EARLY MACINTOSH TECHNICAL INFORMATION

INSIDE MACINTOSH
MACINTOSH USER INTERFACE GUIDELINES

COMMENT
DISCUSSION FOR PROGRAMMERS OF HOW TO DESIGN THE USER INTERFACE FOR THEIR MACINTOSH PROGRAMS, VERY INNOVATIVE GUIDELINES FOR THE PERSONAL COMPUTER INDUSTRY

AUTHOR
APPLE COMPUTER

DATE
30 NOVEMBER 1984

SOURCE
DAVID T CRAIG • JANUARY 2004

MACINTOSH USER EDUCATION

Macintosh User Interface Guidelines /INTF/USER

Modification History:			
	First Draft	Joanna Hoffman	3/17/82
	Rearranged and Revised	Chris Espinosa	5/11/82
	Total Redesign	Chris Espinosa	5/21/82
	Second Draft Prerelease	Chris Espinosa	7/11/82
	Second Draft	Chris Espinosa	1Ø/11/82
	Third Draft	Andy Averill	7/31/84
	Fourth Draft	Andy Averill	11/3Ø/84

ABSTRACT

The User Interface Guidelines describe the most basic common features of Macintosh applications. Unlike the rest of Inside Macintosh, these guidelines describe these features as seen by the user.

Since the last draft, this manual has been reorganized and mostly rewritten. Some new recommendations have been added, and some previous recommendations have been clarified or amplified.

2 User Interface Guidelines

TABLE OF CONTENTS

4 User Interface Guidelines

ABOUT THIS MANUAL

This manual describes the Macintosh user interface, for the benefit of people who want to develop Macintosh applications. *** Eventually it will become part of the comprehensive Inside Macintosh manual. *** More details about many of these features can be found in the "About" sections of the other chapters of Inside Macintosh.

Unlike the rest of Inside Macintosh, this manual describes applications from the outside, not the inside. The terminology used is the terminology users are familiar with, which is not necessarily the same as that used elsewhere in Inside Macintosh.

The Macintosh user interface consists of those features that are generally applicable to a variety of applications. Not all of the features are found in every application. In fact, some features are hypothetical, and may not be found in any current applications.

The best time to familiarize yourself with the user interface is before beginning to design an application. Good application design on the Macintosh happens when a developer has absorbed the spirit as well as the details of the user interface.

Before reading this manual, you should have read Inside Macintosh: A Road Map and have some experience using one or more applications, preferably one each of a word processor, spreadsheet or data base, and graphics application. You should also have read Macintosh, the owner's guide, or at least be familiar with the terminology used in that manual.

INTRODUCTION

The Macintosh is designed to appeal to an audience of nonprogrammers, including people who have previously feared and distrusted computers. To achieve this goal, Macintosh applications should be easy to learn and to use. To help people feel more comfortable with the applications, the applications should build on skills that people already have, not force them to learn new ones. The user should feel in control of the computer, not the other way around. This is achieved in applications that embody three qualities: responsiveness, permissiveness, and consistency.

Responsiveness means that the user's actions tend to have direct results. The user should be able to accomplish what needs to be done spontaneously and intuitively, rather than having to think: "Let's see; to do C, first I have to do A and B and then...". For example, with pull-down menus, the user can choose the desired command directly and instantaneously. This is a typical Macintosh operation: The user moves the pointer to a location on the screen and presses the mouse button.

Permissiveness means that the application tends to allow the user to do anything reasonable. The user, not the system, decides what to do next. Also, error messages tend to come up infrequently. If the user is constantly subjected to a barrage of error messages, something is wrong somewhere.

The most important way in which an application is permissive is in avoiding modes. This idea is so important that it's dealt with in a separate section, "Avoiding Modes", below.

The third and most important principle is consistency. Since Macintosh users usually divide their time among several applications, they would be confused and irritated if they had to learn a completely new interface for each application. The main purpose of this manual is to describe the shared interface ideas of Macintosh applications, so that developers of new applications can gain leverage from the time spent developing and testing existing applications.

Fortunately, consistency is easier to achieve on the Macintosh than on many other computers. This is because many of the routines used to implement the user interface are supplied in the Macintosh Operating System and User Interface Toolbox. However, you should be aware that implementing the user interface guidelines in their full glory often requires writing additional code that isn't supplied.

Of course, you shouldn't feel that you're restricted to using existing features. The Macintosh is a growing system, and new ideas are essential. But the bread-and-butter features, the kind that every application has, should certainly work the same way so that the user can move easily back and forth between applications. The best rule to follow is that if your application has a feature that's described in these guidelines, you should implement the feature exactly as the guidelines describe it. It's better to do something completely different than to half-agree with the guidelines.

Illustrations of most of the features described in this manual can be found in various already-released applications. However, there is probably no one application that illustrates these guidelines in every particular. Although it's useful and important for you to get the feeling of the Macintosh user interface by looking at existing applications, the guidelines in this manual are the ultimate authority. Wherever an existing application disagrees with the guidelines, follow the guidelines.

Avoiding Modes

> "But, gentlemen, you overdo the mode."
>
> -- John Dryden, *The Assignation, or Love in a Nunnery*, 1672

6 User Interface Guidelines

A mode is a part of an application that the user has to formally enter and leave, and that restricts the operations that can be performed while it's in effect. Since people don't usually operate modally in real life, having to deal with modes in computer software reinforces the idea that computers are unnatural and unfriendly.

Modes are most confusing when you're in the wrong one. Unfortunately, this is the most common case. Being in a mode is confusing because it makes future actions contingent upon past ones; it changes the behavior of familiar objects and commands; and it makes habitual actions cause unexpected results.

It's tempting to use modes in a Macintosh application, since most existing software leans on them heavily. If you yield to the temptation too frequently, however, users will consider spending time with your application a chore rather than a satisfying experience.

This is not to say that modes are never used in Macintosh applications. Sometimes a mode is the best way out of a particular problem. Most of these modes fall into one of the following categories:

- Long-term modes with a procedural basis, such as doing word processing as opposed to graphics editing. Each application program is a mode in this sense.

- Short-term "spring-loaded" modes, in which the user is constantly doing something to perpetuate the mode. Holding down the mouse button or a key is the most common example of this kind of mode.

- Alert modes, where the user must rectify an unusual situation before proceeding. These modes should be kept to a minimum.

Other modes are acceptable if they meet one of the following requirements:

- They emulate a familiar real-life model that is itself modal, like picking up different-sized paintbrushes in a graphics editor. MacPaint and other palette-based applications are examples of this use of modes.

- They change only the attributes of something, and not its behavior, like the boldface and underline modes of text entry.

- They block most other normal operations of the system to emphasize the modality, as in error conditions incurable through software ("There's no disk in the disk drive", for example).

If an application uses modes, there must be a clear visual indication of the current mode, and the indication should be near the object being most affected by the mode. It should also be very easy to get into or out of the mode (such as by clicking on a palette symbol).

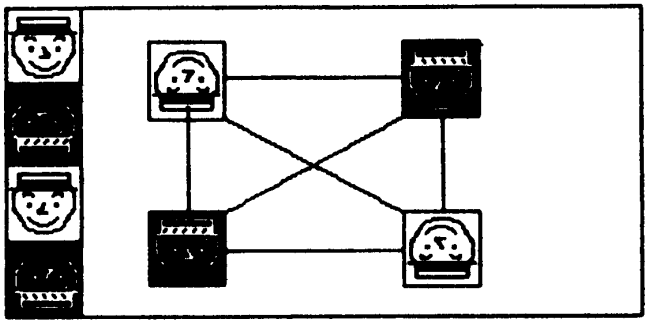
TYPES OF APPLICATIONS

Everything on a Macintosh screen is displayed graphically; the Macintosh has no text mode. Nevertheless, it's useful to make a distinction among three types of objects that an application deals with: text, graphics, and arrays. Examples of each of these are shown in Figure 1.

The rest to some faint meaning make pretence
But Shadwell never deviates into sense.
Some beams of wit on other souls may fall,
Strike through and make a lucid interval;
But Shadwell's genuine night admits no ray,
His rising fogs prevail upon the day.

MacFlecknoe Page 1

Text



Graphics

Advertising	132.9	
Manufacturing	121.3	
R & D	18.7	
Interest	12.2	
Total	285.1	

Array

Figure 1. Ways of Structuring Information

Text can be arranged in a variety of ways on the screen. Some applications, such as word processors, might consist of nothing but text, while others, such as graphics-oriented applications, use text almost incidentally. It's useful to consider all the text appearing together in a particular context as a block of text. The size of the block can range from a single field, as in a dialog box, to the whole document, as in a word processor. Regardless of its size or arrangement, the application sees each block as a one-dimensional string of characters. Text is edited the same way regardless of where it appears.

Graphics are pictures, drawn either by the user or by the application. Graphics in a document tend to consist of discrete objects, which can be selected individually. Graphics are discussed further below, under "Using Graphics".

Arrays are one- or two-dimensional arrangements of fields. If the array is one-dimensional, it's called a form; if it's two-dimensional it's called a table. Each field, in turn, contains a collection of information, usually text, but conceivably graphics. A table can be readily identified on the screen, since it consists of rows and columns of fields (often called cells), separated by horizontal and vertical lines. A form is something you fill out, like a credit-card application. A form may not be as obvious to the user as a table, since the fields can be arranged in any appropriate way. Nevertheless, the application regards the fields as in a definite linear order.

Each of these three ways of presenting information retains its integrity, regardless of the context in which it appears. For example, a field in an array can contain text. When the user is manipulating the field as a whole, the field is treated as part of the array. When the user wants to change the contents of the field, the contents are edited in the same way as any other text.

Another case is text that appears in a graphics application. Depending on the circumstances, the text can be treated as text or as graphics. In MacDraw, for example, the way text is treated depends on which palette symbol is in effect. If the text symbol is in effect, text can be edited in the usual way, but cannot be moved around on the screen. If the selecting arrow is in effect, a block of text can be moved around, or even stretched or shrunk, but cannot be edited.

