two ways of padding 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 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 until the next `align` statement.

Array type

An array is declared as follows:

`[wide] array [` *array-name* | `['` *length* `']' ] {` *array-list* `};`

array-list, 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* `]` may 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 declaration

```
type 'STR#' {           /*define a string list resource*/
      integer = $$Countof(StringArray);
      array StringArray {
      pstring;
      };
};
```

the `$$Countof` function returns the number of array elements (in this case, the number of strings) from the `resource` data.

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 may 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;*]...

case-name is an identifier. *case-body* may 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 may specify a standard Macintosh resource type, as listed in the Resource Manager chapters of *Inside Macintosh,* Volumes I, IV, and V, or you may 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 may be on more than one line (like the `integer` statement in the example). Each statement represents a field in the resource data. Comments may appear anywhere that white space may 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 is true for bitstrings (and their derivatives, `byte`, `integer`, and `longint`). For example, in the statement

```
integer     documentProc, dBoxProc, plainDBox,
            altDBoxProc, noGrowDocProc,
            zoomProc=8, rDocProc=16;
```

the symbolic names `documentProc`, `dBoxProc`, `plainDBox`, `altDBoxProc`, and `noGrowDocProc` are automatically assigned the numeric values 0, 1, 2, 3, and 4.

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;
```

`delete`—delete a resource

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

Syntax

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

Description

Delete the resource of type *resource-type* from the output file with the specified *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
```

`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

Change the resource of type *resource-type1* in the output file with the specified *resource-name1, ID,* or range of ID numbers to a resource of type *resource-type-2* 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 ∂ 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 ∂ character is used to escape the character that follows from performing its usual action. In this case, the following character is a newline, and the line-termination function is escaped.

- *Note:* The `change` statement is only valid 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.

`resource`—specify resource data

`resource` statements specify actual resources, based on previous `type` declarations.

Syntax

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

Description

Specify 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* specifies the actual data; the *data-statement* appropriate to each resource type is defined in the next section.

The `resource` definition causes an actual resource to be generated. A `resource` statement can appear anywhere in the resource description file, or even 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.

Base type	Instance types
`string`	`string, cstring, pstring, wstring char`
`bitstring`	`boolean, byte, integer, longint, bitstring`
`rect`	`rect`
`point`	`point`

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 the section "Switch Type" earlier in this appendix. The *switch-name* example is `CheckBox`.

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

The `boolean` and `pstring` values defined in the *case-body* for 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*.

Array data

Array data statements have the following format:

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

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

For example, the following data might be given for the `'STR#'` type declaration example given earlier in the section "Array Type":

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

Sample resource definition

This section describes a sample resource description file for a window. (See Chapter 9, "The Window Manager" in *Inside Macintosh,* Volume I, for information about resources for windows.)

Once again, here is the `type` declaration given earlier in this appendix under "Sample `type` Statement":

```
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*/
};
```

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

```
resource 'WIND' (128,"My window",appheap,preload){      /*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`, it should not contain more than 255 characters. The resource name may 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 values

```
{40,80,120,300}
```

correspond to the coordinates *top, left, bottom,* and *right.*

Symbolic names

Symbolic names may 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 to 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 previously.

The sample window declaration assigned the values `true` (1) and `false` (0) to two different byte variables. For clarity, the window's resource data used the symbolic names

```
visible,
goAway,
```

instead of their equivalents

```
TRUE,
TRUE,
```

or

```
1,
1,
```

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 may appear on any statement.

In expressions, only the identifier portion of the label (that is, everything up to, but excluding, the colon) may be used. See "Declaring Labels Within Arrays" later in this appendix.

The value of a label is always the offset—in bits—between the beginning of the resource and the position where the label occurs when mapped to the resource data. In this label definition example

```
type 'strb' {
    cstring;
endOfString:
    integer = endOfString;
};
```

the label is defined as a `cstring` followed by an integer containing the bit count of the particular label.

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 the *numberOfBits* (maximum of 32) bitstring found *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, the resource type `'STR '` could be redefined without using a `pstring`. 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 may 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 above example, the label `foo` would take on these values:

```
foo[1,1] = 32      $$Word(foo[1,1]) = 1
foo[1,2] = 48      $$Word(foo[1,2]) = 2
foo[1,3] = 64      $$Word(foo[1,3]) = 3
foo[2,1] = 96      $$Word(foo[2,1]) = 4
foo[2,2] = 112     $$Word(foo[2,2]) = 5
```

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

`derez` is basically a one-pass decompiler. In order for `derez` to decompile a given type, any expression within that type must not 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.

`rez` 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*`)`, `$$Word(`*label*`)`, and `$$Long(`*label*`)` must occur lexically before the expression; otherwise, an error is generated.

Using labels: 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 version 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 would have to be combined into a single hex string (everything below the `PixelData` label). Using labels, the complete `'ppat'` definition can be expressed 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 prior 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           */
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 [$$BitField(pMapRowBytes, 0, 13) *
        ($$BitField(Bounds, 32, 16)            /* bottom                    */
               - $$BitField(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. This section describes the preprocessor directives.

The syntax of the preprocessor is similar to 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 pound 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) may consist of letters (`A-Z`, `a-z`), digits (`0-9`), or the underscore character (_). Identifiers may not start with a digit, are not case sensitive, and may be 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 \
thro\' 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:

Variable	Value
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*

Include 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 chapter, 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* may also include the following expression:

  `defined` *identifier* or `defined '('` *identifier* `')'`

The following directives may also be used in place of `#if`:

`#ifdef` *macro*
`#ifndef` *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 the `printf` statement in the C language with one exception: there can be no more than 20 arguments (this is the same restriction that 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 above file generates the following text:

```
The day is Tuesday, day #3
```

Resource description syntax

This section describes the details of the resource description syntax.

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

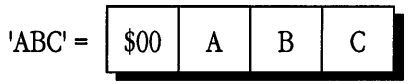
Numeric type	Form	Meaning
`decimal`	*nnn...*	Signed decimal constant between 4294967295 and –2147483648.
`hex`	`0X`*hhh...*	Signed hexadecimal constant between 0X7FFFFFFF and 0X80000000.
	`$`*hhh...*	Alternate form for hexadecimal constants.
`octal`	`0`*ooo...*	Signed octal constant between 017777777777 and 020000000000.
`binary`	`0B`*bbb...*	Signed binary constant between 0B11111111111111111111111111111111 and 0B10000000000000000000000000000000.
`literal`	`'`*aaaa*`'`	A literal with one to four characters. Characters are printable ASCII characters or escape characters (defined below). 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 \' and \\ (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 may 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 left to right when the priority is the same. Variables are defined following 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*	Arithmetic (two's complement) negation of *expr*.
	~*expr*	Bitwise (one's complement) negation of *expr*.
	!*expr*	Logical negation of *expr*.
3.	*expr1* * *expr2*	Multiplication.
	expr1 / *expr2*	Division.
	expr1 % *expr2*	Remainder from dividing *expr1* by *expr2*.
4.	*expr1* + *expr2*	Addition.
	expr1 - *expr2*	Subtraction.

[continued]

■ **Table E-2** Resource-description expression operators [continued]

Operator		Meaning
5.	*expr1* << *expr2*	Shift left—shift *expr1* left by *expr2* bits.
	expr1 >> *expr2*	Shift right—shift *expr1* right by *expr2* bits.
6.	*expr1* > *expr2*	Greater than.
	expr1 >= *expr2*	Greater than or equal to.
	expr1 < *expr2*	Less than.
	expr1 <= *expr2*	Less than or equal to.
7.	*expr1* == *expr2*	Equal to.
	expr1 != *expr2*	Not equal to.
8.	*expr1* & *expr2*	Bitwise AND.
9.	*expr1* ^ *expr2*	Bitwise XOR.
10.	*expr1* | *expr2*	Bitwise OR.
11.	*expr1* && *expr2*	Logical AND.
12.	*expr1* || *expr2*	Logical OR.

Note: The logical operators !, >, >=, <, <=, ==, !=, &&, || 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`. ("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. (See "Print Directive" earlier in this appendix.)

`$$Name` Name of resource from the current resource. The current resource is the resource being generated by a `resource` statement, being included with an `include` statement, being deleted by a `delete` statement, or being changed by a `change` statement. In addition to the `$$Name` string variable, three numeric variables, `$$Type`, `$$ID`, and `$$Attributes`, refer to the current resource. They are described in the next section "Numeric Variables."

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 of 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 `IUTimeString`. ("7:50:54 AM")

`$$Version` Version number of the resource compiler. (`"V3.0"`)

Numeric values

The following variables and functions have numeric values:

`$$Attributes` Attributes of resource from the current resource. See the `$$Name` string variable.

`$$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.

`$$ID` ID of resource from the current resource. See the `$$Name` string variable.

`$$Long(`*`label`*`)`
Return value of the longword found at *label.*

`$$Minute` Current minute, range 0–59.

`$$Month` Current month, range 1–12.

`$$PackedSize(`*`Start, RowBytes, RowCount`*`)`
Reference to the current resource (see the `$$Name` string variable). Provided with an offset, *Start,* into the current resource and two integers, *RowBytes* and *RowCount,* this function calls the Toolbox utility routine `UnpackBits` for *RowCount* times, and returns the unpacked size of the data found at *start.* Use `$$PackedSize( )` only for decompiling resource files. An example using this function is found in `pict.r`.

$$ResourceSize

Current size of resource in bytes. When decompiling, `$$ResourceSize` is the actual size of the resource being decompiled. When compiling, `$$ResourceSize` returns the number of bytes that have been compiled so far for the current resource. (See the `'KCHR'` resource in `systypes.r` for an example.)

`$$Second` Current second, range 0–59.

`$$Type` Type of resource from the current resource. See the `$$Name` string variable.

`$$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...*`"`
The string can contain any printable character except ' " ' and '\'. 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. (In this case, newlines and comments are considered white space.)

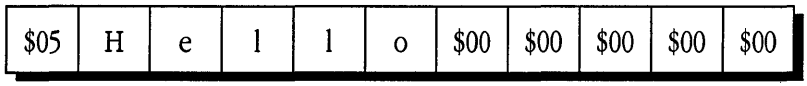
Figure E-4 shows a Pascal string declared as

```
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 in a string, 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
`\t`	Tab	$09	None
`\b`	Backspace	$08	None
`\r`	Return	$0D	None
`\n`	Newline	$0A	None
`\f`	Form feed	$0C	None
`\v`	Vertical tab	$0B	None
`\?`	Rubout	$7F	None
`\\`	Backslash	$5C	\
`\'`	Single quote	$3A	'
`\"`	Double quote	$22	"

Note: In the Macintosh OS, `\n` is treated as a RETURN ($0D). In A/UX, `\n` is treated as a LINE FEED ($0A). `rez` 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	`\0B`*bbbbbbbb*	8	`\0B01000001`
8	`\`*ooo*	3	`\101`
10	`\0D`*ddd*	3	`\0D065`
16	`\0x`*hh*	2	`\0X41`
16	`\$`*hh*	2	`\$41`

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 `derez` 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

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 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, "Inside A/UX Macintosh."

Interface library files

Libraries in this section appear in alphabetical order by library name, not in *Inside Macintosh* order.

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) 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 a `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. For a list of all libraries described in *Inside Macintosh* and their status in the A/UX Toolbox, see Chapter 5, "Inside A/UX Macintosh."

