CHAPTER 1

Introduction to the Macintosh Toolbox

Contents

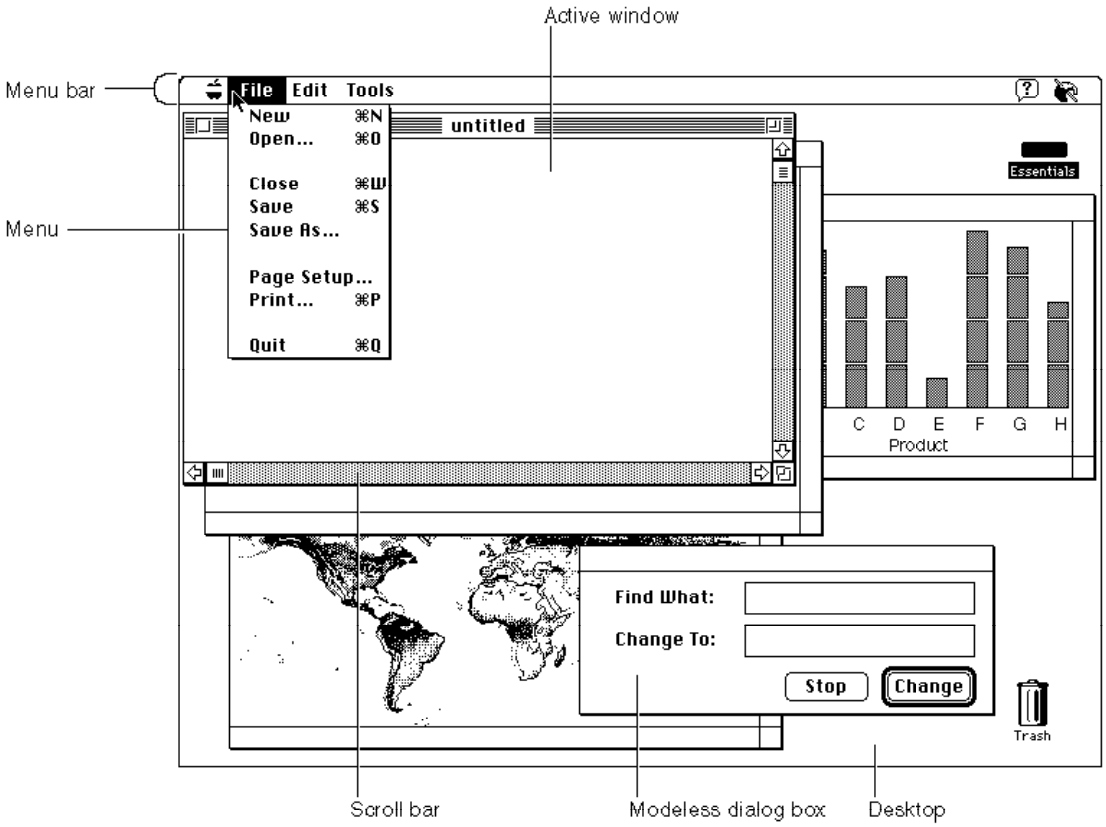
This chapter presents an introduction to the features provided by the Macintosh Toolbox. The Macintosh Toolbox is a collection of system software routines that your application can use to present a consistent and standard interface to the user; these routines also allow you to simplify other tasks your application might need to perform.

A typical Macintosh application presents a friendly, intuitive, easy-to-use, visual interface to the user. The careful design of a Macintosh application gives users the freedom to perform actions and accomplish tasks according to their needs. The idea behind this careful design is to put the user in control. In general, the user of a Macintosh application should always be free to choose the next action he or she will perform. (This is the basic tenet of the event loop and is explained in more detail in the chapter "Event Manager" in this book.)