USING GRAPHICS

A key feature of the Macintosh is its high-resolution graphics screen. To use this screen to its best advantage, Macintosh applications use graphics copiously, even in places where other applications use text. As much as possible, all commands, features, and parameters of an application, and all the user's data, appear as graphic objects on the screen. Figure 2 shows some of the ways in which applications can use graphics to communicate with the user.

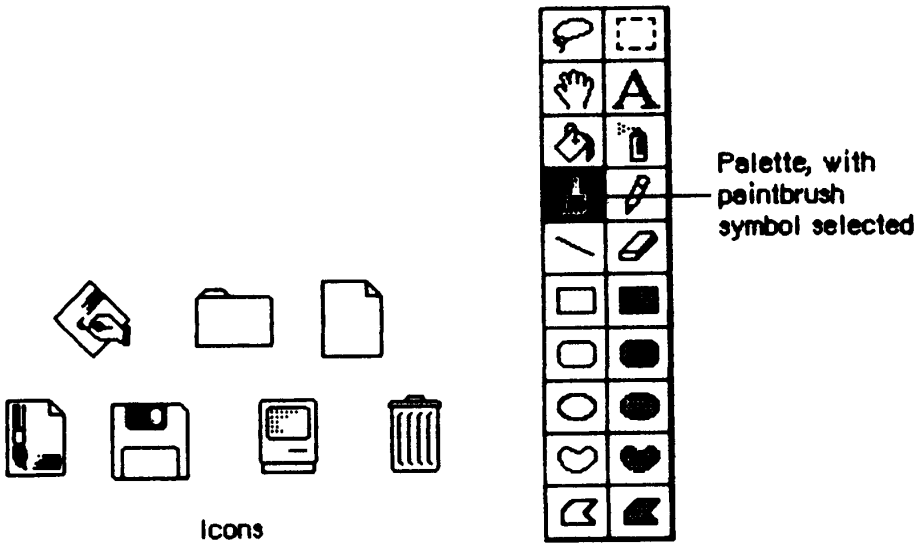


Figure 2. Objects on the Screen

Objects, whenever applicable, resemble the familiar material objects they resemble. Objects that act like pushbuttons "light up" when pressed; the Trash icon looks like a trash can.

Objects are designed to look good on the screen. Predefined graphics patterns can give objects a shape and texture beyond simple line graphics. Placing a drop-shadow slightly below and to the right of an object can give it a three-dimensional appearance.

Generally, when the user clicks on an object, it's highlighted to distinguish it from its peers. The most common way to show this highlighting is by inverting the object: reversing its black and white pixels. In some situations, other forms of highlighting, such as the knobs used in MacDraw, may be more appropriate. The important thing is that there should always be some sort of feedback, so that the user knows that the click had an effect.

One special aspect of the appearance of a document on the screen is visual fidelity. This principle is also known as "what you see is what you get". It primarily refers to printing: The version of a document shown on the screen should be as close as possible to its printed version, taking into account inevitable differences due to different media.

Icons

A fundamental object in Macintosh software is the icon, a small graphic object that is usually symbolic of an operation or of a larger entity such as a document.

Icons should be sprinkled liberally over the screen. Wherever an explanation or label is needed, first consider using an icon instead of using text as the label or explanation. Icons not only contribute to the clarity and attractiveness of the system, they don't need to be translated into foreign languages.

Palettes

Some applications use palettes as a quick way for the user to change from one operation to another. A palette is a collection of small squares, each containing a symbol. A symbol can be an icon, a pattern, a character, or just a drawing, that stands for an operation. When the user clicks on one of the symbols, it's distinguished from the other symbols, such as by highlighting, and the previous symbol goes back to its normal state.

Typically, the symbol that's selected determines what operations the user can perform. Selecting a palette symbol puts the user into a mode. This use of modes can be justified because changing from one mode to another is almost instantaneous, and the use can always see at a glance which mode is in effect. Like all modal features, palettes should be used only when they're the most natural way to structure an application.

A palette can either be part of a window (as in MacDraw), or a separate window (as in MacPaint). Each system has its disadvantages. If the palette is part of the window, then parts of the palette might be concealed if the user makes the window smaller. On the other hand, if it's not part of the window, then it takes up extra space on the desktop. If an application supports multiple documents open at the same time, it might be better to put a separate palette in each window, so that a different palette symbol can be in effect in each document.

COMPONENTS OF THE MACINTOSH SYSTEM

This section explains the relationship among the principal large-scale components of the Macintosh system (from an external point of view).

The main vehicle for the interaction of the user and the system is the application. Only one application is active at a time. When an application is active, it's in control of all communications between the user and the system. The application's menus are in the menu bar, and the application is in charge of all windows as well as the desktop.

To the user, the main unit of information is the document. Each document is a unified collection of information--a single business letter or spreadsheet or chart. A complex application, such as a data base, might require several related documents. Some documents can be processed by more than one application, but each document has a principal application, which is usually the one that created it. The other applications that process the document are called secondary applications.

The only way the user can actually see the document (except by printing it) is through a window. The application puts one or more windows on the screen; each window shows a view of a document or of auxiliary information used in processing the document. The part of the screen underlying all the windows is called the desktop.

The user returns to the Finder to change applications. When the Finder is active, if the user double-clicks on either the application's icon or the icon of a document belonging to that application (or opens the document or application by choosing Open from the File menu), the application becomes active and displays the document window.

Internally, applications and documents are both kept in files. However, the user never sees files as such, so they don't really enter into the user interface.

THE KEYBOARD

The Macintosh keyboard is used primarily for entering text. Since commands are chosen from menus or by clicking somewhere on the screen, the keyboard is not needed for this function, although it can be used for alternative ways to enter commands.

The keys on the keyboard are arranged in familiar typewriter fashion. The U.S. keyboard is shown in Figure 3.