The two following sections, "Structures and Calls by Library" and "Calls in Alphabetic Order," provide information selected from the headers 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`
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 facility	`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	

[continued]

■ **Table F-1** Interface library files [continued]

Library	Header file
Palette Manager	`palettes.h`
Printing Manager	`printing.h`
Print traps	`printtraps.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`
String conversion, Pascal-to-and-from 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 may be displayed, searched, or printed.

The two sections that follow are "Structures and Calls by Library," which lists the structures and calls in the header files listed above, and "Calls in Alphabetic Order," which lists all calls in alphabetical order by name.

Structures and calls by library

The names of the structures and the calls that are available in the header files listed in Table F-1 are given here, 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 alphabetic order. Calls are in alphabetic 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," that reference is given with the information here in Appendix F.

32-Bit QuickDraw with Color QuickDraw

The following structures and calls are available in `quickdraw.h`.

Structure name
`BitMap`
`CCrsr`
`CGrafPort`
`CIcon`
`ColorSpec`
`ColorTable`
`CProcRec`
`CQDProcs`
`Cursor`
`FontInfo`
`GammaTbl`
`GDevice`

Structure name
GrafPort
GrafVars
ITab
MatchRec
PenState
Picture
PixMap
PixPat
Polygon
QDProcs
qdvar
Region
ReqListRec
RGBColor
SProcRec

Call	Return type
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

Call	Return type
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
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

Call	Return type
EqualRgn();	Boolean
EraseArc();	void
EraseOval();	void
ErasePoly();	void
EraseRect();	void
EraseRgn();	void
EraseRoundRect();	void
FillArc();	void
FillCArc();	void
FillCOval();	void
FillCPoly();	void
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

Call	Return type
GetCTable();	CTabHandle
GetCTSeed();	long
GetDeviceList();	GDHandle
GetFontInfo();	void
GetForeColor();	void
GetGDevice();	GDHandle
GetMainDevice();	GDHandle
GetMaskTable();	Ptr
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

Call	Return type
`InvertOval();`	`void`
`InvertPoly();`	`void`
`InvertRect();`	`void`
`InvertRgn();`	`void`
`InvertRoundRect();`	`void`
`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`

Call	Return type
OpenPoly();	PolyHandle
OpenPort();	void
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
RectRgn();	void

Call	Return type
`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`
`SetRectRgn();`	`void`
`SetStdCProcs();`	`void`
`SetStdProcs();`	`void`
`ShowCursor();`	`void`

Call	Return type
`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`
`TextFace();`	`void`
`TextFont();`	`void`
`TextMode();`	`void`
`TextSize();`	`void`
`TextWidth();`	`short`
`UnionRect();`	`void`
`UnionRgn();`	`void`

Call	Return type
`XorRgn();`	`void`

Color Picker

The following structures and calls are available in `picker.h`.

Structure name
`CMYColor`
`HSLColor`
`HSVColor`

Call	Return type
`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	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`
`GetCTitle();`	`void`
`getctitle();`	`void`
`GetCtlAction();`	`ProcPtr`
`GetCtlMax();`	`short`
`GetCtlMin();`	`short`
`GetCtlValue();`	`short`
`GetCVariant();`	`short`

Call	Return type
`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`

Definitions for AUXDispatch

The following structures and calls are available in aux.h. "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
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.

Call	Return type
CloseDeskAcc();	void
OpenDeskAcc();	short
opendeskacc();	short
SystemClick();	void
SystemEdit();	Boolean
SystemEvent();	Boolean

Call	Return type
SystemMenu();	void
SystemTask();	void

Device Manager

The following structures and calls are available in devices.h.

Structure name
AuxDCE
DCtlEntry

Call	Return type
CloseDriver();	OSErr
Control();	OSErr
GetDCtlEntry();	DCtlHandle
KillIO();	OSErr
OpenDriver();	OSErr
opendriver();	OSErr
PBControl();	OSErr
PBKillIO();	OSErr
PBStatus();	OSErr
SetChooserAlert();	Boolean
Status();	OSErr

Dialog Manager

The following structures and calls are available in `dialogs.h`.

Structure name
`AlertTemplate`
`DialogRecord`
`DialogTemplate`

Call	Return type
`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`
`FreeDialog();`	`void`
`GetAlrtStage();`	`short`
`GetDItem();`	`void`
`GetIText();`	`void`
`getitext();`	`void`
`GetNewDialog();`	`DialogPtr`
`HideDItem();`	`void`

Call	Return type
`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
`HFSDefaults`

Call	Return type
`DIBadMount();`	`short`
`dibadmount();`	`OSErr`
`DIFormat();`	`OSErr`
`DILoad();`	`void`
`DIUnload();`	`void`
`DIVerify();`	`OSErr`
`DIZero();`	`OSErr`
`dizero();`	`OSErr`

Disk Initialization Package

The following structures and calls are available in `diskinit.h`.

Structure name
`DrvSts`
`DrvSts2`

Call	Return type
`DiskEject();`	`OSErr`
`DriveStatus();`	`OSErr`
`SetTagBuffer();`	`OSErr`

Event Manager, Operating System

The following structures and calls are available in `osevents.h`.

Structure name
`EvQEl`

Call	Return Type
`FlushEvents();`	`void`
`GetEvQHdr();`	`QHdrPtr`
`GetOSEvent();`	`Boolean`
`OSEventAvail();`	`Boolean`
`PostEvent();`	`OSErr`
`PPostEvent();`	`OSErr`
`SetEventMask();`	`void`

Event Manager, Toolbox

The following structures and calls are available in `events.h`.

Structure name
`EventRecord`

Call	Return Type
`Button();`	`Boolean`
`EventAvail();`	`Boolean`
`GetCaretTime();`	`unsigned long`
`GetDblTime();`	`unsigned long`
`GetKeys();`	`void`
`GetMouse();`	`void`
`GetNextEvent();`	`Boolean`
`StillDown();`	`Boolean`

Call	Return type
`TickCount();`	`unsigned long`
`WaitMouseUp();`	`Boolean`
`WaitNextEvent();`	`Boolean`

File Manager

The following structures and calls are available in `files.h`.

