EARLY MACINTOSH TECHNICAL INFORMATION

INSIDE MACINTOSH
A ROAD MAP

COMMENT

FIRST INSIDE MACINTOSH DOCUMENT WHICH MACINTOSH PROGRAMMERS READ, THIS LISTED ALL THE DIFFERENT INSIDE MACINTOSH CHAPTERS

AUTHOR

APPLE COMPUTER

DATE

10 SEPTEMBER 1984

SOURCE

DAVID T CRAIG • JANUARY 2004

MACINTOSH USER EDUCATION

INSIDE MACINTOSH: A ROAD MAP

/ROAD.MAP/ROAD

See Also:
Macintosh User Interface Guidelines
Macintosh Memory Management: An Introduction
Programming Macintosh Applications in Assembly Language
The Resource Manager: A Programmer's Guide
QuickDraw: A Programmer's Guide
The Font Manager: A Programmer's Guide
The Event Manager: A Programmer's Guide
The Window Manager: A Programmer's Guide
The Control Manager: A Programmer's Guide
The Menu Manager: A Programmer's Guide
TextEdit: A Programmer's Guide
The Dialog Manager: A Programmer's Guide
The Desk Manager: A Programmer's Guide
The Scrap Manager: A Programer's Guide
The Toolbox Utilities: A Programmer's Guide
Macintosh Packages: A Programmer's Guide
The Memory Manager: A Programmer's Guide
The Segment Loader: A Programmer's Guide
The File Manager: A Programmer's Guide
Printing from Macintosh Applications
The Device Manager: A Programmer's Guide
The Sound Driver: A Programmer's Guide
The Vertical Retrace Manager: A Programmer's Guide
The Operating System Utilities: A Programmer's Guide
The Structure of a Macintosh Application
Putting Together a Macintosh Application
Index to Technical Documentation

Modification History:	First Draft (ROM 4.4)	Caroline Rose	8/8/83
	Second Draft (ROM 7)	Caroline Rose	12/22/83
	Third Draft	Caroline Rose	9/1Ø/84

ABSTRACT

This manual introduces you to the Macintosh technical documentation and the "inside" of Macintosh: the Operating System and other routines that your application program will call. It will help you figure out which software you need to learn more about and how to proceed with the rest of the documentation. It also presents a simple example program.

Since the last draft, changes and additions have been made to the overviews, an example program has been added, and the structure of a typical Inside Macintosh manual is discussed.

2 Inside Macintosh Road Map

TABLE OF CONTENTS

3 About This Manual
3 About Inside Macintosh
4 Everything You Know Is Wrong
4 Conventions
5 The Structure of a Typical Manual
6 Overview of the Software
6 The Toolbox and Other High-Level Software
1Ø The Operating System and Other Low-Level Software
11 A Simple Example Program
18 Where to Go From Here
19 Appendix: Resource Compiler Input for Example Program
2Ø Glossary

ABOUT THIS MANUAL

This manual introduces you to the Macintosh technical documentation and the "inside" of Macintosh: the Operating System and User Interface Toolbox routines that your application program will call. It will help you figure out which software you need to learn more about and how to proceed with the rest of the documentation. To orient you to the software, it presents a simple example program. *** Eventually it will become the preface and introductory chapter in the comprehensive Inside Macintosh manual. ***

ABOUT INSIDE MACINTOSH

Inside Macintosh *** (currently a set of separate manuals) *** tells you what you need to know to write software for the Macintosh. Although directed mainly toward programmers writing standard Macintosh applications, it also contains the information necessary for writing simple utility programs, desk accessories, device drivers, or any other Macintosh software. It includes:

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

It does **not** include:

- information about getting started as a developer (for that, see the Apple 32 Developer's Handbook, available from Apple Computer's Software Industry Relations)
- any information that's specific to the development system being used, except where indicated *** (The manual Putting Together a Macintosh Application will not be part of the final Inside Macintosh.) ***

The routines you'll need to call are written in assembly language, but they're also accessible from high-level languages. The development system currently available from Apple supports Lisa Pascal and includes Pascal interfaces to all the routines (except for a few that are called only from assembly language). Inside Macintosh documents these Pascal interfaces; if you're using a development system that supports a different high-level language, its documentation should tell you how to apply the information presented here to that system.

Inside Macintosh is intended to serve the needs of both Pascal and assembly-language programmers. Every routine is shown in its Pascal form (if it has one), but assembly-language programmers are told how to

translate this to assembly code. Information of interest only to assembly-language programmers is isolated and labeled so that Pascal programmers can conveniently skip it.

Familiarity with Lisa Pascal is recommended for all readers, since it's used for most examples. Lisa Pascal is described in the Pascal Reference Manual for the Lisa. You should also be familiar with the basic information that's in Macintosh, the owner's guide, and have some experience using a standard Macintosh application (such as MacWrite).

Everything You Know Is Wrong

On an innovative system like the Macintosh, programs don't look quite the way they do on other systems. For example, instead of carrying out a sequence of steps in a predetermined order, your program is driven primarily by user actions (such as clicking and typing) whose order cannot be predicted. You'll probably find that many of your preconceptions about how to write applications don't apply here. Because of this, and because of the sheer volume of information in Inside Macintosh, it's essential that you read the Road Map *** (the rest of this manual) ***. It will help you get oriented and figure out where to go next.