Figure 3. The Macintosh U.S. Keyboard

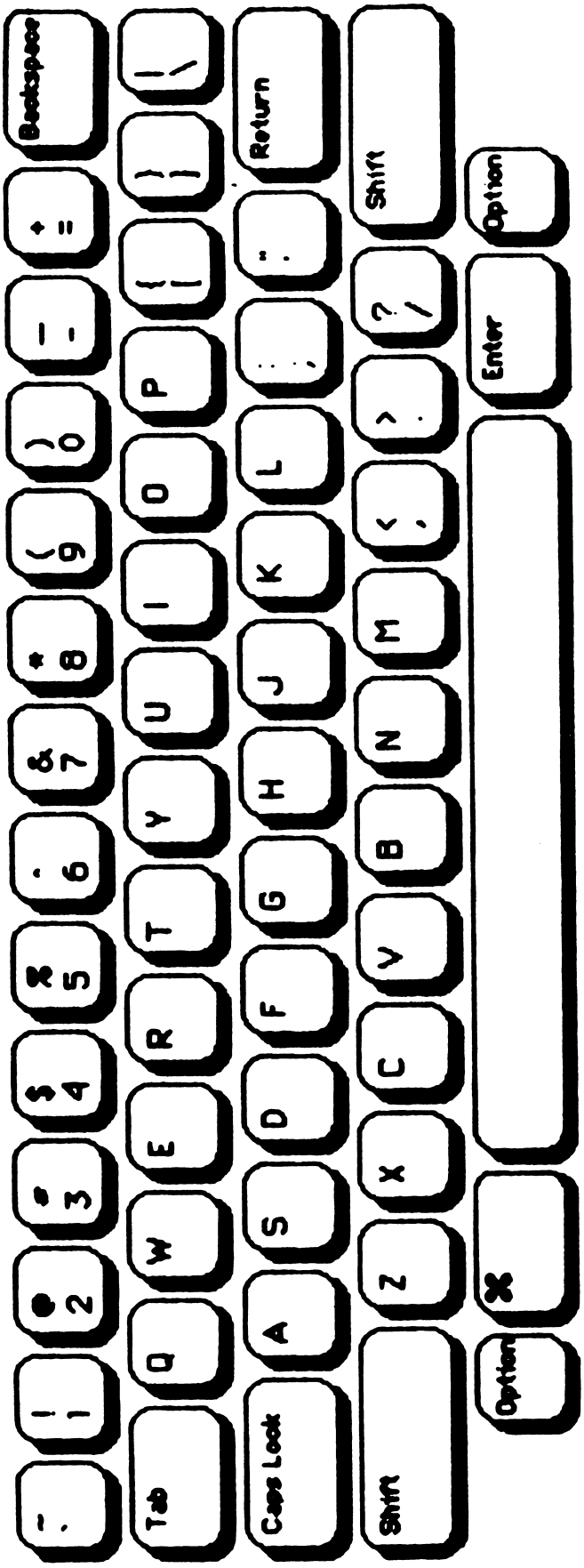


Figure 3. The Macintosh U.S. Keyboard

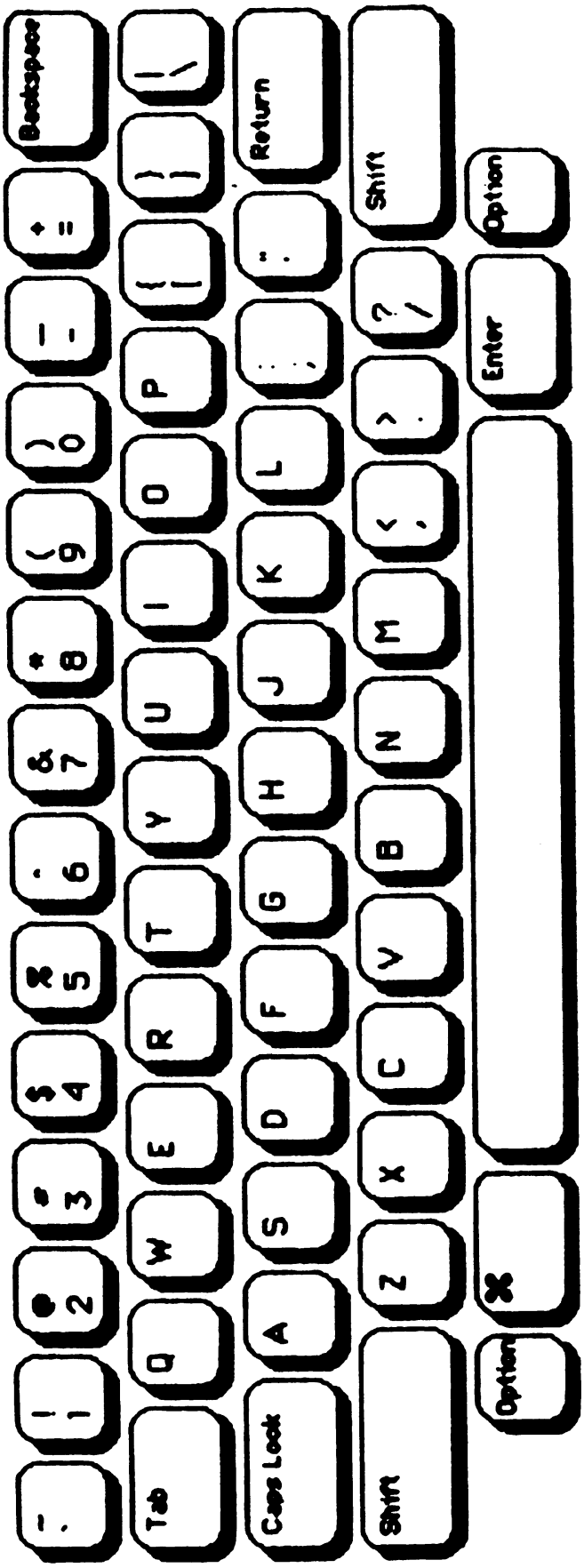


Figure 3. The Macintosh U.S. Keyboard

There are two kinds of keys: character keys and modifier keys. A character key sends characters to the computer; a modifier key alters the meaning of a character key if it's held down while the character key is pressed.

Character Keys

Character keys include keys for letters, numbers, and symbols, as well as the space bar. If the user presses one of these keys while entering text, the corresponding character is added to the text. Other keys, such as the Enter, Tab, Return, Backspace, and Clear keys, are also considered character keys. However, the result of pressing one of these keys depends on the application and the context.

The Enter key tells the application that the user is through entering information in a particular area of the document, such as a field in an array. Most applications add information to a document as soon as the user types or draws it. However, the application may need to wait until a whole collection of information is available before processing it. In this case, the user presses the Enter key to signal that the information is complete.

The Tab key is a signal to proceed: It signals movement to the next item in a sequence. Tab often implies an Enter operation before the Tab motion is performed.

The Return key is another signal to proceed, but it defines a different type of motion than Tab. A press of the Return key signals movement to the leftmost field one step down (just like a carriage return on a typewriter). Return can also imply an Enter operation before the Return operation.

(note)
Return and Enter also dismiss dialog and alert boxes (see "Dialogs and Alerts").

Backspace is used to delete text or graphics. The exact use of Backspace in text is described in the section on text editing.

The Clear key on the keypad has the same effect as the Clear command in the Edit menu; that is, it removes the selection from the document without putting it on the Clipboard. This is also explained in the section on text editing. Because the keypad is optional equipment, no application should ever require use of the Clear key or any other key on the pad.

Modifier Keys: Shift, Caps Lock, Option, and Command

There are six keys on the keyboard that change the interpretation of keystrokes: two labeled Shift, two labeled Option, one labeled Caps Lock, and one labeled with the "freeway interchange" symbol, which is usually called the Command key. These keys change the

interpretation of keystrokes, and sometimes mouse actions. When one of these keys is held down, the effect of the other keys (or the mouse button) may change.

The Shift and Option keys choose among the characters on each character key. Shift gives the upper character on two-character keys, or the uppercase letter on alphabetic keys. The Shift key is also used in conjunction with the mouse for extending a selection; see "Selecting". Option gives an alternate character set interpretation, including international characters, special symbols, and so on. Shift and Option can be used in combination.

Caps Lock latches in the down position when pressed, and releases when pressed again. When down it gives the uppercase letter on alphabetic keys. The operation of Caps Lock on alphabetic keys is parallel to that of the Shift key, but the Caps Lock key has no effect whatsoever on any of the other keys. Caps Lock and Option can be used in combination on alphabetic keys.

Pressing a character key while holding down the Command key usually tells the application to interpret the key as a command, not as a character (see "Commands").

Typeahead and Auto-Repeat

If the user types when the Macintosh is unable to process the keystrokes immediately, or types more quickly than the Macintosh can handle, the extra keystrokes are queued, to be processed later. This queuing is called typeahead. There's a limit to the number of keystrokes that can be queued, but the limit is usually not a problem unless the user types while the application is performing a lengthy operation.

When the user holds down a character key for a certain amount of time, it starts repeating automatically. The delays and the rates of repetition are global preferences that the user can set through the Control Panel desk accessory. An application can tell whether a series of n keystrokes was generated by auto-repeat or by pressing the same key n times. It can choose to disregard keystrokes generated by auto-repeat; this is usually a good idea for menu commands chosen with the Command key.

Holding down a modifier key has the same effect as pressing it once. However, if the user holds down a modifier key and a character key at the same time, the effect is the same as if the user held down the modifier key while pressing the character key repeatedly.

Auto-repeat does not function during typeahead; it operates only when the application is ready to accept keyboard input.

Versions of the Keyboard

There are two physical versions of the keyboard: U.S. and European. The European version has one more key than the U.S. version. The standard layout on the European version is designed to conform to the ISO (Internation Standards Organization) standard; the U.S. key layout mimics that of common American office typewriters. European keyboards have different labels on the keys in different countries, but the overall layout is the same.

The Numeric Keypad

An optional numeric keypad can be hooked up between the main unit and the standard keyboard; see Figure 4.

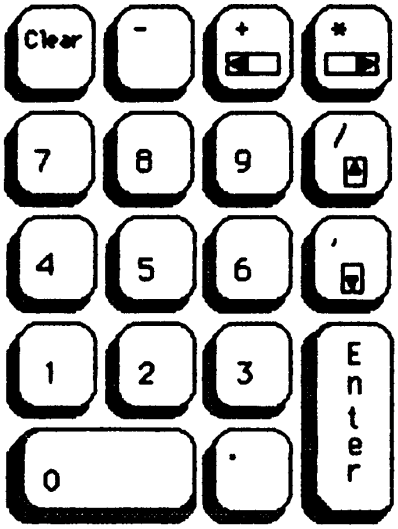


Figure 4. Numeric Keypad

The keypad contains 18 keys, some of which duplicate keys on the main keyboard, and some of which are unique to the keypad. The application can tell whether the keystrokes have come from the main keyboard or the numeric keypad.

The character keys on the keypad are labeled with the digits Ø through 9, a decimal point, the four standard arithmetic operators for addition, subtraction, multiplication, and division, and a comma. The keypad also contains the Enter and Clear keys; it has no modifier keys.

The keys on the numeric keypad follow the same rules for typeahead and auto-repeat as the main keyboard.

Four keys on the numeric keypad are labeled with "field-motion" symbols: small rectangles with arrows pointing in various directions.

Some applications may use these keys to select objects in the direction indicated by the key; the most likely use for this feature is in tables. When a key is used this way, the user must use the Shift key to obtain the four characters (+ * / ,) normally available on those keys.

Since the numeric keypad is optional equipment, no application should require it or any keys available on it in order to perform standard functions. Specifically, since the Clear key is not available on the main keyboard, a Clear function may be implemented with this key only as the equivalent of the Clear command in the Edit menu.

THE MOUSE

The mouse is a small device the size of a deck of playing cards, connected to the computer by a long, flexible cable. There's a button on the top of the mouse. The user holds the mouse and rolls it on a flat, smooth surface. A pointer on the screen follows the motion of the mouse.

Simply moving the mouse results only in a corresponding movement of the pointer and no other action. Most actions take place when the user positions the "hot spot" of the pointer over an object on the screen and presses and releases the mouse button. The hot spot should be intuitive, like the point of an arrow or the center of a crossbar.

Mouse Actions

The three basic mouse actions are:

- clicking: positioning the pointer with the mouse, and briefly pressing and releasing the mouse button without moving the mouse
- pressing: positioning the pointer with the mouse, and holding down the mouse button without moving the mouse
- dragging: positioning the pointer with the mouse, holding down the mouse button, moving the mouse to a new position, and releasing the button

The system provides "mouse-ahead"; that is, any mouse actions the user performs when the application isn't ready to process them are saved in a buffer and can be processed at the application's convenience. Alternatively, the application can choose to ignore saved-up mouse actions, but should do so only to protect the user from possibly damaging consequences.

Clicking something with the mouse performs an instantaneous action, such as selecting a location within the user's document or activating an object.

For certain kinds of objects, pressing on the object has the same effect as clicking it repeatedly. For example, clicking a scroll arrow causes a document to scroll one line; pressing on a scroll arrow causes the document to scroll repeatedly until the mouse button is released or the end of the document is reached.

Dragging can have different effects, depending on what's under the pointer when the mouse button is pressed. The uses of dragging include choosing a menu item, selecting a range of objects, moving an object from one place to another, and shrinking or expanding an object.

Some objects, especially graphic objects, can be moved by dragging. In this case, the application attaches a dotted outline of the object to the pointer and redraws the outline continually as the user moves the pointer. When the user releases the mouse button, the application redraws the complete object at the new location.

An object being moved can be restricted to certain boundaries, such as the edges of a window frame. If the user moves the pointer outside of the boundaries, the application stops drawing the dotted outline of the object. If the user releases the mouse button while the pointer is outside of the boundaries, the object isn't moved. If, on the other hand, the user moves the pointer back within the boundaries again before releasing the mouse button, the outline is drawn again.