Structure name
`AccessParam`
`CMovePBRec`
`CntrlParam`
`CopyParam`
`DInfo`
`DirInfo`
`DrvQEl`
`DXInfo`
`FCBPBRec`
`FileParam`
`FInfo`
`FXInfo`
`HFileInfo`
`HFileParam`
`HIOParam`
`HParamBlockRec`
`HVolumeParam`
`IOParam`
`MultiDevParam`
`NumVersion`
`ObjParam`

Structure name
SlotDevParam
VCB
VersRec
VolumeParam
WDParam
WDPBRec

Call	Return Type
AddDrive();	void
Allocate();	OSErr
AllocContig();	OSErr
CatMove();	OSErr
CloseWD();	OSErr
Create();	OSErr
create();	OSErr
DirCreate();	OSErr
Eject();	OSErr
eject();	OSErr
FInitQueue();	void
FlushVol();	OSErr
flushvol();	OSErr
FSClose();	OSErr
FSDelete();	OSErr
fsdelete();	OSErr
FSOpen();	OSErr
fsopen();	OSErr
FSRead();	OSErr
fsrename();	OSErr
FSWrite();	OSErr
GetDrvQHdr();	QHdrPtr
GetEOF();	OSErr

Call	Return Type
GetFInfo();	OSErr
getfinfo();	OSErr
GetFPos();	OSErr
GetFSQHdr();	QHdrPtr
GetVCBQHdr();	QHdrPtr
GetVInfo();	OSErr
getvinfo();	OSErr
GetVol();	OSErr
getvol();	OSErr
GetVRefNum();	OSErr
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

Call	Return Type
PBDelete();	OSErr
PBDirCreate();	OSErr
PBEject();	OSErr
PBFlushFile();	OSErr
PBFlushVol();	OSErr
PBGetCatInfo();	OSErr
PBGetEOF();	OSErr
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

Call	Return Type
PBHSetDirAccess();	OSErr
PBHSetFInfo();	OSErr
PBHSetFLock();	OSErr
PBHSetVol();	OSErr
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

Call	Return Type
`setflock();`	`OSErr`
`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`
`FamRec`
`FMetricRec`
`FMInput`
`FMOutput`
`FontAssoc`
`FontRec`
`KernEntry`
`KernPair`
`KernTable`
`NameTable`
`StyleTable`
`WidEntry`
`WidTable`
`WidthTable`

Call	Return Type
FMSwapFont();	FMOutPtr
FontMetrics();	void
GetFNum();	void
getfnum();	void
GetFontName();	void
getfontname();	void
InitFonts();	void
RealFont();	Boolean
SetFontLock();	void
SetFractEnable();	void
SetFScaleDisable();	void

Gestalt facility

No structures or calls are available in `gestalt.h`.

List Manager Package

The following structures and calls are available in `lists.h`.

Structure name
ListRec

Call	Return Type
LActivate();	void
LAddColumn();	short
LAddRow();	short
LAddToCell();	void
LAutoScroll();	void

Call	Return Type
`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`
`LLastClick();`	`Cell`
`LNew();`	`ListHandle`
`lnew();`	`ListHandle`
`LNextCell();`	`Boolean`
`LRect();`	`void`
`LScroll();`	`void`
`LSearch();`	`Boolean`
`LSetCell();`	`void`
`LSetSelect();`	`void`
`LSize();`	`void`
`LUpdate();`	`void`

List of Macintosh traps

No structures or calls are available in `traps.h`, which provides a list of definitions for A-line trap words.

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 following structures and calls are available in `memory.h`.

Structure name
`Zone`

Call	Return Type
`GetApplLimit();`	`Ptr`
`GetZone();`	`THz`
`SystemZone();`	`THz`
`ApplicZone();`	`THz`
`NewHandle();`	`Handle`
`HandleZone();`	`THz`
`RecoverHandle();`	`Handle`
`NewPtr();`	`Ptr`
`PtrZone();`	`THz`
`GZSaveHnd();`	`Handle`
`TopMem();`	`Ptr`
`MaxBlock();`	`long`
`StackSpace();`	`long`
`NewEmptyHandle();`	`Handle`
`HLock();`	`void`
`HUnlock();`	`void`
`HPurge();`	`void`
`HNoPurge();`	`void`
`StripAddress();`	`Ptr`

Call	Return Type
MFMaxMem();	Size
MFFreeMem();	long
MFTempNewHandle();	Handle
MFTempHLock();	void
MFTempHUnlock();	void
MFTempDisposHandle();	void
MFTopMem();	Ptr
InitApplZone();	void
InitZone();	void
SetZone();	void
CompactMem();	Size
PurgeMem();	void
FreeMem();	long
ResrvMem();	void
MaxMem();	Size
SetGrowZone();	void
SetApplLimit();	void
MoveHHi();	void
DisposPtr();	void
GetPtrSize();	Size
SetPtrSize();	void
DisposHandle();	void
GetHandleSize();	Size
SetHandleSize();	void
EmptyHandle();	void
ReallocHandle();	void
HSetRBit();	void
HClrRBit();	void
MoreMasters();	void
BlockMove();	void
MemError();	OSErr

Call	Return Type
PurgeSpace();	void
HGetState();	short
HSetState();	void
SetApplBase();	void
MaxApplZone();	void

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

Call	Return Type
`FlashMenuBar();`	`void`
`GetItem();`	`void`
`getitem();`	`void`
`GetItemCmd();`	`void`
`GetItemIcon();`	`void`
`GetItemMark();`	`void`
`GetItemStyle();`	`void`
`GetMCEntry();`	`MCEntryPtr`
`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`

Call	Return Type
SetItemMark();	void
SetItemStyle();	void
SetMCEntries();	void
SetMCInfo();	void
SetMenuBar();	void
SetMenuFlash();	void

Notification Manager

The following structures 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`
`IUDateString();`	`void`
`iudatestring();`	`void`
`IUEqualString();`	`short`
`iuequalstring();`	`short`
`IUGetIntl();`	`Handle`
`IUMagIDString();`	`short`
`IUMagString();`	`short`
`IUMetric();`	`Boolean`
`IUSetIntl();`	`void`
`IUTimePString();`	`void`
`iutimepstring();`	`void`
`IUTimeString();`	`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`

Call	Return Type
`stringtonum();`	`void`

Palette Manager

The following structures and calls are available in `palettes.h`.