Conventions

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

(note)
A note that may be interesting or useful

(warning)
A point you need to be cautious about

Assembly-language note: A note of interest to assembly-language programmers only *** (in final manual, may instead be a shaded note or warning) ***

[No trap macro]

This notation is of interest only to assembly-language programmers *** (may be shaded in final manual) ***; it's explained along with other general information on using assembly language in the manual Programming Macintosh Applications in Assembly Language.

The Structure of a Typical Manual

*** This section refers to "manuals" for the time being; when the individual manuals become chapters of Inside Macintosh, this will be changed to "chapters". ***

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

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

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

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

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

For review and quick reference, each manual ends with a summary of the subject matter, including the entire Pascal interface and a subsection for assembly-language programmers. *** For now, this is followed by a glossary of terms used in the manual. Eventually, all the individual glossaries will be combined into one. ***

OVERVIEW OF THE SOFTWARE

The routines available for use in Macintosh programs are divided according to function, into what are in most cases called "managers" of the application feature that they support. As shown in Figure 1 on the following page, most are part of either the Operating System or the User Interface Toolbox and are in the Macintosh ROM.

The Operating System is at the lowest level; it does basic tasks such as input and output, memory management, and interrupt handling. The User Interface Toolbox is a level above the Operating System; it helps you implement the standard Macintosh user interface in your application. The Toolbox calls the Operating System to do low-level operations, and you'll also call the Operating System directly yourself.

RAM-based software is available as well. In most cases this software performs specialized operations that aren't integral to the user interface but may be useful to some applications (such as printing and floating-point arithmetic).

The Toolbox and Other High-Level Software

The Macintosh User Interface Toolbox provides a simple means of constructing application programs that conform to the standard Macintosh user interface. By offering a common set of routines that every application calls to implement the user interface, the Toolbox not only ensures familiarity and consistency for the user but also helps reduce the application's code size and development time. At the same time, it allows a great deal of flexibility: an application can use its own code instead of a Toolbox call wherever appropriate, and can define its own types of windows, menus, controls, and desk accessories.

Figure 2 shows the various parts of the Toolbox in rough order of their relative level. There are many interconnections between these parts; the higher ones often call those at the lower levels. A brief description of each part is given below, to help you figure out which ones you'll need to learn more about. Details are given in the Inside Macintosh documentation on that part of the Toolbox. The basic Macintosh terms used below are explained in the Macintosh owner's guide.