In general, moving the mouse changes nothing except the location, and possibly the shape, of the pointer. Pressing the mouse button indicates the intention to do something, and releasing the button completes the action. Pressing by itself should have no effect except in well-defined areas, such as scroll arrows, where it has the same effect as repeated clicking.

Multiple-Clicking

A variant of clicking involves performing a second click shortly after the end of an initial click. If the downstroke of the second click follows the upstroke of the first by a short amount of time (as set by the user in the Control Panel), and if the locations of the two clicks are reasonably close together, the two clicks constitute a double-click. Its most common use is as a faster or easier way to perform an action that can also be performed in another way. For example, clicking twice on an icon is a faster way to open it than choosing Open; clicking twice on a word to select it is faster than dragging through it.

To allow the software to distinguish efficiently between single clicks and double-clicks on objects that respond to both, an operation invoked by double-clicking an object must be an enhancement, superset, or extension of the feature invoked by single-clicking that object.

Triple-clicking is also possible; it should similarly represent an extension of a double-click.

Changing Pointer Shapes

The pointer may change shape to give feedback on the range of activities that make sense in a particular area of the screen, in a current mode, or both.

- The result of any mouse action depends on the item under the pointer when the mouse button is pressed. To emphasize the differences among mouse actions, the pointer may assume different appearances in different areas to indicate the actions possible in each area.

- Where an application uses modes for different functions, the pointer can be a different shape in each mode. For example, in MacPaint, the pointer shape always reflects the active palette symbol.

Figure 5 shows some examples of pointers and their effect. An application can design additional pointers for other contexts.

Pointer	Used for
	Scroll bar and other controls, size box title bar, menu bar, desktop, and so on
	Selecting text
	Drawing, shrinking, or stretching graphic objects
	Selecting fields in an array
	Showing that a lengthy operation is in progress

Figure 5. Pointers

SELECTING

The user selects an object to distinguish it from other objects, just before performing an operation on it. Selecting the object of an operation before identifying the operation is a fundamental characteristic of the Macintosh system.

Selecting an object has no effect on the contents of a document. Making a selection shouldn't commit the user to anything; the user is

never penalized for making an incorrect selection. The user fixes an incorrect selection by making the correct selection.

Although there is a variety of ways to select objects, they fall into easily recognizable groups. Users get used to doing specific things to select objects, and applications that use these methods are therefore easier to learn. Some of these methods apply to every type of application, and some only to particular types of applications.

This section discusses first the general methods, and then the specific methods that apply to text applications, graphics applications, and arrays. Figure 6 shows a comparison of some of the general methods.

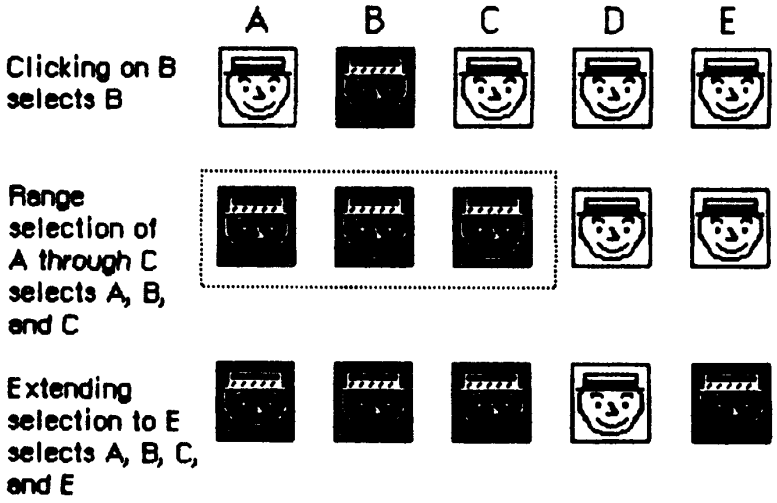


Figure 6. Selection Methods

Selection by Clicking

The most straightforward method of selecting an object is by clicking on it once. Most things that can be selected in Macintosh applications can be selected this way.

Some applications support selection by double-clicking and triple-clicking. As always with multiple clicks, the second click extends the effect of the first click, and the third click extends the effect of the second click. In the case of selection, this means that the second click selects the same sort of thing as the first click, only more of them. The same holds true for the third click.

For example, in text, the first click selects an insertion point, whereas the second click selects a whole word. The third click might select a whole block or paragraph of text. In graphics, the first click selects a single object, and double- and triple-clicks might select increasingly larger groups of objects.

Range Selection

The user selects a range of objects by dragging through them. Although the exact meaning of the selection depends on the type of application, the procedure is always the same:

1. The user positions the pointer at one corner of the range and presses the mouse button. This position is called the anchor point of the range.

2. The user moves the pointer in any direction. As the pointer is moved, visual feedback keeps the user informed of the objects that would be selected if the mouse button were released. For text and arrays, the selected area is continually highlighted. For graphics, a dotted rectangle expands or contracts to show the range that will be selected.

3. When the feedback shows the desired range, the user releases the mouse button. The point at which the button is released is called the endpoint of the range.

Extending a Selection

A user can change the extent of an existing selection by holding down the Shift key and clicking the mouse button. Exactly what happens next depends on the context.

In text or an array, the result of a Shift-click is always a range. The position where the button is clicked becomes the new endpoint or anchor point of the range; the selection can be extended in any direction. If the user clicks within the current range, the new range will be smaller than the old range.

In graphics, a selection is extended by adding objects to it; the added objects do not have to be adjacent to the objects already selected. The user can add either an individual object or a range of objects to the selection by holding down the Shift key before making the additional selection. If the user holds down the Shift key and selects one or more objects that are already highlighted, the objects are deselected.

Extended selections can be made across the panes of a split window. (See "Splitting Windows".)

Making a Discontinuous Selection

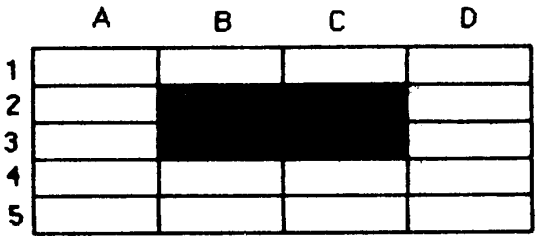
In graphics applications, objects aren't usually considered to be in any particular sequence. Therefore, the user can use Shift-click to extend a selection by a single object, even if that object is nowhere near the current selection. When this happens, the objects between the current selection and the new object are not automatically included in the selection. This kind of selection is called a discontinuous selection. In the case of graphics, all selections are discontinuous selections.

This is not the case with arrays and text, however. In these two kinds of applications, an extended selection made by a Shift-click always includes everything between the old selection and the new endpoint. To provide the possibility of a discontinuous selection in these applications, Command-click is included in the user interface.

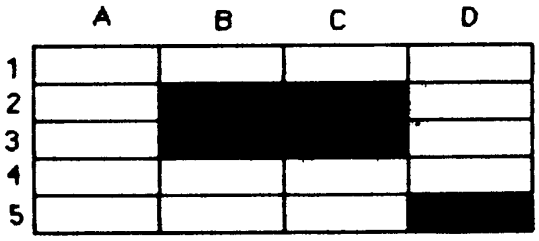
To make a discontinuous selection in a text or array application, the user selects the first piece in the normal way, then holds down the Command key before selecting the remaining pieces. Each piece is selected in the same way as if it were the whole selection, but because the Command key is held down, the new pieces are added to the existing selection instead of supplanting it.

If one of the pieces selected is already within an existing part of the selection, then instead of being added to the selection it's removed from the selection. Figure 7 shows a sequence in which several pieces are selected and deselected.

Cells B2, B3, C2, and C3 are selected



The user holds down the Command key and clicks in D5



The user holds down the Command key and clicks in C3

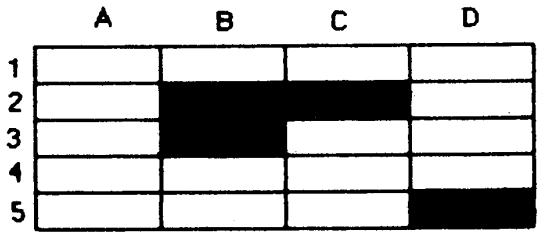


Figure 7. Discontinuous Selection

Not all applications support discontinuous selections, and those that do might restrict the operations that a user can perform on them. For example, a word processor might allow the user to choose a font after making a discontinuous selection, but not to choose Cut or Paste.

Selecting Text

Text is used in most applications; it's selected and edited in a consistent way, regardless of where it appears.

A block of text is a string of characters. A text selection is a substring of this string, which can have any length from zero characters to the whole block. Each of the text selection methods selects a different kind of substring. Figure 8 shows different kinds of text selections.

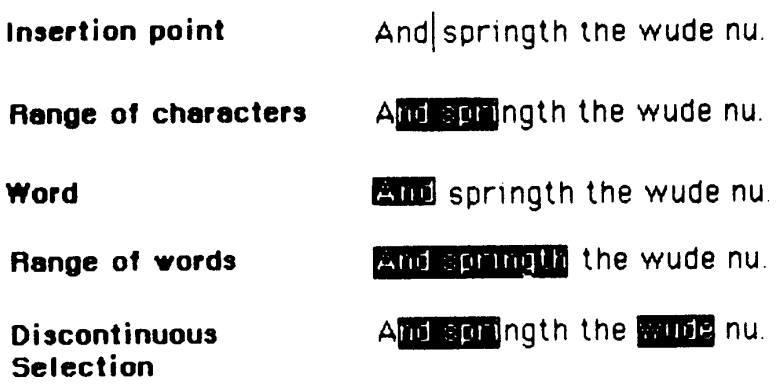


Figure 8. Text Selections

Insertion Point

The insertion point is a zero-length text selection. The user establishes the location of the insertion point by clicking between two characters. The insertion point then appears at the nearest character boundary. If the user clicks to the right of the last character on a line, the insertion point appears immediately after the last character. The converse is true if the user clicks to the left of the first character in the line.