Structure name
`ColorInfo`
`Palette`

Call	Return Type
`ActivatePalette();`	`void`
`AnimateEntry();`	`void`
`AnimatePalette();`	`void`
`CopyPalette();`	`void`
`CTab2Palette();`	`void`
`DisposePalette();`	`void`
`GetEntryColor();`	`void`
`GetEntryUsage();`	`void`
`GetNewPalette();`	`PaletteHandle`
`GetPalette();`	`PaletteHandle`
`InitPalettes();`	`void`
`NewPalette();`	`PaletteHandle`
`NSetPalette();`	`void`
`Palette2CTab();`	`void`
`PmBackColor();`	`void`
`PmForeColor();`	`void`
`SetEntryColor();`	`void`
`SetEntryUsage();`	`void`
`SetPalette();`	`void`

Printing Manager

The following structures and calls are available in `printing.h`.

Structure name
`TDftBitsBlk`
`TGetRotnBlk`
`TGetRslBlk`
`TGnlData`
`TPfPgDir`
`TPrDlg`
`TPrInfo`
`TPrint`
`TPrJob`
`TPrPort`
`TPrStatus`
`TPrStl`
`TPrXInfo`
`TRslRec`
`TRslRg`
`TSetRslBlk`

Call	Return Type
`PrClose();`	`void`
`PrCloseDoc();`	`void`
`PrClosePage();`	`void`
`PrCtlCall();`	`void`
`PrDlgMain();`	`Boolean`
`PrDrvrClose();`	`void`
`PrDrvrDCE();`	`Handle`
`PrDrvrOpen();`	`void`
`PrDrvrVers();`	`short`
`PrError();`	`short`

Call	Return Type
`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`

Print traps

The following structures and calls are available in `printtraps.h`.

Structure name
`TDftBitsBlk`
`TGetRotnBlk`
`TGetRslBlk`
`TGnlData`
`TPfPgDir`
`TPrDlg`
`TPrInfo`
`TPrint`
`TPrJob`

Structure name
TPrPort
TPrStatus
TPrStl
TPrXInfo
TRslRec
TRslRg
TSetRslBlk

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
PrOpenPage();	void
PrPicFile();	void
PrPurge();	void
PrSetError();	void

Call	Return Type
`PrStlDialog();`	`Boolean`
`PrStlInit();`	`TPPrDlg`
`PrValidate();`	`Boolean`

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.

Call	Return Type
`AddResource();`	`void`
`addresource();`	`void`
`ChangedResource();`	`void`
`CloseResFile();`	`void`
`Count1Resources();`	`short`
`Count1Types();`	`short`
`CountResources();`	`short`
`CountTypes();`	`short`
`CreateResFile();`	`void`
`createresfile();`	`void`
`CurResFile();`	`short`
`DetachResource();`	`void`
`Get1IndResource();`	`Handle`
`Get1IndType();`	`void`
`Get1NamedResource();`	`Handle`
`get1namedresource();`	`Handle`
`Get1Resource();`	`Handle`
`GetIndResource();`	`Handle`

Call	Return Type
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
RmveResource();	void
RsrcMapEntry();	long
RsrcZoneInit();	void
SetResAttrs();	void
SetResFileAttrs();	void
SetResInfo();	void
setresinfo();	void
SetResLoad();	void
SetResPurge();	void
SizeResource();	long

Call	Return Type
`Unique1ID();`	`short`
`UniqueID();`	`short`
`UpdateResFile();`	`void`
`UseResFile();`	`void`
`WriteResource();`	`void`

The following structures and calls are available in `asd.h`.

Structure name
`FileInfo`
`FInfo`

Call	Return Type
`CloseASD();`	`int`
`OpenASD();`	`FileHandle`
`ReadASD();`	`long`
`SeekASD();`	`long`
`WriteASD();`	`long`

The following calls are available in `aux_rsrc.h`, which has no structures.

Call	Return Type
`int`	`mrclose();`
`int`	`mrinfo();`
`ResHandle`	`mropen();`
`Resource`	`mrget();`
`Resource`	`mrgetnamed();`
`short`	`mrattr();`
`void`	`mrrel();`

Scrap Manager

The following structures and calls are available in `scrap.h`.

Structure name
`ScrapStuff`

Call	Return Type
`GetScrap();`	`long`
`InfoScrap();`	`PScrapStuff`
`LoadScrap();`	`long`
`PutScrap();`	`long`
`UnloadScrap();`	`long`
`ZeroScrap();`	`long`

Script Manager

The following structures and calls are available in `script.h`.

Structure name
`BreakTable`
`ItlcRecord`
`ItlbRecord`
`Location`
`Token`
`TokenBlock`
`UntokenTable`
`DateCacheRecord`
`TogglePB`
`WideCharArr`
`FormatString`
`Itl4Rec`

Structure name
NumberParts
StartLength
FindBlockStatus

Call	Return Type
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
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

Call	Return Type
`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`
`Tokenize();`	`TokenResult`
`Transliterate();`	`OSErr`
`UprText();`	`void`
`ValidDate();`	`short`
`VisibleLength();`	`long`
`WriteLocation();`	`void`

Segment Loader

The following structures and calls are available in `segload.h`.