Figure 1-1 shows the screen as it might appear when a user is interacting with a typical Macintosh application, such as SurfWriter. The SurfWriter application is an application that lets a user do simple text editing. Like most Macintosh applications, the SurfWriter application uses

- menus to let the user choose commands
- windows to allow the user to enter and edit information
- scroll bars to allow the user to view more information in a window

Figure 1-1 The SurfWriter application with multiple windows on the desktop



- other controls (such as the Change button) to let the user control various settings or options
- dialog boxes to solicit information from the user

You can create an application that incorporates these user-interface elements and that helps users accomplish specific tasks by taking advantage of the routines provided by the Macintosh Toolbox.

Overview of the Macintosh Toolbox

Macintosh system software contains a powerful set of routines that your application can use to create windows, manage menus, paint objects, display text, open files, share data between programs, and print files, as well as perform many other helpful tasks.

The Macintosh Toolbox encompasses a number of system software routines, most (but not all) of which help present your application's interface to the user. Some of these routines include those provided by the Event Manager, Menu Manager, Window Manager, Control Manager, Dialog Manager, Help Manager, Resource Manager, and Scrap Manager.

You can directly call these routines from within your application. By using system software routines, you can take advantage of the many tasks they can perform for you, and you can concentrate on the parts of your application that are specific to your particular product.

Using the Macintosh Toolbox, you can

- respond to user actions, such as mouse actions or keyboard input
- create and display menus
- create and display windows, alert boxes, and dialog boxes
- create and display controls in windows, alert boxes, and dialog boxes
- create icons for your application and its documents

This book, *Macintosh Toolbox Essentials*, describes these fundamental elements of a Macintosh application. *Inside Macintosh: More Macintosh Toolbox* describes additional features of a Macintosh application, including how you can

- create help balloons for your application's menus, windows, and dialog boxes
- support copy and paste
- specify characteristics of your application's menus, windows, controls, dialog boxes, and help balloons in resources so that you can more easily localize your application

The best Macintosh applications are designed according to the guidelines in *Macintosh Human Interface Guidelines*. You should always design your application so that it meets the needs of its users and responds in consistent and expected ways. *Macintosh Human Interface Guidelines* describes

- the philosophy and the design principles behind the Macintosh interface
- the parts of the Macintosh interface including the interface elements and behaviors
- ways to do human interface design for Macintosh products

You can often get valuable feedback on the design of your application by performing user testing. Do usability testing of your application early and often in the development phase of your product.

Events

At the core of every Macintosh application is the application's event loop. The event loop is that piece of code in an application that processes and responds to user actions and other events. You can use the Event Manager to retrieve information about these actions. For example, you can get information that tells your application whether the user pressed a key or the mouse button, whether one of your application's windows needs updating as a result of the user moving windows, or whether some other hardware or software action requires a response from your application.

You should structure your application so that it can respond to events and so that the user is able to perform tasks in any order. For example, a user should be able to type text in a window, select a graphic and copy it, open a new document, paste in the graphic, open another document, and then go back to the first window to select text and change its typeface, size, or style.

Your application should respond to events in a way that lets the user switch between your application and others whenever the user chooses to do so (for example, by clicking in a window belonging to another application). Your application should also yield time to other applications when it isn't busy. System software provides a cooperative multitasking environment that allows users to switch between many open applications and that allows applications to receive available processing time when other applications aren't using the processor. System software coordinates the scheduling of processing time between your application and other applications.

You can also let your application communicate with other applications in order to request services or information from another application or to provide services to other applications. You can use the Event Manager or Apple Event Manager to do this.

The chapter "Event Manager" in this book describes how to structure your event loop and event-handling code to receive notification of user actions and changes in the processing status of your application, how to communicate with other applications, and how to respond to these events.