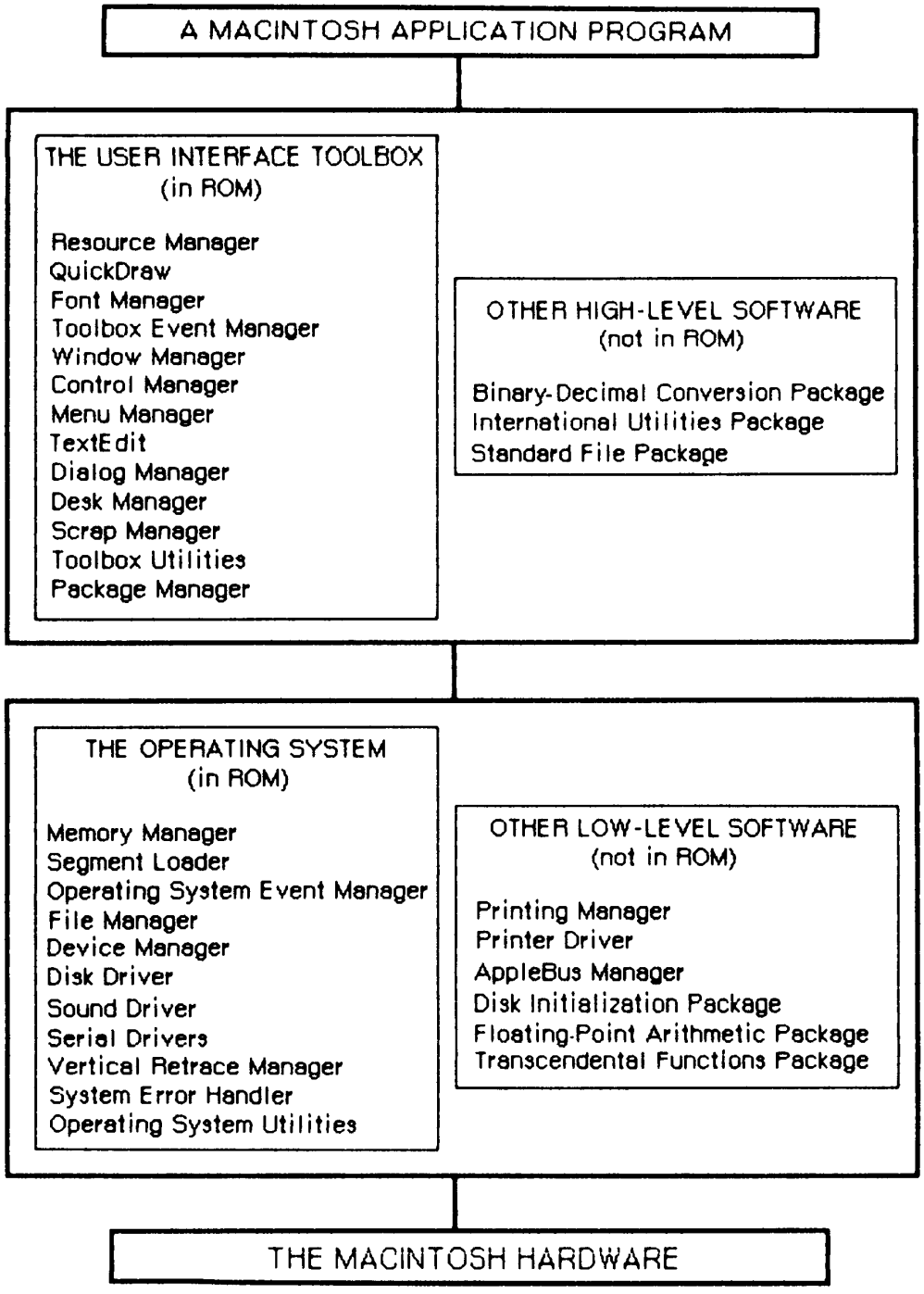


Figure 1. Overview

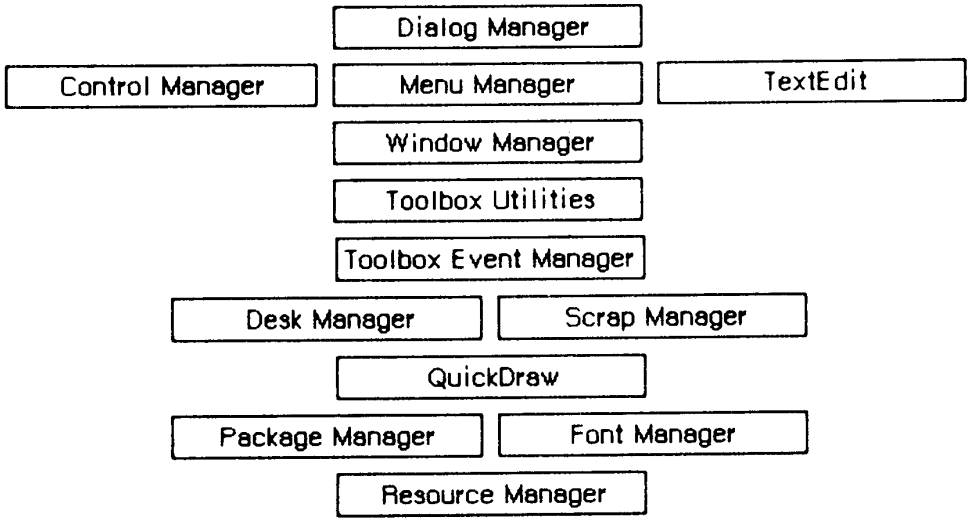


Figure 2. Parts of the Toolbox

To keep the data of an application separate from its code, making the data easier to modify and easier to share among applications, the Toolbox includes the Resource Manager. The Resource Manager lets you, for example, store menus separately from your code so that they can be edited or translated without requiring recompilation of the code. It also allows you to get standard data, such as the I-beam pointer for inserting text, from a shared system file. When you call other parts of the Toolbox that need access to the data, they call the Resource Manager. Although most applications never need to call the Resource Manager directly, an understanding of the concepts behind it is essential because they're basic to so many other Toolbox operations.

Graphics are an important part of every Macintosh application. All graphic operations on the Macintosh are performed by QuickDraw. To draw something on the screen, you'll often call one of the other parts of the Toolbox, but it will in turn call QuickDraw. You'll also call QuickDraw directly, usually to draw inside a window, or just to set up constructs like rectangles that you'll need when making other Toolbox calls. QuickDraw's underlying concepts, like those of the Resource Manager, are important for you to understand.

Graphics include text as well as pictures. To draw text, QuickDraw calls the Font Manager, which does the background work necessary to make a variety of character fonts available in various sizes and styles. Unless your application includes a font menu, you need to know only a minimal amount about the Font Manager.

An application decides what to do from moment to moment by examining input from the user in the form of mouse and keyboard actions. It learns of such actions by repeatedly calling the Toolbox Event Manager (which in turn calls another, lower-level Event Manager in the Operating System). The Toolbox Event Manager also reports occurrences within the application that may require a response, such as when a

window that was overlapped becomes exposed and needs to be redrawn.

All information presented by a standard Macintosh application appears in windows. To create windows, activate them, move them, resize them, or close them, you'll call the Window Manager. It keeps track of overlapping windows, so you can manipulate windows without concern for how they overlap. For example, the Window Manager tells the Toolbox Event Manager when to inform your application that a window has to be redrawn. Also, when the user presses the mouse button, you call the Window Manager to learn which part of which window it was pressed in, or whether it was pressed in the menu bar or a desk accessory.

Any window may contain controls, such as buttons, check boxes, and scroll bars. You create and manipulate controls with the Control Manager. When you learn from the Window Manager that the user pressed the mouse button inside a window containing controls, you call the Control Manager to find out which control it was pressed in, if any.

A common place for the user to press the mouse button is, of course, in the menu bar. You set up menus in the menu bar by calling the Menu Manager. When the user gives a command, either from a menu with the mouse or from the keyboard with the Command key, you call the Menu Manager to find out which command was given.

To accept text typed by the user and allow the standard editing capabilities, including cutting and pasting text within a document via the Clipboard, your application can call TextEdit. TextEdit also handles basic formatting such as word wraparound and justification. You can use it just to display text if you like.

When an application needs more information from the user about a command, it presents a dialog box. In case of errors or potentially dangerous situations, it alerts the user with a box containing a message or with sound from the Macintosh's speaker (or both). To create and present dialogs and alerts, and find out the user's responses to them, you call the Dialog Manager.