The insertion point shows where text will be inserted when the user begins typing, or where the contents of the Clipboard will be pasted. After each character is typed, the insertion point is relocated to the right of the insertion.

If, between the mouse-down and the mouse-up, the user moves the pointer more than about half the width of a character, the selection is a range selection rather than an insertion point.

Selecting Words

The user selects a whole word by double-clicking somewhere within that word. If the user begins a double-click sequence, but then drags the mouse between the mouse-down and the mouse-up of the second click, the selection becomes a range of words rather than a single word. As the pointer moves, the application highlights or unhighlights a whole word at a time.

A word, or range of words, can also be selected in the same way as any other range; whether this type of selection is treated as a range of characters or as a range of words depends on the operation. For example, in MacWrite, a range of individual characters that happens to coincide with a range of words is treated like characters for purposes

of extending a selection, but is treated like words for purposes of intelligent cut and paste.

A word is defined as any continuous substring that contains only the following characters:

- a letter (including letters with diacritical marks)
- a digit
- a nonbreaking space (Option-space)
- a dollar sign, cent sign, English pound symbol, or yen symbol
- a percent sign
- a comma between digits
- a period before a digit
- an apostrophe between letters or digits
- a hyphen, but not a minus sign (Option-hyphen) or a dash (Option-Shift-hyphen)

This is the definition in the United States and Canada; in other countries, it would have to be changed to reflect local formats for numbers, dates, and currency.

If the user double-clicks over any character not on the list above, only that character is selected.

Examples of words:

```
$123,456.78
shouldn't
3 1/2 [with a nonbreaking space]
.5%
```

Examples of nonwords:

```
7/1Ø/6
 blue cheese [with a breaking space]
 "Yoicks!"  [the quotation
             marks and exclamation point aren't part of the word]
```

Selecting a Range of Text

The user selects a range of text by dragging through the range. A range is either a range of words or a range of individual characters, as described under "Selecting Words", above.

If the user extends the range, the way the range is extended depends on what kind of range it is. If it's a range of individual characters, it can be extended one character at a time. If it's a range of words (including a single word), it's extended only by whole words.

Graphics Selections

There are several different ways to select graphic objects and to show selection feedback in existing Macintosh applications. MacDraw, MacPaint, and the Finder all illustrate different possibilities. This section describes the MacDraw paradigm, which is the most extensible to other kinds of applications.

A MacDraw document is a collection of individual graphic objects. To select one of these objects, the user clicks once on the object, which is then shown with knobs. (The knobs are used to stretch or shrink the object, and won't be discussed in this manual.) Figure 9 shows some examples of selection in MacDraw.

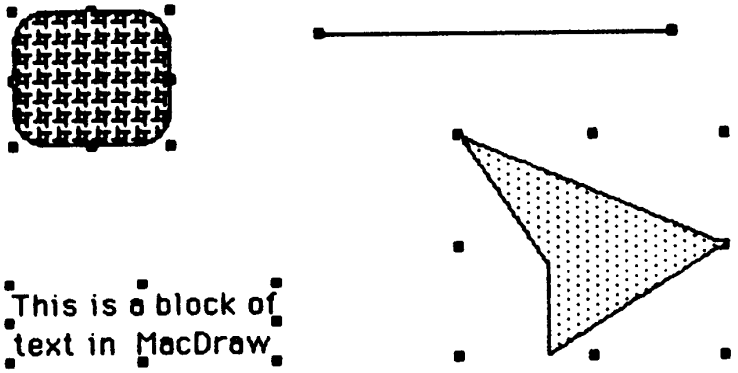


Figure 9. Graphics Selections in MacDraw

To select more than one object, the user can select either a range or a multiple selection. A range selection includes every object completely contained within the dotted rectangle that encloses the range, while an extended selection includes only those objects explicitly selected.

Selections in Arrays

As described above, under "Types of Applications", an array is a one- or two-dimensional arrangement of fields. If the array is one-dimensional, it's called a form; if it's two-dimensional, it's called a table. The user can select one or more fields, or part of the contents of a field.

To select a single field, the user clicks in the field. The user can also implicitly select a field by moving into it with the Tab or Return key.

The Tab key cycles through the fields in an order determined by the application. From each field, the Tab key selects the "next" field. Typically, the sequence of fields is first from left to right, and then from top to bottom. When the last field in a form is selected, pressing the Tab key selects the first field in the form. In a form, an application might prefer to select the fields in logical, rather than physical, order.

The Return key selects the first field in the next row. If the idea of rows doesn't make sense in a particular context, then the Return key should have the same effect as the Tab key.

Tables are more likely than forms to support range selections and extended selections. A table can also support selection of rows and columns. The most convenient way for the user to select a column is to click in the column header. To select more than one column, the user drags through several column headers. The same applies to rows.

To select part of the contents of a field, the user must first select the field. The user then clicks again to select the desired part of the field. Since the contents of a field are either text or graphics, this type of selection follows the rules outlined above. Figure 1Ø shows some selections in an array.

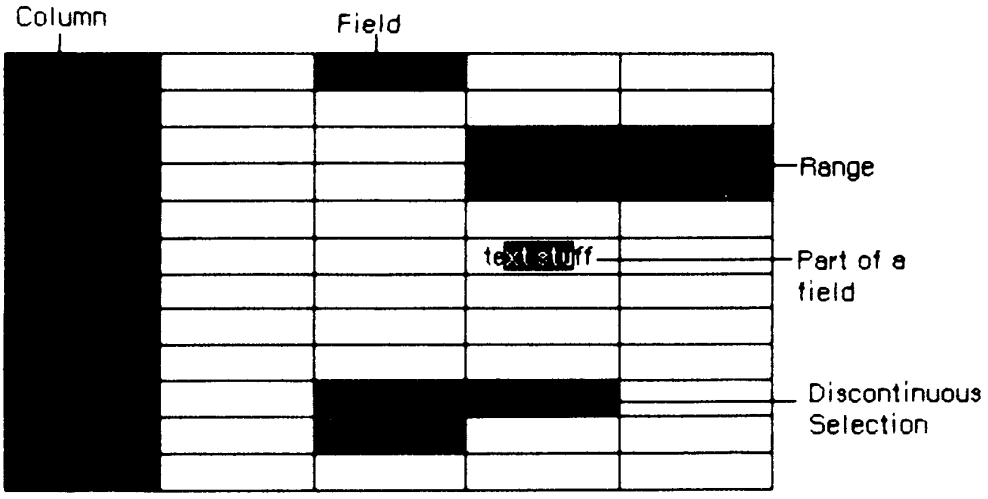


Figure 1Ø. Array Selections

WINDOWS

Windows are the rectangles on the desktop that display information. The most commmon types of windows are document windows, desk accessories, dialog boxes, and alert boxes. (Dialog and alert boxes

are discussed under "Dialogs and Alerts".) Some of the features described in this section are applicable only to document windows. Figure 11 shows a typical active window and some of its components.

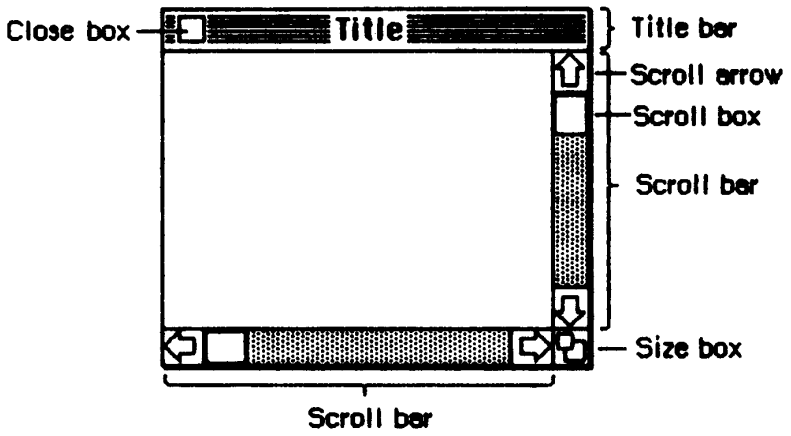


Figure 11. An Active Window

Multiple Windows

Some applications may be able to keep several windows on the desktop at the same time. Each window is in a different plane. Windows can be moved around on the Macintosh's desktop much like pieces of paper can be moved around on a real desktop. Each window can overlap those behind it, and can be overlapped by those in front of it. Even when windows don't overlap, they retain their front-to-back ordering.

Different windows can represent:

- different parts of the same document, such as the beginning and end of a long report
- different interpretations of the same document, such as the tabular and chart forms of a set of numerical data
- related parts of a logical whole, like the listing, execution, and debugging of a program
- separate documents being viewed or edited simultaneously

Each application may deal with the meaning and creation of multiple windows in its own way.

The advantage of multiple windows is that the user can isolate unrelated chunks of information from each other. The disadvantage is that the desktop can become cluttered, especially if some of the

windows can't be moved. Figure 12 shows multiple windows.

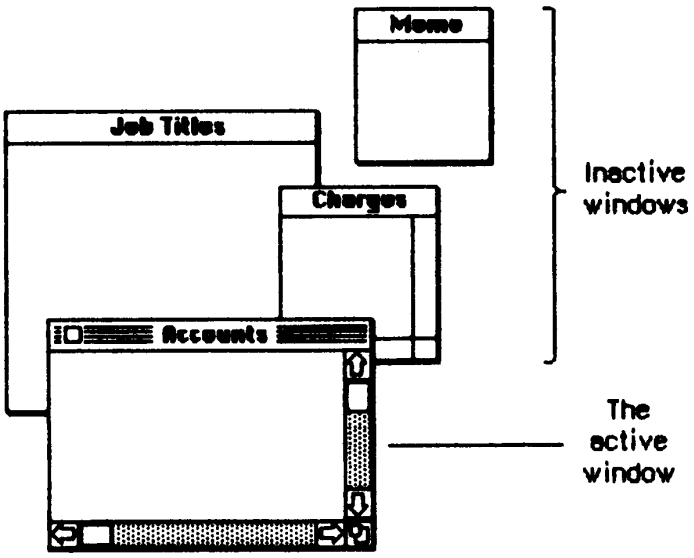


Figure 12. Multiple Windows

Opening and Closing Windows

Windows come up onto the screen in different ways as appropriate to the purpose of the window. The application controls at least the initial size and placement of its windows.