Menus

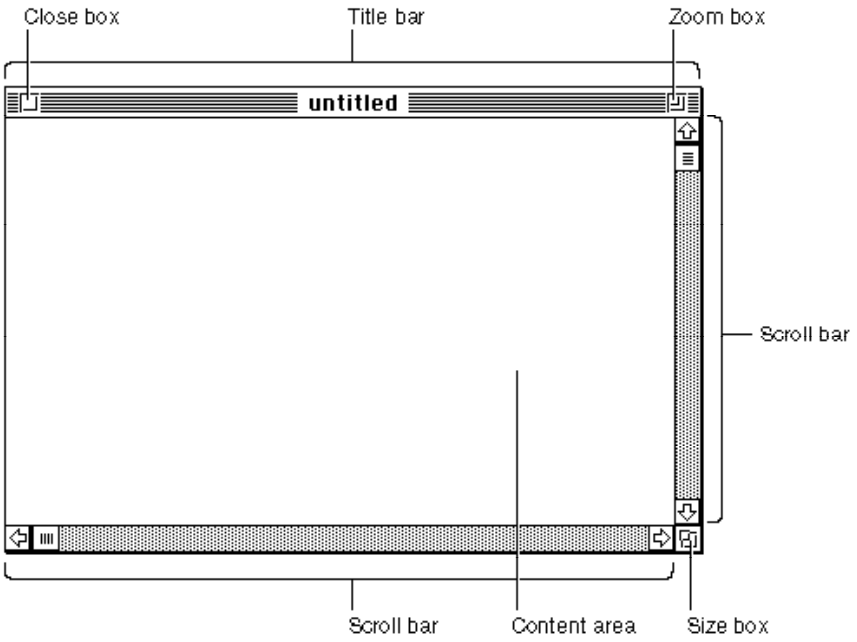
Menus are an important part of the design of a Macintosh application. Menus let users view or choose an item from a list of choices or commands that your application provides. You design your menus according to the tasks or actions your application performs. All applications should support the Apple, File, Edit, Help, Keyboard, and Application menus. Figure 1-1 on page 1-3 shows the File menu of the SurfWriter application.

System software makes it easy for you to create pull-down, hierarchical, and pop-up menus. The chapter "Menu Manager" in this book describes how to create your application's menus, set up your menu bar, display menus, and respond to the user's choice of an item from a menu.

Windows

Most applications interact with the user through windows. Figure 1-2 shows a common window and its elements. The chapter "Window Manager" in this book describes the types of windows your application can create and how to respond to user actions involving windows.

Figure 1-2 A typical window



The user typically has one or more windows on the desktop, often from a number of different applications. Although the user can have multiple windows on the desktop, only one window is the active window. The active window is the window that appears frontmost on the desktop and is identified by racing stripes in its title bar. Figure 1-2

shows an active window; in Figure 1-1 on page 1-3, the window titled SalesReport is an inactive window.

All keyboard activity is directed toward the active window. You should make sure that your application follows the human interface guidelines regarding active and inactive windows. For example, you should show the scroll bars and highlight any selection in an active window belonging to your application; you should hide the scroll bars and remove highlighting from any selection in an inactive window belonging to your application. The menu bar of your application also should always reflect the state of your application's active window—that is, your application should enable only those menu commands that pertain to the active window.

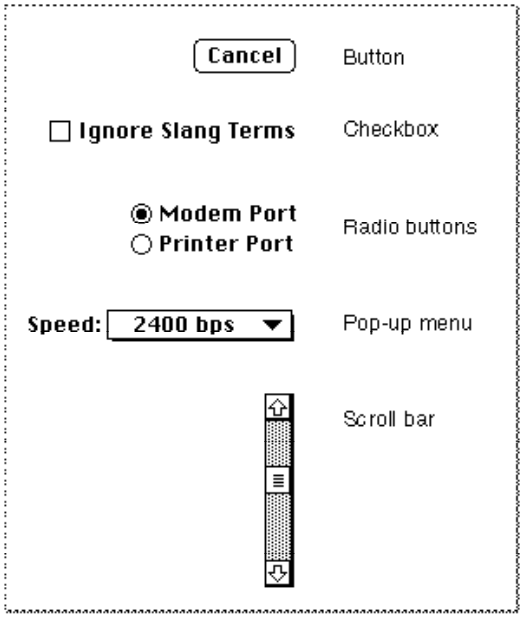
You can use system software routines to assist you when your application needs to create, move, size, zoom, or update the contents of your window. The chapter "Window Manager" in this book describes how you can accomplish these tasks.