Every Macintosh application should support the use of desk accessories. The user opens desk accessories through the Apple menu, which you set up by calling the Menu Manager. When you learn that the user has pressed the mouse button in a desk accessory, you pass that information on to the accessory by calling the Desk Manager. The Desk Manager also includes routines that you must call to ensure that desk accessories work properly.

As mentioned above, you can use TextEdit to implement the standard text editing capability of cutting and pasting via the Clipboard in your application. To allow the use of the Clipboard for cutting and pasting text or graphics between your application and another application or a desk accessory, you need to call the Scrap Manager.

Some generally useful operations such as fixed-point arithmetic, string manipulation, and logical operations on bits may be performed with the Toolbox Utilities.

The final part of the Toolbox, the Package Manager, lets you use RAM-based software called packages. The Standard File Package will be called by every application whose File menu includes the standard commands for saving and opening documents; it presents the standard user interface for specifying the document. Some of the Macintosh packages can be seen as extensions to the Toolbox Utilities: the Binary-Decimal Conversion Package converts integers to decimal strings and vice versa, and the International Utilities Package gives you access to country-dependent information such as the formats for numbers, currency, dates, and times.

The Operating System and Other Low-Level Software

The Macintosh Operating System provides the low-level support that applications need in order to use the Macintosh hardware. As the Toolbox is your program's interface to the user, the Operating System is its interface to the Macintosh.

The Memory Manager dynamically allocates and releases memory for use by applications and by the other parts of the Operating System. Most of the memory that your program uses is in an area called the heap; the code of the program itself occupies space in the heap. Memory space in the heap must be obtained from the Memory Manager.

The Segment Loader is the part of the Operating System that loads program code into memory to be executed. Your program can be loaded all at once, or you can divide it up into dynamically loaded segments to economize on memory usage. The Segment Loader also serves as a bridge between the Finder and your application, letting you know whether the application has to open or print a document on the desktop when it starts up.

Low-level, hardware-related events such as mouse-button presses and keystrokes are reported by the Operating System Event Manager. (The Toolbox Event Manager then passes them to the application, along with higher-level, software-generated events added at the Toolbox level.) Your program will ordinarily deal only with the Toolbox Event Manager and rarely call the Operating System Event Manager directly.

File I/O is supported by the File Manager, and device I/O by the Device Manager. The task of making the various types of devices present the same interface to the application is performed by specialized device drivers. The Operating System includes three built-in drivers:

- The Disk Driver controls data storage and retrieval on 3 1/2-inch disks.
- The Sound Driver controls sound generation, including music composed of up to four simultaneous tones.
- The Serial Driver reads and writes asynchronous data through the two serial ports, providing communication between applications and serial peripheral devices such as a modem or printer.

The above drivers are all in ROM; other drivers are RAM-based. There's a Serial Driver in RAM as well as the one in ROM, and there's a Printer Driver in RAM that enables applications to print information on any variety of printer via the same interface (called the Printing Manager). The AppleBus Manager is an interface to a pair of RAM drivers that enable programs to send and receive information via an AppleBus network. More RAM drivers can be added independently or built on the existing drivers. For example, the Printer Driver was built on the Serial Driver, and a music driver could be built on the Sound Driver.

The Macintosh video circuitry generates a vertical retrace interrupt 60 times a second. An application can schedule routines to be executed at regular intervals based on this "heartbeat" of the system. The Vertical Retrace Manager handles the scheduling and execution of tasks during the vertical retrace interrupt.

If a fatal error occurs while your application is running (for example, if it runs out of memory), the System Error Handler assumes control. The System Error Handler displays a box containing an error message and provides a mechanism for the user to start up the system again or resume execution of the application.

The Operating System Utilities perform miscellaneous operations such as getting the date and time, finding out the user's preferred speaker volume and other preferences, and doing simple string comparison. (More sophisticated string comparison routines are available in the International Utilities Package.)

Finally, there are three Macintosh packages that perform low-level operations: the Disk Initialization Package, which the Standard File Package calls to initialize and name disks; the Floating-Point Arithmetic Package; and the Transcendental Functions Package.

A SIMPLE EXAMPLE PROGRAM

To illustrate various commonly used parts of the software, this section presents an extremely simple example of a Macintosh application program. Though too simple to be practical, this example shows the overall structure that every application program will have, and it does many of the basic things every application will do. By looking it over, you can become more familiar with the software and see how your own program code will be structured.

The example program's source code is shown in Figure 4, which begins on page 15. A lot of comments are included so that you can see which part of the Toolbox or Operating System is being called and what operation is being performed. These comments, and those that follow below, may contain terms that are unfamiliar to you, but for now just read along to get the general idea. All the terms are explained at length within Inside Macintosh. If you want more information right away, you can look up the terms in the Glossary or the Index *** (currently the

individual glossaries in the various manuals, and the manual Index to Technical Documentation) ***

The application, called Samp, displays a single, fixed-size window in which the user can enter and edit text (see Figure 3). It has three menus: the standard Apple menu, from which desk accessories can be chosen; a File menu, containing only a Quit command; and an Edit menu, containing the standard editing commands Undo, Cut, Copy, Paste, and Clear. The Backspace key may be used to delete, and Shift-clicking will extend or shorten a selection. The user can move the document window around the desktop by dragging it by its title bar.