Most windows have a close box that, when clicked, makes the window go away. The application in control of the window determines what's done with the window visually and logically when the close box is clicked. Visually, the window can either shrink to a smaller object such as an icon, or leave no trace behind when it closes. Logically, the information in the window is either retained and then restored when the window is reopened (which is the usual case), or else the window is reinitialized each time it's opened. When a document is closed, the user is given the choice whether to save any changes made to the document since the last time it was saved.

If an application doesn't support closing a window with a close box, it should not include a close box on the window.

The Active Window

Of all the windows that are open on the desktop, the user can work in only one window at a time. This window is called the active window. All other open windows are inactive. To make a window active, the user clicks in it. Making a window active has two immediate consequences:

- The window's title bar is highlighted, the scroll bars and size box are shown, and any controls inside the window become active. If the window is being reactivated, the selection that was in effect when it was deactivated is rehighlighted.

- The window is moved to the frontmost plane, so that it's shown in front of any windows that it overlaps.

Clicking in a window does nothing except activate it. To make a selection in the window, the user must click again. When the user clicks in a window that has been deactivated, the window should be reinstated just the way it was when it was deactivated, with the same position of the scroll box, and the same selection highlighted.

When a window becomes inactive, all the visual changes that took place when it was activated are reversed. The title bar becomes unhighlighted, the scroll bars and size box aren't shown, any controls inside the window are dimmed, and no selection is shown in the window.

Moving a Window

Each application initially places windows on the screen wherever it wants them. The user can move a window--to make more room on the desktop or to uncover a window it's overlapping--by dragging it by its title bar. As soon as the user presses in the title bar, that window becomes the active window. A dotted outline of the window follows the pointer until the user releases the mouse button. At the release of the button the full window is drawn in its new location. Moving a window doesn't affect the appearance of the document within the window.

If the user holds down the Command key while moving the window, the window isn't made active; it moves in the same plane.

The application should ensure that a window can never be moved completely off the screen.

Changing the Size of a Window

If a window has a size box in its bottom right corner, where the scroll bars come together, the user can change the size of the window--enlarging or reducing it to the desired size.

Dragging the size box attaches a dotted outline of the window to the pointer. The outline's top left corner stays fixed, while the bottom

right corner follows the pointer. When the mouse button is released, the entire window is redrawn in the shape of the dotted outline.

Moving windows and sizing them go hand in hand. If a window can be moved, but not sized, then the user ends up constantly moving windows on and off the screen. The reason for this is that if the user moves the window off the right or bottom edge of the screen, the scroll bars are the first thing to disappear. To scroll the window, the user must move the window back onto the screen again. If, on the other hand, the window can be resized, then the user can change its size instead of moving it off the screen, and will still be able to scroll.

Sizing a window doesn't change the position of the top left corner of the window over the document or the appearance of the part of the view that's still showing; it changes only how much of the view is visible inside the window. One exception to this rule is a command such as Reduce to Fit in MacDraw, which changes the scaling of the view to fit the size of the window. If, after choosing this command, the user resizes the window, the application changes the scaling of the view.

The application can define a minimum window size. Any attempt to shrink the window below this size is ignored.

Scroll Bars

Scroll bars are used to change which part of a document view is shown in a window. Only the active window can be scrolled.

A scroll bar (see Figure 11 above) is a light gray shaft, capped on each end with square boxes labeled with arrows; inside the shaft is a white rectangle. The shaft represents one dimension of the entire document; the white rectangle (called the scroll box) represents the location of the portion of the document currently visible inside the window. As the user moves the document under the window, the position of the rectangle in the scroll bar moves correspondingly. If the document is no larger than the window, the scroll bars are inactive (the scrolling apparatus isn't shown in them). If the document window is inactive, the scroll bars aren't shown at all.

There are three ways to move the document under the window: by sequential scrolling, by "paging" windowful by windowful through the document, and by directly positioning the scroll box.

Clicking a scroll arrow moves the document in the opposite direction from the scroll arrow. For example, when the user clicks the top scroll arrow, the document moves down, bringing the view closer to the top of the document. The scroll box moves towards the arrow being clicked.

Each click in a scroll arrow causes movement a distance of one unit in the chosen direction, with the unit of distance being appropriate to the application: one line for a word processor, one row or column for a spreadsheet, and so on. Within a document, units should always be

the same size, for smooth scrolling. Pressing the scroll arrow causes continuous movement in its direction.

Clicking the mouse anywhere in the gray area of the scroll bar advances the document by windowfuls. The scroll box, and the document view, move toward the place where the user clicked. Clicking below the scroll box, for example, brings the user the next windowful towards the bottom of the document. Pressing in the gray area keeps windowfuls flipping by until the user releases the button, or until the location of the scroll box catches up to the location of the pointer. Each windowful is the height or width of the window, minus one unit overlap (where a unit is the distance the view scrolls when the scroll arrow is clicked once).

In both the above schemes the user moves the document incrementally until it's in the proper position under the window; as the document moves, the scroll box moves accordingly. The user can also move the document directly to any position simply by moving the scroll box to the corresponding position in the scroll bar. To move the scroll box, the user drags it along the scroll bar; an outline of the scroll box follows the pointer. When the mouse button is released, the scroll box jumps to the position last held by the outline, and the document jumps to the position corresponding to the new position of the scroll box.

If the user starts dragging the scroll box, and then moves the pointer a certain distance outside the scroll bar, the scroll box detaches itself from the pointer and stops following it; if the user releases the mouse button, the scroll box returns to its original position and the document remains unmoved. But if the user still holds the mouse button and drags the pointer back into the scroll bar, the scroll box reattaches itself to the pointer and can be dragged as usual.

If a document has a fixed size, and the user scrolls to the right or bottom edge of the document, the application displays a small amount of gray background (the same pattern as the desktop) between the edge of the document and the window frame.

Automatic Scrolling

There are several instances when the application, rather than the user, scrolls the document. These instances involve some potentially sticky problems about how to position the document within the window after scrolling.

The first case is when the user moves the pointer out of the window while selecting by dragging. The window keeps up with the selection by scrolling automatically in the direction the pointer has been moved. The rate of scrolling is the same as if the user were pressing on the corresponding scroll arrow or arrows.

The second case is when the selection isn't currently showing in the window, and the user performs an operation on it. When this happens, it's usually because the user has scrolled the document after making a

selection. In this case, the application scrolls the window so that the selection is showing before performing the operation.

The third case is when the application performs an operation whose side effect is to make a new selection. An example is a search operation, after which the object of the search is selected. If this object isn't showing in the window, the application must scroll the window so as to show it.

The second and third cases present the same problem: Where should the selection be positioned within the window after scrolling? The primary rule is that the application should avoid unnecessary scrolling; users prefer to retain control over the positioning of a document. The following guidelines should be helpful:

- If part of the new selection is already showing in the window, don't scroll at all. An exception to this rule is when the part of the selection that isn't showing is more important than the part that's showing.

- If scrolling in one orientation (horizontal or vertical) is sufficient to reveal the selection, don't scroll in both orientations.

- If the selection is smaller than the window, position the selection so that some of its context is showing on each side. It's better to put the selection somewhere near the middle of the window than right up against the corner.

- Even if the selection is too large to show in the window, it might be preferable to show some context rather than to try to fit as much as possible of the selection in the window.

Splitting a Window

Sometimes it's desirable to be able to see disjoint parts of a document simultaneously. Applications that accommodate such a capability allow the window to be split into independently scrollable panes.

Applications that support splitting a window into panes place split bars at the top of the vertical scroll bar and to the left of the horizontal one. Pressing a split bar attaches it to the pointer. Dragging the split bar positions it anywhere along the scroll bar; releasing the mouse button moves the split bar to a new position, splits the window at that location, and divides the appropriate scroll bar (horizontal or vertical) into separate scroll bars for each pane. Figure 13 shows the ways a window can be split.

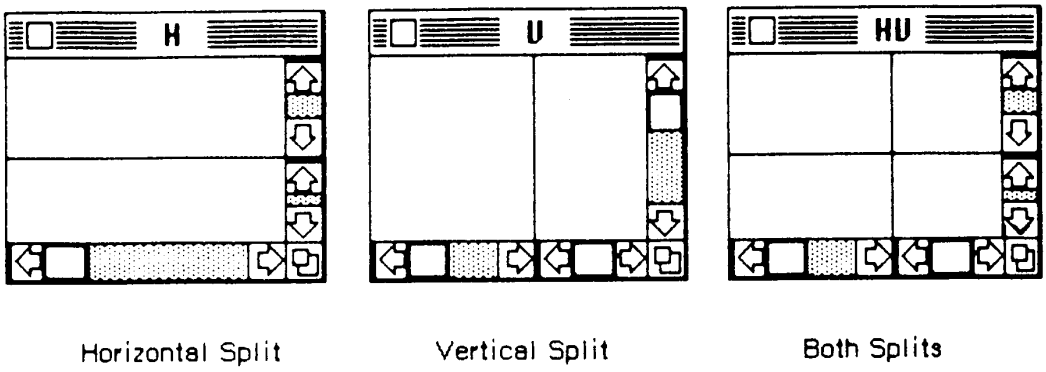


Figure 13. Types of Split Windows

After a split, the document appears the same, except for the split line lying across it. But there are now separate scroll bars for each pane. The panes are still scrolled together in the orientation of the split, but can be scrolled independently in the other orientation. For example, if the split is horizontal, then horizontal scrolling (using the scroll bar along the bottom of the window), is still synchronous. Vertical scrolling is controlled separately for each pane, using the two scroll bars along the right of the window. This is shown in Figure 14.