Structure name
`AppFile`

Call	Return Type
`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

No structures and calls are available in `shutdown.h`.

Slot Manager

The following structures and calls are available in `slots.h`.

Structure name
`SlotIntQElement`
`SpBlock`
`SInfoRecord`
`SDMRecord`
`FHeaderRec`
`SEBlock`

Call	Return Type
`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`

Call	Return Type
SGetCString();	OSErr
SGetDriver();	OSErr
SGetsRsrc();	OSErr
SGetsRsrcInfo();	OSErr
SGetTypesRsrc();	OSErr
SInitPRAMRecs();	OSErr
SInitSRsrcTable();	OSErr
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
SReadPRAMRec();	OSErr
SReadStruct();	OSErr
SReadWord();	OSErr
SRsrcInfo();	OSErr
SSearchSRT();	OSErr
SSetsRsrcState();	OSErr
SUpdateSRT();	OSErr
SVersion();	OSErr

Sound Manager

The following structures and calls are available in `sm.h`.

Structure name
`ModifierStub`
`SndChannel`
`SndCommand`
`SndListResource`
`SoundHeader`

Call	Return Type
`aSndAddModifier();`	`short`
`aSndControl();`	`short`
`aSndDisposeChannel();`	`short`
`aSndDoCommand();`	`short`
`aSndDoImmediate();`	`short`
`aSndNewChannel();`	`short`
`aSndPlay();`	`short`

String conversion, Pascal-to-and-from C

The following calls are available in `strings.h`, which has no structures.

Call	Return Type
`*p2cstr();`	`char`
`c2pstr();`	`StringPtr`

System Error Handler

The following calls are available in `errors.h`, which has no structures.

Call	Return Type
`SysError();`	`void`

TextEdit

The following structures and calls are available in `textedit.h`.

Structure name
`LHElement`
`NullStRec`
`ScrpSTElement`
`STElement`
`StScrpRec`
`StyleRun`
`TERec`
`TEStyleRec`
`TextStyle`

Call	Return Type
`GetStylHandle();`	`TEStyleHandle`
`GetStylScrap();`	`StScrpHandle`
`SetClikLoop();`	`void`
`SetStylHandle();`	`void`
`SetStylScrap();`	`void`
`SetWordBreak();`	`void`
`TEActivate();`	`void`
`TEAutoView();`	`void`
`TECalText();`	`void`

Call	Return Type
`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`
`TEReplaceStyle();`	`void`
`TEScrapHandle();`	`Handle`
`TEScroll();`	`void`
`TESelView();`	`void`
`TESetJust();`	`void`
`TESetScrapLen();`	`void`
`TESetSelect();`	`void`

Call	Return Type
TESetStyle();	void
TESetText();	void
TEStylInsert();	void
TEStylNew();	TEHandle
TEStylPaste();	void
TEToScrap();	OSErr
TEUpdate();	void
TextBox();	void

Time Manager

The following structures and calls are available in `timer.h`.

Structure name
TMTask

Call	Return Type
InsTime();	void
PrimeTime();	void
RmvTime();	void

Utilities, Operating System

The following structures and calls are available in `osutils.h`.

Structure name	Structure name
DateTimeRec	SysEnvRec
QElem	SysParmType
QHdr	

Call	Return Type
`Date2Secs();`	`void`
`Delay();`	`void`
`Dequeue();`	`OSErr`
`DTInstall();`	`OSErr`
`Enqueue();`	`void`
`Environs();`	`void`
`EqualString();`	`Boolean`
`equalstring();`	`Boolean`
`GetDateTime();`	`void`
`GetMMUMode();`	`char`
`GetTime();`	`void`
`GetTrapAddress();`	`long`
`HandAndHand();`	`OSErr`
`HandToHand();`	`OSErr`
`InitUtil();`	`OSErr`
`KeyTrans();`	`long`
`NGetTrapAddress();`	`long`
`NSetTrapAddress();`	`void`
`PtrAndHand();`	`OSErr`
`PtrToHand();`	`OSErr`
`PtrToXHand();`	`OSErr`
`ReadDateTime();`	`OSErr`
`RelString();`	`short`
`relstring();`	`short`
`Restart();`	`void`
`Secs2Date();`	`void`
`SetA5();`	`long`
`SetCurrentA5();`	`long`
`SetDateTime();`	`OSErr`
`SetTime();`	`void`
`SetTrapAddress();`	`void`

Call	Return Type
SwapMMUMode();	void
SysBeep();	void
SysEnvirons();	OSErr
UprString();	void
uprstring();	void
WriteParam();	OSErr

Utilities, Toolbox

The following structures and calls are available in `toolutils.h`.

Structure name
Int64Bit

Call	Return Type
AngleFromSlope();	short
BitAnd();	long
BitClr();	void
BitNot();	long
BitOr();	long
BitSet();	void
BitShift();	long
BitTst();	Boolean
BitXor();	long
DeltaPoint();	long
deltapoint();	long
FixMul();	Fixed
FixRatio();	Fixed
FixRound();	short
GetCursor();	CursHandle

Call	Return Type
`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`
`setstring();`	`void`
`ShieldCursor();`	`void`
`shieldcursor();`	`void`
`SlopeFromAngle();`	`Fixed`
`UnpackBits();`	`void`

Vertical Retrace Manager

The following structures and calls are available in `retrace.h`.

Structure name
`VBLTask`

Call	Return Type
`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`
`VDEntryRecord`
`VDGrayRecord`
`VDPageInfo`
`VDSetEntryRecord`
`VDSettings`
`VDSizeInfo`
`VPBlock`

Window Manager

The following structures and calls are available in `windows.h`.