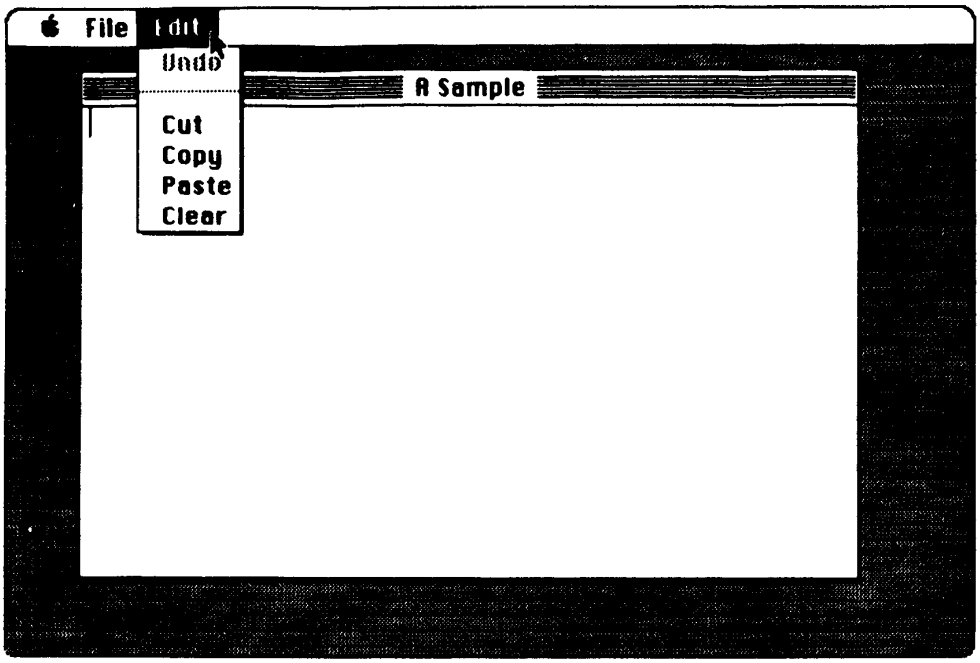


Figure 3. The Samp Application

The Undo command doesn't work in the application's document window, but it and all the other editing commands do work in any desk accessories that allow them (Note Pad, for example). Some standard features this simple example doesn't support are as follows:

- Text cannot be cut (or copied) and pasted between the document and a desk accessory.

- The pointer remains an arrow rather than changing to an I-beam within the document.

- The standard keyboard equivalents--Command-Z, X, C, and V for Undo, Cut, Copy, and Paste--aren't in the Edit menu. They won't work in the document window (but they will work in desk accessories that allow those commands).

Because the File menu contains only a Quit command, the document can't be saved or printed. Also, the application doesn't have an "About

Samp" command as the first item in its Apple menu, nor does it present any dialog boxes or alerts. All of these features and more are illustrated in programs in the Sample Macintosh Programs manual *** (forthcoming) ***.

In addition to the code shown in Figure 4, the Samp application has a resource file that includes the data listed below. The program uses the numbers in the second column to identify the resources; for example, it makes a Menu Manager call to get menu number 128 from the resource file.

Resource	Resource ID	Description
Menu	128	Menu with the apple symbol as its title and no commands in it
Menu	129	File menu with one command, Quit
Menu	13Ø	Edit menu with the commands Undo (dimmed), Cut, Copy, Paste, and Clear, in that order, with a dividing line between Undo and Cut
Window template	128	Document window without a size box; top left corner of (5Ø,4Ø) on QuickDraw's coordinate plane, bottom right corner of (3ØØ,45Ø); title "A Sample"; no close box

Each menu resource also contains a "menu ID" that's used to identify the menu when the user chooses a command from it; for all three menus, this ID is the same as the resource ID.

(note)

To create a resource file with the above contents, you can use the Resource Editor *** (for now, the Resource Compiler) *** or any similar program that may be available on the development system you're using; for more information, see the documentation for that program. *** The Resource Compiler is documented in Putting Together a Macintosh application. The Resource Compiler input file for the Samp application is shown in the appendix of this manual. All these files will eventually be provided to developers by Macintosh Technical Support. ***

The program starts with a USES clause that specifies all the necessary Pascal interface files. (The names shown are for the Lisa Workshop development system, and may be different for other systems.) This is followed by declarations of some useful constants, to make the source code more readable. Then there are a number of variable declarations, some having simple Pascal data types and others with data types defined in the Pascal interface files (like Rect and WindowPtr). Variables used in the program that aren't declared here are global variables defined in the interface to QuickDraw.

The variable declarations are followed by two procedure declarations: SetUpMenus and DoCommand. You can understand them better after looking

at the main program and seeing where they're called.

The program begins with a standard initialization sequence. Every application will need to do this same initialization (in the order shown), or something close to it.

Additional initialization needed by the program follows. This includes setting up the menus and the menu bar (by calling SetUpMenus) and creating the application's document window (reading its description from the resource file and displaying it on the screen).

The heart of every application program is its main event loop, which repeatedly calls the Toolbox Event Manager to get events and then responds to them as appropriate. The most common event is a press of the mouse button; depending on where it was pressed, as reported by the Window Manager, the sample program may execute a command, move the document window, make the window active, or pass the event on to a desk accessory. The DoCommand procedure takes care of executing a command; it looks at information received by the Menu Manager to determine which command to execute.