Controls

Most windows and dialog boxes contain controls. Controls are onscreen objects that the user can manipulate with the mouse to cause an immediate action from your application or to change settings in order to modify a future action.

Buttons, checkboxes, radio buttons, pop-up menus, and scroll bars are examples of common controls used by most applications. Checkboxes, radio buttons, and pop-up menus are most often used in dialog boxes; buttons are most often used in alert boxes or dialog boxes; scroll bars are most often used in windows. Figure 1-3 illustrates these types of controls.

Figure 1-3 Common controls



A button appears as a rounded rectangle with a title centered inside. Use a button to perform an instantaneous action when the user clicks the button, such as completing operations defined by a dialog box or acknowledging an error message in an alert box.

A checkbox appears as a small square with a title beside it; the box contains an X when the setting associated with the box is on and is empty when the setting is off. Use a checkbox to indicate an option that must be either off or on.

A radio button appears as a circle with a title beside it; the circle contains a small black dot when the setting associated with the radio button is on and is empty when the setting is off. Radio buttons are similar to checkboxes in that they retain and display an on-or-off setting; however, only one radio button in a group of radio buttons should be on at any one time. You must decide how to group your radio buttons, and your application must ensure that only one radio button in a group is on.

A pop-up menu is a menu that appears in a dialog box or window. You can use pop-up menus as an alternative to radio buttons, to allow the user to select from a list of choices or settings.

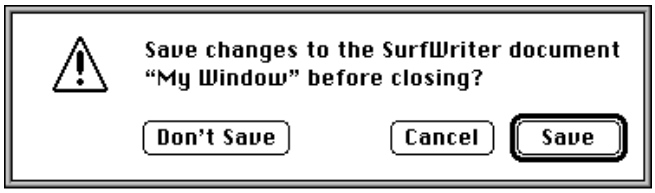
A scroll bar appears as a light gray rectangle that has scroll arrows at each end of the rectangle. A window can have a horizontal scroll bar, a vertical scroll bar, or both. You can use scroll bars to let the user change the portion of a document that the user can view within a window.

You can track and respond to user actions in controls, redraw controls, and manipulate controls using Control Manager routines. You usually use the Dialog Manager to handle most controls in dialog boxes or alert boxes for you. The chapter “Control Manager” in this book describes how to create controls (with special emphasis on creating and using scroll bars in windows), and the chapter “Dialog Manager” in this book provides additional information about how to create controls in dialog boxes and alert boxes.

Alert Boxes and Dialog Boxes

In addition to standard windows, your application typically also uses alert boxes and dialog boxes. An alert box is a window that your application displays on the screen to warn the user or to report an error to the user. An alert box typically consists of text describing the situation and buttons for the user to acknowledge or rectify the problem. Figure 1-4 shows an alert box that the SurfWriter application displays when the user attempts to close a window without saving the document. The alert box gives the user a chance to save the document before the SurfWriter application closes the window; this prevents the user from accidentally losing data.