Structure name
`AuxWinRec`
`CWindowRecord`
`WinCTab`
`WindowRecord`
`WStateData`

Call	Return Type
`BeginUpdate();`	`void`
`BringToFront();`	`void`
`CalcVis();`	`void`
`CalcVisBehind();`	`void`
`CheckUpdate();`	`Boolean`
`ClipAbove();`	`void`
`CloseWindow();`	`void`
`DisposeWindow();`	`void`
`DragGrayRgn();`	`long`
`draggrayrgn();`	`long`
`DragWindow();`	`void`
`dragwindow();`	`void`
`DrawGrowIcon();`	`void`
`DrawNew();`	`void`
`EndUpdate();`	`void`
`FindWindow();`	`short`
`findwindow();`	`short`
`FrontWindow();`	`WindowPtr`
`GetAuxWin();`	`Boolean`
`GetCWMgrPort();`	`void`
`GetGrayRgn();`	`RgnHandle`

Call	Return Type
GetNewCWindow();	WindowPtr
GetNewWindow();	WindowPtr
GetWindowPic();	PicHandle
GetWMgrPort();	void
GetWRefCon();	long
GetWTitle();	void
getwtitle();	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
SetWinColor();	void
SetWindowPic();	void
SetWRefCon();	void

Call	Return Type
`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 alphabetic order

All calls in the prior section are listed here in alphabetic order by name, followed by the return type and the name of the header file containing that 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 "Differences in Language Conventions" in Chapter 4.

Call	Return type	Header file
`*p2cstr();`	`char`	`strings.h`
`ActivatePalette();`	`void`	`palettes.h`
`AddComp();`	`void`	`quickdraw.h`
`AddDrive();`	`void`	`files.h`
`AddPt();`	`void`	`quickdraw.h`
`addpt();`	`void`	`quickdraw.h`
`AddResMenu();`	`void`	`menus.h`
`AddResource();`	`void`	`resources.h`
`addresource();`	`void`	`resources.h`
`AddSearch();`	`void`	`quickdraw.h`
`Alert();`	`short`	`dialogs.h`
`Allocate();`	`OSErr`	`files.h`
`AllocContig();`	`OSErr`	`files.h`
`AllocCursor();`	`void`	`quickdraw.h`
`AngleFromSlope();`	`short`	`toolutils.h`
`AnimateEntry();`	`void`	`palettes.h`
`AnimatePalette();`	`void`	`palettes.h`
`AppendMenu();`	`void`	`menus.h`
`appendmenu();`	`void`	`menus.h`

Call	Return type	Header file
ApplicZone();	THz	memory.h
aSndAddModifier();	short	sm.h
aSndControl();	short	sm.h
aSndDisposeChannel();	short	sm.h
aSndDoCommand();	short	sm.h
aSndDoImmediate();	short	sm.h
aSndNewChannel();	short	sm.h
aSndPlay();	short	sm.h
AttachVBL();	OSErr	retrace.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

Call	Return type	Header file
CatMove();	OSErr	files.h
CautionAlert();	short	dialogs.h
ChangedResource();	void	resources.h
Char2Pixel();	short	script.h
CharByte();	short	script.h
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

Call	Return type	Header file
CopyBits();	void	quickdraw.h
CopyMask();	void	quickdraw.h
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
Delay();	void	osutils.h
DelComp();	void	quickdraw.h
DeleteMenu();	void	menus.h
DelMCEntries();	void	menus.h
DelMenuItem();	void	menus.h
DelSearch();	void	quickdraw.h
DeltaPoint();	long	toolutils.h

Call	Return type	Header file
deltapoint();	long	toolutils.h
Dequeue();	OSErr	osutils.h
DetachResource();	void	resources.h
DialogSelect();	Boolean	dialogs.h
DIBadMount();	short	disks.h
dibadmount();	OSErr	disks.h
DiffRgn();	void	quickdraw.h
DIFormat();	OSErr	disks.h
DILoad();	void	disks.h
DirCreate();	OSErr	files.h
DisableItem();	void	menus.h
DiskEject();	OSErr	diskinit.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
DisposPtr();	void	memory.h
DIUnload();	void	disks.h
DIVerify();	OSErr	disks.h
DIZero();	OSErr	disks.h
dizero();	OSErr	disks.h

Call	Return type	Header file
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
Draw1Control();	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
DriveStatus();	OSErr	diskinit.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

Call	Return type	Header file
EndUpdate();	void	windows.h
Enqueue();	void	osutils.h
Environs();	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
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

Call	Return type	Header file
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
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

Call	Return type	Header file
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
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

Call	Return type	Header file
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
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

Call	Return type	Header file
GetForeColor();	void	quickdraw.h
GetFormatOrder();	void	script.h
GetFPos();	OSErr	files.h
GetFSQHdr();	QHdrPtr	files.h
GetGDevice();	GDHandle	quickdraw.h
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

Call	Return type	Header file
GetMMUMode();	char	osutils.h
GetMouse();	void	events.h
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
GetOSEvent();	Boolean	osevents.h
GetPalette();	PaletteHandle	palettes.h
GetPattern();	PatHandle	toolutils.h
GetPen();	void	quickdraw.h
GetPenState();	void	quickdraw.h
GetPicture();	PicHandle	toolutils.h
GetPixel();	Boolean	quickdraw.h
GetPixPat();	PixPatHandle	quickdraw.h
GetPort();	void	quickdraw.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
GetScrap();	long	scrap.h
GetScript();	long	script.h
GetString();	StringHandle	toolutils.h
GetStylHandle();	TEStyleHandle	textedit.h

Call	Return type	Header file
GetStylScrap();	StScrpHandle	textedit.h
GetSubTable();	void	quickdraw.h
GetSysFont();	short	script.h
GetSysJust();	short	script.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
getwtitle();	void	windows.h
GetWVariant();	short	windows.h
GetZone();	THz	memory.h
GlobalToLocal();	void	quickdraw.h
GrafDevice();	void	quickdraw.h
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