Besides events resulting directly from user actions such as pressing the mouse button or a key on the keyboard, events are detected by the Window Manager as a side effect of those actions. For example, when a window changes from active to inactive or vice versa, the Window Manager tells the Toolbox Event Manager to report it to the application program. A similar process happens when all or part of a window needs to be updated (redrawn). The internal mechanism in each case is invisible to the program, which simply responds to the event when notified.

The main event loop terminates when the user takes some action to leave the program--in this case, when the Quit command is chosen.

That's it! Of course, the program structure and level of detail will get more complicated as the application becomes more complex, and every actual application will be more complex than this one. But each will be based on the structure illustrated here.

```
PROGRAM Samp;

{ Samp -- A small sample application written in Pascal by Macintosh User Education }
{ It displays a single, fixed-size window in which the user can enter and edit text. }

USES  {SU Obj/MemTypes } MemTypes,   {basic Memory Manager data types}
      {SU Obj/QuickDraw} QuickDraw,  {interface to QuickDraw}
      {SU Obj/OSIntf   } OSIntf,     {interface to the Operating System}
      {SU Obj/ToolIntf } ToolIntf;   {interface to the Toolbox}

CONST appleID = 128;      {resource IDs/menu IDs for Apple, File, and Edit menus}
      fileID = 129;
      editID = 130;
      appleM = 1;         {index for each menu in array of menu handles}
      fileM = 2;
      editM = 3;
      menuCount = 3;      {total number of menus}
      windowID = 128;     {resource ID for application's window}
      undoCommand = 1;    {menu item numbers identifying commands in Edit menu}
      cutCommand = 3;
      copyCommand  = 4;
      pasteCommand = 5;
      clearCommand = 6;

VAR myMenus: ARRAY [1..menuCount] OF MenuHandle;
    dragRect, txRect: Rect;
    extended, doneFlag: BOOLEAN;
    myEvent: EventRecord;
    wRecord: WindowRecord;
    myWindow, whichWindow: WindowPtr;
    textH: TEHandle;


PROCEDURE SetUpMenus;
{ Set up menus and menu bar }

  VAR i: INTEGER;

  BEGIN
  myMenus[appleM] := GetMenu(appleID); {read Apple menu from resource file}
  AddResMenu(myMenus[appleM], 'DRVR');  {add desk accessory names to Apple menu}
  myMenus[fileM] := GetMenu(fileID);   {read File menu from resource file}
  myMenus[editM] := GetMenu(editID);   {read Edit menu from resource file}
  FOR i:=1 TO menuCount DO InsertMenu(myMenus[i],0);  {install menus in menu bar }
  DrawMenuBar;                                        { and draw menu bar}
  END;     {of SetUpMenus}


PROCEDURE DoCommand (mResult: LONGINT);
{ Execute command specified by mResult, the result of MenuSelect }

  VAR theItem, temp: INTEGER;
      name: Str255;

  BEGIN
  theItem := LoWord(mResult);             {call Toolbox Utility routine to get }
                                          { menu item number from low-order word}
```

Figure 4. Example Program

```
  CASE HiWord(mResult) OF                  {case on menu ID in high-order word}

  appleID:
    BEGIN                                   {call Menu Manager to get desk accessory }
    GetItem(myMenus[appleM], theItem, name); { name, and call Desk Manager to open }
    temp := OpenDeskAcc(name);              { accessory (OpenDeskAcc result not used)}
    SetPort(myWindow);                      {call QuickDraw to restore application }
    END;    {of appleID}                    { window as grafPort to draw in (may have }
                                            { been changed during OpenDeskAcc)}
  fileID:
    doneFlag := TRUE;               {quit (main loop repeats until doneFlag is TRUE)}

  editID:
    BEGIN                           {call Desk Manager to handle editing command if }
    IF NOT SystemEdit(theItem-1)    { desk accessory window is the active window}
      THEN                          {application window is the active window}
        CASE theItem OF             {case on menu item (command) number}

        cutCommand:     TECut(textH);       {call TextEdit to handle command}
        copyCommand:    TECopy(textH);
        pasteCommand:   TEPaste(textH);
        clearCommand:   TEDelete(textH);

        END;    {of item case}
    END;    {of editID}

  END;    {of menu case}            {to indicate completion of command, call }
  HiliteMenu(0);                    { Menu Manager to unhighlight menu title }
                                    { (highlighted by MenuSelect)}
  END;  {of DoCommand}


BEGIN    { main program }
InitGraf(@thePort);             {initialize QuickDraw}
InitFonts;                      {initialize Font Manager}
FlushEvents(everyEvent,0);      {call OS Event Manager to discard any previous events}
InitWindows;                    {initialize Window Manager}
InitMenus;                      {initialize Menu Manager}
TEInit;                         {initialize TextEdit}
InitDialogs(NIL);               {initialize Dialog Manager}
InitCursor;                     {call QuickDraw to make cursor (pointer) an arrow}

SetUpMenus;                     {set up menus and menu bar}
WITH screenBits.bounds DO       {call QuickDraw to set dragging boundaries; ensure at }
  SetRect(dragRect,4,24,right-4,bottom-4); { least 4 by 4 pixels will remain visible}
doneFlag := FALSE;              {flag to detect when Quit command is chosen}

myWindow := GetNewWindow(windowID,@wRecord,POINTER(-1)); {put up application window}
SetPort(myWindow);              {call QuickDraw to set current grafPort to this window}
txRect := thePort^.portRect;    {rectangle for text in window; call QuickDraw to bring }
InsetRect(txRect,4,0);          { it in 4 pixels from left and right edges of window}
textH := TENew(txRect,txRect);          {call TextEdit to prepare for receiving text}

{ Main event loop }
REPEAT                                  {call Desk Manager to perform any periodic }
  SystemTask;                           { actions defined for desk accessories}
  TEIdle(textH);                        {call TextEdit to make vertical bar blink}
```