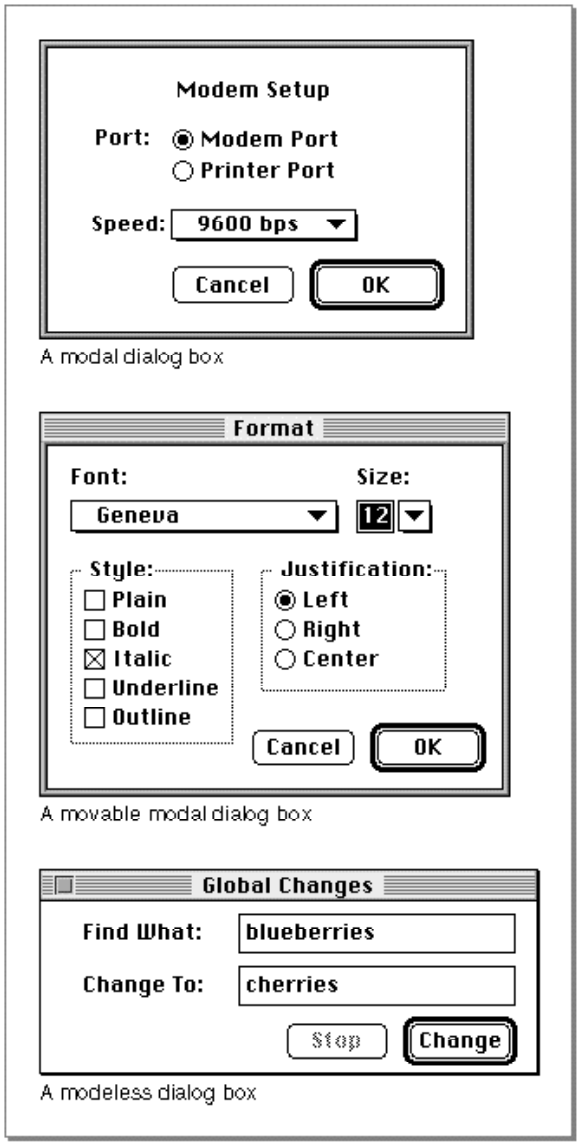
Figure 1-4 An alert box



A dialog box is a window that you can use for the specific purpose of soliciting additional information from the user. The Dialog Manager provides routines to help you display dialog boxes and provides standard and consistent methods of interacting with the user. Dialog boxes can contain editable text items, informative or instructional text, and controls such as buttons and checkboxes. You can create modal, movable modal, or modeless dialog boxes. Figure 1-5 shows an example of each type of dialog box.

A modal dialog box is a dialog box that puts the user in the state or "mode" of being able to work only inside the dialog box. A modal dialog box is similar in appearance to an alert box, except that a modal dialog box can contain editable text items and additional

Figure 1-5 Modal, movable modal, and modeless dialog boxes



controls, such as radio buttons and pop-up menus. The user cannot move a modal dialog box, and the user can dismiss a modal dialog box only by clicking its buttons. You should use a modal dialog box only when it's essential for the user to complete an operation before performing any other work.

A movable modal dialog box is a modal dialog box with a title bar (but no close box) that allows the user to move the dialog box. The user can dismiss the dialog box only by clicking its buttons; however, when you use movable modal dialog boxes, you should allow the user to switch to another application if the user clicks in the window of another application or chooses another application from the Apple or Application menu. Use a movable modal dialog box when the user might need to move the dialog box to view other areas of the screen or when the user can switch to another application without affecting the state of your application.

A modeless dialog box is a dialog box that looks like a document window without a size box or scroll bars. A modeless dialog box does not require the user to respond before doing anything else. The user can move a modeless dialog box, move between a modeless dialog box and other windows, and close a modeless dialog box just like a document window. Whenever possible, use a modeless dialog box instead of a movable modal or modal dialog box. Use a modeless dialog box when the user can perform other operations—such as working in document windows—without dismissing the modeless dialog box.

The chapter "Dialog Manager" in this book describes in detail how you can create alert boxes and dialog boxes for your application.

Icons and Other Interactions With the Finder

Once you've designed your application, you need to create icons to represent the application and the documents it creates. The Finder displays these icons to the user. If your application appears as an item in the Apple or Application menu, the Menu Manager displays your application's icon next to its name, and the Menu Manager displays your application's icon as the title of the Application menu when your application is the active application.

The chapter "Finder Interface" in this book describes how to define and create the icons for your application and its documents. The chapter also describes how your application interacts with the Finder.

When a user opens your application or opens or prints one of its documents, the Finder uses the Process Manager to schedule your application for execution and then sets up the information your application needs to determine which, if any, files to open or print. In System 7, your application can choose to receive this information through Apple events. By supporting these and other Apple events, your application can efficiently respond to requests from the user as well as requests from other applications. See *Inside Macintosh: Interapplication Communication* for information about supporting Apple events.