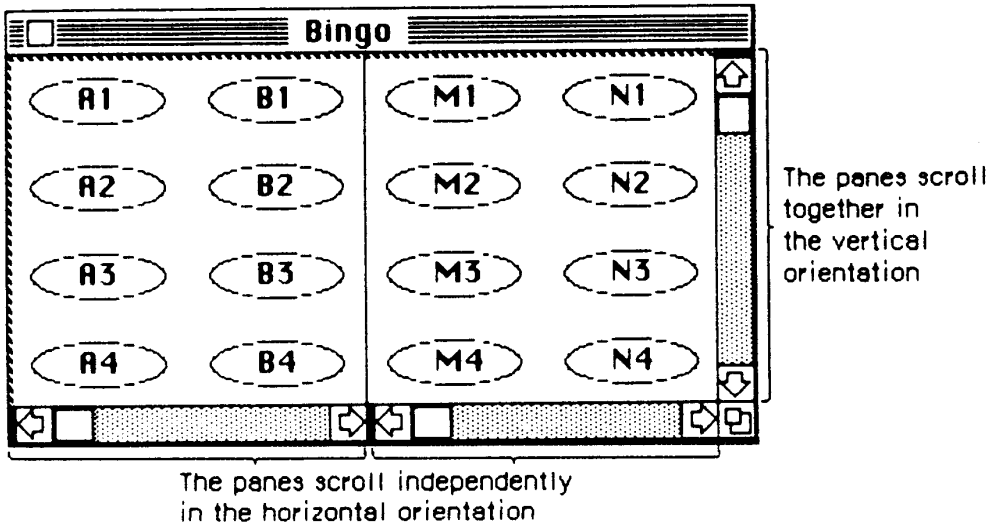


Figure 14. Scrolling a Split Window

To remove a split, the user drags the split bar to the bottom or the right of the window.

The number of views in a document doesn't alter the number of selections per document: that is, one. The active selection appears highlighted in all views that show it. If the application has to

scroll automatically to show the selection, the pane that should be scrolled is the last one that the user clicked in. If the selection is already showing in one of the panes, no automatic scrolling takes place.

Panels

If a document window is more or less permanently divided into different regions, each of which has different content, these regions are called panels. Unlike panes, which show different parts of the same document but are functionally identical, panels are functionally different from each other but might show different interpretations of the same part of the document. For example, one panel might show a graphic version of the document while another panel shows a textual version.

Panels can behave much like subwindows; they can have scroll bars, and can even be split into more than one pane. An example of a panel with scroll bars is the list of files in the Open dialog box.

Whether to use panels instead of separate windows is up to the application. Multiple panels in the same window are more compact than separate windows, but they have to be moved, opened, and closed as a unit.

COMMANDS

Once the information to be operated on has been selected, a command to operate on that information can be chosen from lists of commands called menus.

Macintosh's pull-down menus have the advantage that they're not visible until the user wants to see them; at the same time they're easy for the user to see and choose items from.

Most commands either do something, in which case they're verbs or verb phrases, or else they specify an attribute of an object, in which case they're adjectives. They usually apply to the current selection, although some commands apply to the whole document or window.

When you're designing your application, don't assume that everything has to be done through menu commands. Sometimes it's more appropriate for an operation to take place as a result of direct user manipulation of a graphic object on the screen, such as a control or icon. Alternatively, a single command can execute complicated instructions if it brings up a dialog box for the user to fill in.

The Menu Bar

The menu bar is displayed at the top of the screen. It contains a number of words and phrases: These are the titles of the menus associated with the current application. Each application has its own menu bar. The names of the menus do not change, except when the user calls for a desk accessory that uses different menus.

Only menu titles appear in the menu bar. If all of the commands in a menu are currently disabled (that is, the user can't choose them), the menu title should be dimmed (in gray type). The user can pull down the menu to see the commands, but can't choose any of them.

Choosing a Menu Command

To choose a command, the user positions the pointer over the menu title and presses the mouse button. The application highlights the title and displays the menu, as shown in Figure 15.

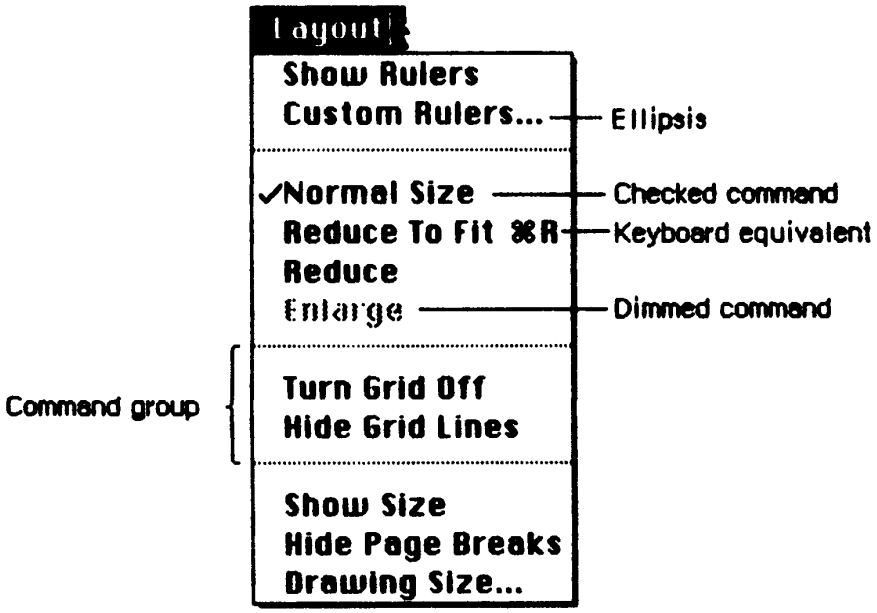


Figure 15. Menu

While holding down the mouse button, the user moves the pointer down the menu. As the pointer moves to each command, the command is highlighted. The command that's highlighted when the user releases the mouse button is chosen. As soon as the mouse button is released, the command blinks briefly, the menu disappears, and the command is executed. (The user can set the number of times the command blinks in the Control Panel desk accessory.) The menu title in the menu bar

remains highlighted until the command has completed execution.

Nothing actually happens until the user chooses the command; the user can look at any of the menus without making a commitment to do anything.

The most frequently used commands should be at the top of a menu; research shows that the easiest item for the user to choose is the second item from the top. The most dangerous commands should be at the bottom of the menu, preferably isolated from the frequently used commands.

Appearance of Menu Commands

The commands in a particular menu should be logically related to the title of the menu. In addition to command names, three features of menus help the user understand what each command does: command groups, toggles, and special visual features.

Command Groups

As mentioned above, menu commands can be divided into two kinds: verbs and adjectives, or actions and attributes. An important difference between the two kinds of commands is that an attribute stays in effect until it's cancelled, while an action ceases to be relevant after it has been performed. Each of these two kinds can be grouped within a menu. Groups are separated by gray lines, which are implemented as disabled commands.

The most basic reason to group commands is to break up a menu so it's easier to read. Commands grouped for this reason are logically related, but independent. Commands that are actions are usually grouped this way, such as Cut, Copy, Paste, and Clear in the Edit menu.

Attribute commands that are interdependent are grouped to show this interdependence. Two kinds of attribute command groups are mutually exclusive groups and accumulating groups.

In a mutually exclusive attribute group, only one command in the group is in effect at the same time. The command that's in effect is preceded by a check mark. If the user chooses a different command in the group, the check mark is moved to the new command. An example is the Font menu in MacWrite; no more than one font can be in effect at a time.

In an accumulating attribute group, any number of attributes can be in effect at the same time. One special command in the group cancels all the other commands. An example is the Style menu in MacWrite: the user can choose any combination of Bold, Italic, Underline, Outline, or Shadow, but Plain Text cancels all the other commands.

Toggles

Another way to show the presence or absence of an attribute is by a toggled command. In this case, the attribute has two states, and a single command allows the user to toggle between the states. For example, when rulers are showing in MacWrite, a command in the Format menu reads "Hide Rulers". If the user chooses this command, the rulers are hidden, and the command is changed to read "Show Rulers". This kind of group should be used only when the wording of the commands makes it obvious that they're opposites.

Special Visual Features

In addition to the command names and how they're grouped, several other features of commands communicate information to the user:

- A check mark indicates whether an attribute command is currently in effect.
- An ellipsis (...) after a command name means that choosing that command brings up a dialog box. The command isn't actually executed until the user has finished filling in the dialog box and has clicked the OK button or its equivalent.
- The application dims a command when the user can't choose it. If the user moves the pointer over a dimmed item, it isn't highlighted.
- If a command can be chosen from the keyboard, it's followed by the Command key symbol and the character used to choose it. To choose a command this way, the user holds down the Command key and then presses the character key.

Some characters are reserved for special purposes, but there are different degrees of stringency. Since almost every application has a File menu and an Edit menu, the keyboard equivalents in those menus are strongly reserved, and should never be used for any other purpose:

Character	Command
C	Copy (Edit menu)
Q	Quit (File menu)
V	Paste (Edit menu)
X	Cut (Edit menu)
Z	Undo (Edit menu)

The keyboard equivalents in the Style menu are conditionally reserved. If an application has this menu, it shouldn't use these characters for any other purpose, but if it doesn't, it can use them however it likes:

Character	Command
B	Bold
I	Italic
O	Outline
P	Plain text
S	Shadow
U	Underline

One keyboard command doesn't have a menu equivalent:

Character	Command
.	Stop current operation

Several other menu features are also supported:

- A command can be shown in Bold, Italic, Outline, Underline, or Shadow type style.
- A command can be preceded by an icon.
- The application can draw its own type of menu. An example of this is the Fill menu in MacDraw.