Call	Return type	Header file
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
HomeResFile();	short	resources.h
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

Call	Return type	Header file
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
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
int	mrclose();	aux_rsrc.h

Call	Return type	Header file
int	mrinfo();	aux_rsrc.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
iudatepstring();	void	packages.h
IUDateString();	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

Call	Return type	Header file
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
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

Call	Return type	Header file
lnew();	ListHandle	lists.h
LNextCell();	Boolean	lists.h
LoadResource();	void	resources.h
LoadScrap();	long	scrap.h
LocalToGlobal();	void	quickdraw.h
LongDate2Secs();	void	script.h
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

Call	Return type	Header file
MenuKey();	long	menus.h
MenuSelect();	long	menus.h
menuselect();	long	menus.h
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
Munger();	long	toolutils.h
NewCDialog();	DialogPtr	dialogs.h
newcdialog();	DialogPtr	dialogs.h
NewControl();	ControlHandle	controls.h
newcontrol();	ControlHandle	controls.h
NewCWindow();	WindowPtr	windows.h
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

Call	Return type	Header file
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
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

Call	Return type	Header file
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
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

Call	Return type	Header file
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
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

Call	Return type	Header file
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
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

Call	Return type	Header file
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
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

Call	Return type	Header file
PrError();	short	printing.h
PrError();	short	printtraps.h
PrGeneral();	void	printing.h
PrGeneral();	void	printtraps.h
PrimeTime();	void	timer.h
PrintDefault();	void	printing.h
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

Call	Return type	Header file
PrStlInit();	TPPrDlg	printtraps.h
PrValidate();	Boolean	printing.h
PrValidate();	Boolean	printtraps.h
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

Call	Return type	Header file
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
ResHandle	mropen();	aux_rsrc.h
Resource	mrget();	aux_rsrc.h
Resource	mrgetnamed();	aux_rsrc.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
SaveEntries();	void	quickdraw.h
SaveOld();	void	windows.h
SCalcSPointer();	OSErr	slots.h
SCalcStep();	OSErr	slots.h
ScalePt();	void	quickdraw.h
SCardChanged();	OSErr	slots.h

Call	Return type	Header file
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
SetClip();	void	quickdraw.h
SetCPixel();	void	quickdraw.h
SetCRefCon();	void	controls.h
SetCTitle();	void	controls.h

Call	Return type	Header file
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
SetFontLock();	void	fonts.h
SetFPos();	OSErr	files.h
SetFractEnable();	void	fonts.h
SetFScaleDisable();	void	fonts.h
SetGDevice();	void	quickdraw.h
SetGrowZone();	void	memory.h
SetHandleSize();	void	memory.h

Call	Return type	Header file
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
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

Call	Return type	Header file
SetString();	void	toolutils.h
setstring();	void	toolutils.h
SetStylHandle();	void	textedit.h
SetStylScrap();	void	textedit.h
SetSysJust();	void	script.h
SetTagBuffer();	OSErr	diskinit.h
SetTime();	void	osutils.h
SetTrapAddress();	void	osutils.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
SetWTitle();	void	windows.h
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

Call	Return type	Header file
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
short	mrattr();	aux_rsrc.h
ShowControl();	void	controls.h
ShowCursor();	void	quickdraw.h
ShowDItem();	void	dialogs.h
ShowHide();	void	windows.h
ShowPen();	void	quickdraw.h
ShowWindow();	void	windows.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
SNextSRsrc();	OSErr	slots.h
SNextTypeSRsrc();	OSErr	slots.h
SOffsetData();	OSErr	slots.h
SpaceExtra();	void	quickdraw.h

Call	Return type	Header file
SPrimaryInit();	OSErr	slots.h
SPtrToSlot();	OSErr	slots.h
SPutPRAMRec();	OSErr	slots.h
SReadByte();	OSErr	slots.h
SReadDrvrName();	OSErr	slots.h
SReadFHeader();	OSErr	slots.h
SReadInfo();	OSErr	slots.h
SReadLong();	OSErr	slots.h
SReadPBSize();	OSErr	slots.h
SReadPRAMRec();	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
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

Call	Return type	Header file
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
subpt();	void	quickdraw.h
SUpdateSRT();	OSErr	slots.h
SVersion();	OSErr	slots.h
SwapMMUMode();	void	osutils.h
SysBeep();	void	osutils.h
SysEnvirons();	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

Call	Return type	Header file
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
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

Call	Return type	Header file
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
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
TrackGoAway();	Boolean	windows.h
trackgoaway();	Boolean	windows.h
Transliterate();	OSErr	script.h
UnionRect();	void	quickdraw.h
UnionRgn();	void	quickdraw.h
Unique1ID();	short	resources.h
UniqueID();	short	resources.h

Call	Return type	Header file
UnloadScrap();	long	scrap.h
UnloadSeg();	void	segload.h
UnmountVol();	OSErr	files.h
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
void	mrrel();	aux_rsrc.h
VRemove();	OSErr	retrace.h
WaitMouseUp();	Boolean	events.h
WaitNextEvent();	Boolean	events.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

C

D

E

F

G

H

I, J

R

S

T

U

V

W, X, Y, Z

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and Microsoft Word software. Proof pages were created on Apple LaserWriter® printers. Final pages were created on the Varityper VT600W imagesetter. Line art was created using Adobe Illustrator. POSTSCRIPT®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type 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.

Writer: Walt Bryant
Developmental Editor: George Truett
Production Supervisor: Josephine Manuele
Formatter: Teresa Lujan

Special thanks to David Berry, Mike Chow, Rick Daley, Mike Elola, Sharon Everson, Li Greiner, Winston Hendrickson, Andy Heninger, Sue Luttner, Dave Radcliffe, Rob Smith, George Towner, Justin Walker, and Kathleen Wallace

030-0787-A