Resources

Resources are basic elements of every Macintosh application. By defining descriptions of menus, windows, controls, dialog boxes, sounds, fonts, and icons in resources, you can make these and other elements easier to create and manage. Using resources also eases translation of user interface elements into other languages.

A **resource** is any data stored according to a defined structure in the resource fork of a file; the data in a resource is interpreted according to its resource type. You usually create resources using a resource compiler or resource editor. This book shows resources in Rez format; Rez is a resource compiler provided with the Macintosh Programmer's Workshop (available from APDA). You can also use other resource tools, such as ResEdit (also available from APDA), to create the resources for your application.

Most of the managers described in this book use the Resource Manager to read resources for you. For example, you can use the Menu Manager, Window Manager, Dialog Manager, and Control Manager to read descriptions of your application's menus, windows, dialog boxes, and controls from resources. These managers all interpret a resource's data accordingly once it is read into memory. While you'll typically use these managers to access resources for you, you can also directly use the Resource Manager to read and write resources.

The chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox* describes the Resource Manager in detail. However, to help you understand how the Menu Manager, Window Manager, Dialog Manager, and Control Manager use resources, this section gives a brief overview of resources and provides a general introduction to the Resource Manager.

Macintosh system software treats a **file** as a named, ordered sequence of bytes stored on a Macintosh volume and divided into two forks, the data fork and the resource fork. The **data fork** contains data that usually corresponds to data created by the user; the application creating the file can store and interpret the data in the data fork in whatever manner is appropriate. The **resource fork** of a file consists of a resource map and the resources themselves.

When you write data to a file, you write to either the file's resource fork or its data fork. You typically read from and write to a file's data fork using File Manager routines and read from and write to a file's resource fork using Resource Manager routines.

You typically store as resources data that has a defined structure—such as icons and sounds—and descriptions of menus, controls, dialog boxes, and windows. When you create a resource, you assign it a resource type and resource ID. A **resource type** is a sequence of four characters that uniquely identifies a specific type of resource, and a **resource ID** identifies by number a specific resource within that type. (You can also use a resource name in place of a resource ID to identify a particular resource within a resource type.) For example, to create a description of a menu in a resource, you create a resource of type `'MENU'` and give it a resource ID or resource name that is unique from any other `'MENU'` resources that you have defined. Some resources have restrictions on the numbers you can use for resource IDs; in general, numbers 128 through 32767 are available for your use.

System software defines a number of standard resource types, such as 'ALRT', 'CNTL', 'CODE', 'DITL', 'DLOG', 'FONT', 'ICN#', 'ICON', 'MBAR', 'MENU', 'STR ', 'STR#', and 'WIND'. You can use these resource types to define their corresponding elements (for example, use a 'WIND' resource to define a window). You can also create your own resource types if your application needs resources other than the standard resource types defined by the system software.

The Resource Manager does not interpret the format of an individual resource type. When you request a resource of a particular type with a given resource ID, the Resource Manager looks for the specified resource and, if it finds it, reads the resource into memory and returns a handle to it. Your application or other system software routines can use the Resource Manager to read resources into memory. For example, when you use the Window Manager to read a description of a window from a 'WIND' resource, the Window Manager uses the Resource Manager to read the resource into memory. Once the resource is in memory, the Window Manager interprets the resource's data and creates a window with the characteristics described by the resource.

System software stores certain resources used by the system software in the System file. Although many of these resources are used only by the system software, your application can access some of these resources if needed. For example, the standard images for the I-beam and wristwatch cursors are stored as resources of type 'CURS' in the System file. You can use these resources to change the appearance of the cursor used by your application.

Occasionally you may need to write resources to the resource fork of a file. For example, if your application saves the last position and size of a window (as determined by the user), you can store this information in the resource fork of the document in a resource defined by your application. The next time the user opens the document, your application can read the location saved in this resource and position the document accordingly.