Figure 4. Example Program (continued)

```
  IF GetNextEvent(everyEvent,myEvent) {call Toolbox Event Manager to get the next }
    THEN                              { event that the application should handle}
      CASE myEvent.what OF            {case on event type}

      mouseDown:            {mouse button down:  call Window Manager to learn where}
        CASE FindWindow(myEvent.where,whichWindow) OF

        inMenuBar:          {menu bar:  call Menu Manager to learn which command; }
          DoCommand(MenuSelect(myEvent.where)); { then execute it}

        inSysWindow:        {desk accessory window: call Desk Manager to handle it}
          SystemClick(myEvent,whichWindow);

        inDrag:             {title bar:  call Window Manager to drag}
          DragWindow(whichWindow,myEvent.where,dragRect);

        inContent:                              {body of application window: }
          BEGIN                                 { call Window Manager to check whether }
          IF whichWindow <> FrontWindow         { it's the active window and make it }
            THEN SelectWindow(whichWindow)      { active if not }
            ELSE
              BEGIN                             {it's already active:  call QuickDraw to }
              GlobalToLocal(myEvent.where);     { convert to window coordinates for }
                                                { TEClick, use Toolbox Utility BitAnd to }
              extended := BitAnd(myEvent.modifiers,shiftKey) <> 0;  { test for Shift }
              TEClick(myEvent.where,extended,textH);    { key down, and call TextEdit }
              END;                                      { to process the event}
          END;    {of inContent}

        END;    {of mouseDown}

      keyDown, autoKey:             {key pressed:  pass character to TextEdit}
          TEKey(CHR(BitAnd(myEvent.message,charCodeMask)),textH);

      activateEvt:
          BEGIN
          IF BitAnd(myEvent.modifiers,activeFlag) <> 0
            THEN                    {application window is becoming active: }
              BEGIN                 { call TextEdit to highlight selection }
              TEActivate(textH);    { or display blinking vertical bar, and call }
              DisableItem(myMenus[editM],undoCommand); { Menu Manager to disable }
              END                   { Undo (since application doesn't support Undo)}
            ELSE
              BEGIN                 {application window is becoming inactive: }
              TEDeactivate(textH);  { unhighlight selection or remove blinking }
              EnableItem(myMenus[editM],undoCommand);  { vertical bar, and enable }
              END;                  { Undo (since desk accessory may support it)}
          END;   {of activateEvt}

      updateEvt:                                {window appearance needs updating}
        BEGIN
        BeginUpdate(WindowPtr(myEvent.message)); {call Window Manager to begin update}
        EraseRect(thePort^.portRect);            {call QuickDraw to erase text area}
        TEUpdate(thePort^.portRect,textH);       {call TextEdit to update the text}
        EndUpdate(WindowPtr(myEvent.message));   {call Window Manager to end update}
        END;    {of updateEvt}

      END;    {of event case}

  UNTIL doneFlag;
  END.
```

Figure 4. Example Program (continued)

WHERE TO GO FROM HERE

*** This section refers to "manuals" for the time being; when the individual manuals become chapters of Inside Macintosh, this will be changed to "chapters". It also refers to the "order" of the manuals; this means the order of the documentation when it's combined into a single manual. For a list of what's been distributed so far and how it will be ordered, see the cover page of this manual. Anything not listed there hasn't been distributed yet by Macintosh User Education, but programmer's notes or other preliminary documentation may be available. ***

This section contains important directions for every reader of Inside Macintosh. It will help you figure out which manuals to read next.

The Inside Macintosh documentation is ordered in such a way that you can follow it if you read through it sequentially. Forward references are given wherever necessary to any additional information that you'll need in order to understand what's being discussed. Special-purpose information that can possibly be skipped is indicated as such. Most likely you won't need to read everything in each manual and can even skip entire manuals.

You should begin by reading the following:

1. Macintosh User Interface Guidelines. All Macintosh applications should follow these guidelines to ensure that the end user is presented with a consistent, familiar interface.

2. Macintosh Memory Management: An Introduction.

3. Programming Macintosh Applications in Assembly Language, if you're using assembly language. Depending on the debugging tools available on the development system you're using, it may also be helpful or necessary for Pascal programmers to read this manual. You'll also have to read it if you're creating your own development system and want to know how to write interfaces to the routines.

4. The documentation of the parts of the Toolbox that deal with the fundamental aspects of the user interface: the Resource Manager, QuickDraw, the Toolbox Event Manager, the Window Manager, and the Menu Manager.

Read the other manuals if you're interested in what they discuss, which you should be able to tell from the overviews in this "road map" and from the introductions to the manuals themselves. Each manual's introduction will also tell you what you should already know before reading that manual.

When you're ready to try something out, refer to the appropriate documentation for the development system you'll be using. *** (Lisa Workshop users, see Putting Together a Macintosh Application.) ***

APPENDIX: RESOURCE COMPILER INPUT FOR EXAMPLE PROGRAM

For Lisa Workshop users, this appendix shows the Resource Compiler input file used with the example program presented earlier. For more information on the format of the file, see Putting Together a Macintosh Application.