You typically store the resources specific to your application, such as descriptions of its menus, windows, controls, and dialog boxes, in the resource fork of your application. You can store resources specific to a document created by your application in the resource fork of the document file.

The resource map in the resource fork of a file contains entries that provide the location of each resource in the resource fork. When the Resource Manager opens the resource fork of a file, it reads the resource map into memory. As the Resource Manager reads resources into memory, it replaces their entries in the resource map with handles to their data in memory. The Resource Manager always searches the resource map in memory, not the resource map of the resource fork on disk, when it searches for a resource. If a requested resource is in memory, the Resource Manager uses the resource in memory; otherwise it reads the resource from the resource fork on disk into memory.

Once the Resource Manager has opened a resource fork and read its resource map into memory, it keeps the map in memory until the file is closed. You can specify that a resource be read into memory immediately when the Resource Manager opens a file's resource fork, or you can specify that the Resource Manager read it into memory only when needed. The Resource Manager stores resources from resource forks opened by

your application in relocatable blocks in your application's heap. You can also specify whether the Resource Manager should purge a resource from memory in order to make room in memory for other data. If you specify that a resource is purgeable, you need to use the Resource Manager to make sure the resource is in memory before accessing it through its resource handle.

When a user opens your application, system software opens your application's resource fork. When your application opens a file, your application typically opens both the file's data fork and the file's resource fork. When your application requests a resource from the Resource Manager, the Resource Manager follows a specific search order. (If necessary, your application can change the search order using Resource Manager routines.) The Resource Manager normally looks first for the resource in the resource fork of the last file that your application opened. So, if your application has a single file open, the Resource Manager looks first in that file's resource fork. If the Resource Manager doesn't find the resource there, it continues to search each resource fork open to your application in the reverse order that the files were opened. After looking in the resource forks of files your application has opened, the Resource Manager searches your application's resource fork. If it doesn't find the resource there, it searches the resource fork of the System file.

This search path allows your application to use resources defined in the System file, to override resources defined in the System file, to share resources between files by using resources stored in your application's resource fork, and to override your application-defined resources and use resources specific to a document.

A Macintosh file always contains both a resource fork and a data fork, although one or both of those forks can be empty. Document files typically contain the document's data in the data fork and any document-specific resources—such as preference settings, window location, and the document icon—in the resource fork. The resource fork of an application typically includes resources that describe the application's menus, windows, controls, dialog boxes, and icons, as well as the code itself, which is also stored as a resource.

Whether you store data in the data fork or the resource fork of a document file depends largely on whether you can structure that data in a useful manner as a resource. For example, it's often convenient to store document-specific settings, such as the document's previous window size and location, as a resource in the document's resource fork. Data that is likely to be edited by the user is usually stored in the data fork of a document.

A resource fork can contain at most 2700 resources. The Resource Manager uses a linear search when searching a resource fork's resource types and resource IDs. In general, you should not create more than 500 resources of the same type in any one resource fork.

Inside Macintosh: More Macintosh Toolbox describes resources and the use of the Resource Manager in more detail. For information on writing data to a file's data fork, see *Inside Macintosh: Files*.

Help Balloons

Your application can provide help balloons for elements such as menus, dialog boxes, or the content area of a window. The chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox* describes how your application can provide help balloons. You can also create help balloons for some elements of your application's interface—such as its menus—using the application BalloonWriter, which is available from APDA.

Copy and Paste

All Macintosh applications should support the copying of data from and pasting of data to the Clipboard. The chapter "Scrap Manager" in *Inside Macintosh: More Macintosh Toolbox* describes how to copy data from the Clipboard and paste data to the Clipboard by using the Scrap Manager.

Related System Software Features

In addition to the managers provided by the Macintosh Toolbox, you can also use other managers and system software routines. For example, you can use QuickDraw routines to draw the content of your application's windows, TextEdit (in conjunction with the Dialog Manager) to handle editable text items in dialog boxes, the File Manager to read and write files, the Process Manager and Memory Manager to control various aspects of your application's execution, and the Edition Manager and Apple Event Manager to support interapplication communication. The rest of this chapter describes some of these managers and where you can find more information about them.