(note)
This entire appendix is temporary; it will not be part of the final Inside Macintosh manual, because all the information in that manual will be independent of the development system being used. Authors of the documentation for a particular development system may choose to show how the resource file for Samp would be created on that system.

```
*  SampR -- Resource Compiler input file for Samp application
*           written by Macintosh User Education

Work/Samp.Rsrc

Type MENU
  ,128 (4)
* the apple symbol
  \14

  ,129 (4)
  File
    Quit

  ,13Ø (4)
  Edit
    (Undo
    (-
    Cut
    Copy
    Paste
    Clear

Type WIND
  ,128 (36)
  A Sample
  5Ø 4Ø 3ØØ 45Ø
  Visible NoGoAway
  4
  Ø

Type SAMP = STR
  ,Ø
Samp Version 1.Ø -- September 4, 1984

Type CODE
  Work/SampL,Ø
```

9/10/84 Rose /ROAD.MAP/ROAD.3

GLOSSARY

AppleBus Manager: An interface to a pair of RAM drivers that enable programs to send and receive information via an AppleBus network.

Binary-Decimal Conversion Package: A Macintosh package for converting integers to decimal strings and vice versa.

Control Manager: The part of the Toolbox that provides routines for creating and manipulating controls (such as buttons, check boxes, and scroll bars).

Desk Manager: The part of the Toolbox that supports the use of desk accessories from an application.

device driver: A piece of software that controls a peripheral device and makes it present a standard interface to the application.

Device Manager: The part of the Operating System that supports device I/O.

Dialog Manager: The part of the Toolbox that provides routines for implementing dialogs and alerts.

Disk Driver: The device driver that controls data storage and retrieval on 3 1/2-inch disks.

Disk Initialization Package: A Macintosh package for initializing and naming new disks; called by the Standard File Package.

Event Manager: See Toolbox Event Manager or Operating System Event Manager.

File Manager: The part of the Operating System that supports file I/O.

Font Manager: The part of the Toolbox that supports the use of various character fonts for QuickDraw when it draws text.

heap: An area of memory in which space can be allocated and released on demand, using the Memory Manager.

International Utilities Package: A Macintosh package that gives you access to country-dependent information such as the formats for numbers, currency, dates, and times.

main event loop: In a standard Macintosh application program, a loop that repeatedly calls the Toolbox Event Manager to get events and then responds to them as appropriate.

Memory Manager: The part of the Operating System that dynamically allocates and releases memory space in the heap.

Menu Manager: The part of the Toolbox that deals with setting up menus and letting the user choose from them.

Operating System: The lowest-level software in the Macintosh. It does basic tasks such as I/O, memory management, and interrupt handling.

Operating System Event Manager: The part of the Operating System that reports hardware-related events such as mouse-button presses and keystrokes.

Operating System Utilities: Operating System routines that perform miscellaneous tasks such as getting the date and time, finding out the user's preferred speaker volume and other preferences, and doing simple string comparison.

package: A set of routines and data types that's stored as a resource and brought into memory only when needed.

Package Manager: The part of the Toolbox that lets you access Macintosh RAM-based packages.

Printer Driver: The device driver for the currently installed printer.

Printing Manager: The routines and data types that enable applications to communicate with the Printer Driver to print on any variety of printer via the same interface.

QuickDraw: The part of the Toolbox that performs all graphic operations on the Macintosh screen.

resource: Data used by an application (such as menus, fonts, and icons), and also the application code itself.

Resource Manager: The part of the Toolbox that reads and writes resources.

Scrap Manager: The part of the Toolbox that enables cutting and pasting between applications, desk accessories, or an application and a desk accessory.

Segment Loader: The part of the Operating System that loads the code of an application into memory, either as a single unit or divided into dynamically loaded segments.

Serial Driver: The device driver that controls communication, via serial ports, between applications and serial peripheral devices.

Sound Driver: The device driver that controls sound generation in an application.

Standard File Package: A Macintosh package for presenting the standard user interface when a file is to be saved or opened.

System Error Handler: The part of the Operating System that assumes control when a fatal error (such as running out of memory) occurs.

TextEdit: The part of the Toolbox that supports the basic text entry and editing capabilities of a standard Macintosh application.

Toolbox: Same as User Interface Toolbox.

Toolbox Event Manager: The part of the Toolbox that allows your application program to monitor the user's actions with the mouse, keyboard, and keypad.

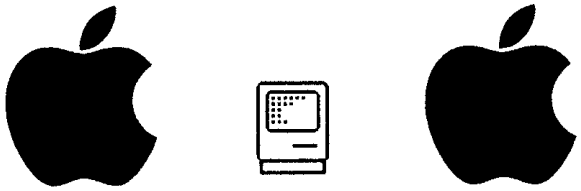
Toolbox Utilities: The part of the Toolbox that performs generally useful operations such as fixed-point arithmetic, string manipulation, and logical operations on bits.

User Interface Toolbox: The software in the Macintosh ROM that helps you implement the standard Macintosh user interface in your application.

vertical retrace interrupt: An interrupt generated 60 times a second by the Macintosh video circuitry while the beam of the display tube returns from the bottom of the screen to the top.

Vertical Retrace Manager: The part of the Operating System that schedules and executes tasks during the vertical retrace interrupt.

Window Manager: The part of the Toolbox that provides routines for creating and manipulating windows.



END OF DOCUMENT