Drawing on the Screen

System software routines, such as the routines provided by QuickDraw, perform all drawing on the screen. For example, your application tells QuickDraw what and where to draw, and QuickDraw does the actual drawing to the screen. The graphics routines provided by system software support quick drawing of objects such as circles, ovals, rectangles, lines, text, and pictures. See *Inside Macintosh: Imaging* for specific graphics-related information.

Handling Text

You can use the system software routines provided by TextEdit to greatly simplify basic text editing and formatting that your application would otherwise have to implement. For example, most applications use editable text items in dialog boxes; your application can use the Dialog Manager (which calls TextEdit) to automatically handle user interaction in editable text items. The Dialog Manager and your application can use

TextEdit to insert new text, delete characters that the user backspaces over, scroll text within a window, cut text, copy text, paste text, select text, and handle word wrapping. Most applications use TextEdit only for simple text editing in a dialog box and use their own techniques for handling editable text in document windows.

You should design your application so that it can handle text in more than one language or script. System software provides many routines to help you accomplish this. For example, if your application automatically displays the date in the footer of your document, you can use Text Utility routines to automatically display the date in the format common to the current script. Similarly, if your application provides a Find command, it can use Text Utility routines to search according to the word-break tables and according to the current script.

See *Inside Macintosh: Text* for information on how you can provide support for text editing in documents created by your application and for information on designing your application so that it can support text editing in more than one language or script.

Managing Files

When the user chooses the Save or Save As menu command, you usually write to a file the data that the user has entered in the active window. When the user selects a file using the Open command, you read information from a file. You can use the File Manager to read and write files. You can use the system software routines provided by the Standard File Package to present a standard and consistent interface to the user when saving and opening files. See the chapters "Introduction to File Management" and "Standard File Package" in *Inside Macintosh: Files* for information about these topics.

Allocating Memory and Launching Processes

For information about how the Process Manager launches your application, see the chapter "Process Manager" in *Inside Macintosh: Processes*. See the chapter "Introduction to Memory Management" in *Inside Macintosh: Memory* for information about how system software manages memory; how you can manage the memory in your application's partition effectively; and how your application can allocate, release, or manipulate memory.

Creating Publishers and Subscribers

Your application should support Edition Manager features so that users can share and automatically update data between documents. See the chapter "Edition Manager" in *Inside Macintosh: Interapplication Communication* for information about supporting publish and subscribe features.

Communicating With Other Applications

System software provides various means of communication between applications. You can use Event Manager routines to communicate, in the form of high-level events, with other applications. High-level events are not required to adhere to any specific protocol, so their interpretation is defined by each application that sends or receives them. Apple events are high-level events that follow a standard defined protocol (the Apple Event Interprocess Messaging Protocol). In most cases, you should use Apple events for communication between applications. Because Apple has defined a standard set of Apple events, all applications can interpret specific Apple events in the same way and respond in an expected manner.

Both the Event Manager and Apple Event Manager rely on the services of the Program-to-Program Communications (PPC) Toolbox to actually send and receive events between applications. Your application can also directly access the PPC Toolbox if you need to get additional control or services not provided by the Event Manager or Apple Event Manager.

If your application supports publish and subscribe features, the Edition Manager sends your application Apple events to notify it when new data is available for a subscriber or to request that it create a new publisher.

For information on Apple events, publish and subscribe features, or direct access to the PPC Toolbox, see *Inside Macintosh: Interapplication Communication*.

Designing Your Application

As previously described, you'll need to make extensive use of this book and *Macintosh Human Interface Guidelines* as you begin to design your application. Once you implement the basic elements of a Macintosh application, you can begin to add features unique to your application. Once again, you'll find *Macintosh Human Interface Guidelines* and other books in the *Inside Macintosh* library valuable tools as you create applications.