STANDARD MENUS

One of the strongest ways in which Macintosh applications can take advantage of the consistency of the user interface is by using standard menus. The operations controlled by these menus occur so frequently that it saves considerable time for users if they always match exactly. Three of these menus, the Apple, File, and Edit menus, appear in almost every application. The Font, FontSize, and Style menus affect the appearance of text, and appear only in applications where they're relevant.

The Apple Menu

Macintosh doesn't allow two applications to be running at once. Desk accessories, however, are mini-applications that are available while using any application.

At any time the user can issue a command to call up one of several desk accessories; the available accessories are listed in the Apple menu, as shown in Figure 16.

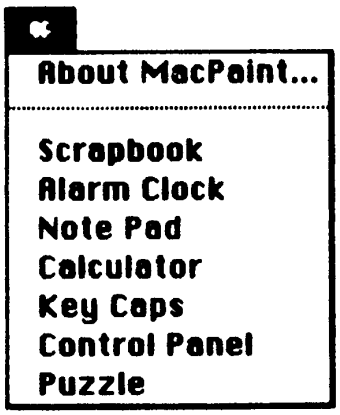


Figure 16. Apple Menu

Accessories are disk-based: Only those accessories on an available disk can be used. The list of accessories is expanded or reduced according to what's available. More than one accessory can be on the desktop at a time.

For a description of these desk accessories, see Macintosh, the owner's guide. An application can also provide its own desk accessories.

The Apple menu also contains the "About xxx" menu item, where "xxx" is the name of the application. Choosing this item brings up a dialog box with the name and copyright information for the application, as well as any other information the application wants to display.

The File Menu

The File menu allows the user to perform certain simple filing operations without leaving the application and returning to the Finder. It also contains the commands for printing and for leaving the application. The standard File menu includes the commands shown in Figure 17.

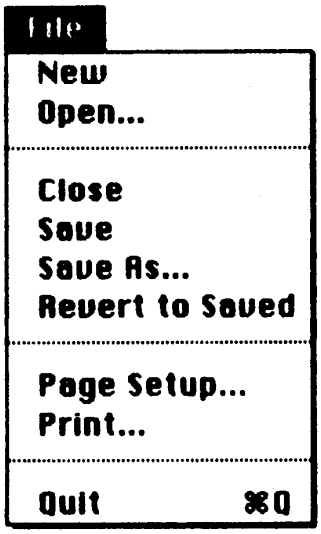


Figure 17. File Menu

Other frequently used commands are Print Draft, Print Final, and Print One. All of these commands are described below.

New

New opens a new, untitled document. The user names the document the first time it's saved. This command is disabled when the maximum number of documents allowed by the application is already open.

Open

Open opens an existing document. To select the document, the user is presented with a dialog box (Figure 18). This dialog box shows a list of all the documents on the disk whose name is displayed that can be handled by the current application. The user can scroll this list forward and backward. The dialog box also gives the user the chance to look at the documents on the disk in the other disk drive that belong to the current application, or to eject either disk.

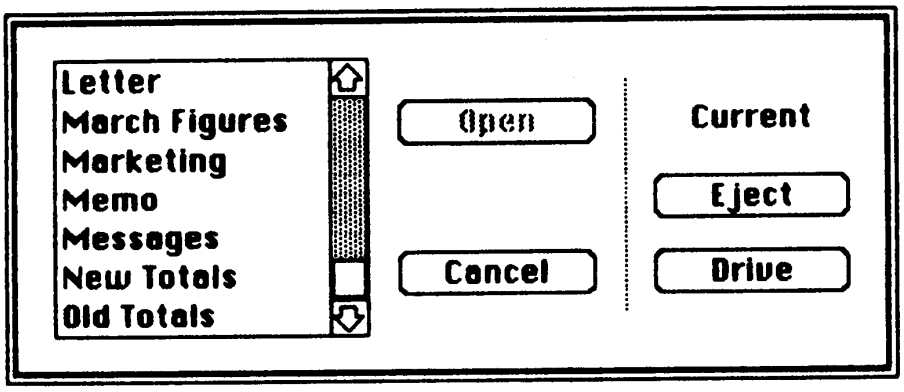


Figure 18. Open Dialog Box

Using the Open command, the user can only open a document that can be processed by the current application. Opening a document that can only be processed by a different application requires leaving the application and returning to the Finder.

This command is disabled when the maximum number of documents allowed by the application is already open.

Close

Close closes the active document or desk accessory. If the user has changed the document since the last time it was saved, the command presents an alert box giving the user the choice of whether or not to save the changes.

Clicking in the close box of a window is the same as choosing Close.

Save

Save makes permanent any changes to the active document since the last time it was saved. It leaves the document open.

If the user chooses Save for a new document that hasn't been named yet, the application presents the Save As dialog (see below) to name the document, and then continues with the save. The active document remains active.

If there's not enough room on the disk to save the document, the application asks if the user wants to save the document on another disk. If the answer is yes, the application goes through the Save As dialog to find out which disk.

Save As

Save As saves a copy of the active document under a file name provided by the user.

If the document already has a name, Save As closes the old version of the document, creates a copy, and displays the copy in the window.

If the document is untitled, Save As saves the original document under the specified name. The active document remains active.

Revert to Saved

Revert to Saved returns the document to the state it was in the last time it was saved. Before doing so, it puts up an alert box to confirm that this is what the user wants.

Page Setup

Page Setup lets the user specify printing parameters such as what its paper size and printing orientation are. These parameters remain with the document.

Print

Print lets the user specify various parameters such as print quality and number of copies, and then prints the document. The parameters apply only to the current printing operation.

Quit

Quit leaves the application and returns to the Finder. If any open documents have been changed since the last time they were saved, the application presents the same alert box as for Close, once for each document.

Other Commands

Other commands that are in the File menu in some applications include:

- Print Draft. This command prints one copy of a rough version of a document more quickly than Print. It's useful in applications where ordinary printing is slow. If an application has this command, it should change the name of the Print command to Print Final.

- Print One. This command saves time by printing one copy using default parameters without bringing up the Print dialog box.

The Edit Menu

The Edit menu contains the commands that delete, move, and copy objects, as well as commands such as Undo, Show Clipboard, and Select All. This section also discusses the Clipboard, which is controlled by the Edit menu commands. Text editing methods that don't use menu commands are discussed under "Text Editing".

If the application supports desk accessories, the order of commands in the Edit menu should be exactly as shown here. This is because, by default, the application passes the numbers, not the names, of the menu commands to the desk accessories. (For more details, see the Desk Manager manual.) In particular, your application must provide an Undo command for the benefit of the desk accessories, even if it doesn't support the command (in which case it can disable the command until a desk accessory is opened).

The standard order of commands in the Edit menu is shown in Figure 19.

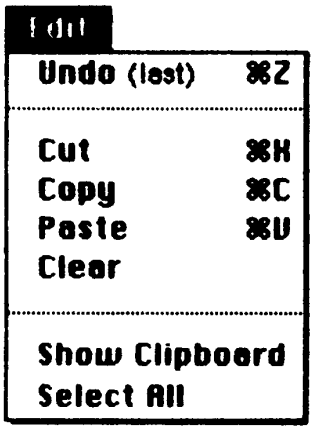


Figure 19. Edit Menu

The Clipboard

The Clipboard is a special kind of window with a well-defined function: it holds whatever is cut or copied from a document. Its contents stay intact when the user changes documents, opens a desk accessory, or leaves the application. An application can choose whether to have the Clipboard open or closed when the application starts up.

The Clipboard looks like a document window, with a close box but with no scroll bars. Its contents cannot be edited.

Every time the user performs a Cut or Copy on the current selection, a copy of the selection replaces the previous contents of the Clipboard. The previous contents are kept around in case the user chooses Undo.

The user can see the contents of the Clipboard but can't edit them. In most other ways the Clipboard behaves just like any other window.

There is only one Clipboard, which is present for all applications that support Cut, Copy, and Paste. The user can see the Clipboard window by choosing Show Clipboard from the Edit menu. If the window is already showing, it's hidden by choosing Hide Clipboard. (Show Clipboard and Hide Clipboard are a single toggled command.)

Because the contents of the Clipboard remain unchanged when applications begin and end, or when the user opens a desk accessory, the Clipboard can be used for transferring data among mutually compatible applications and desk accessories.

Undo

Undo reverses the effect of the previous operation. Not all operations can be undone; the definition of an undoable operation is somewhat application-dependent. The general rule is that operations that change the contents of the document are undoable, and operations that don't are not. Most menu items are undoable, and so are typing sequences.

A typing sequence is any sequence of characters typed from the keyboard or numeric keypad, including Backspace, Return, and Tab, but not including keyboard equivalents of commands.

Operations that aren't undoable include selecting, scrolling, and splitting the window or changing its size or location. None of these operations interrupts a typing sequence. That is, if the user types a few characters and then scrolls the document, the Undo command still undoes the typing. Whenever the location affected by the Undo operation isn't currently showing on the screen, the application should scroll the document so the user can see the effect of the Undo.

An application should also allow the user to undo any operations that are initiated directly on the screen, without a menu command. This includes operations controlled by setting dials, clicking check boxes, and so on, as well as drawing graphic objects with the mouse.

The actual wording of the Undo command as it appears in the Edit menu is "Undo xxx", where xxx is the name of the last operation. If the last operation isn't a menu command, use some suitable term after the word Undo. If the last operation can't be undone, the command reads "Undo", but is disabled.

If the last operation was Undo, the menu command says "Redo xxx", where xxx is the operation that was undone. If this command is chosen, the Undo is undone.

Cut

The user chooses Cut either to delete the current selection or to move it. If it's a move, it's eventually completed by choosing Paste.

When the user chooses Cut, the application removes the current selection from the document and puts it in the Clipboard, replacing the Clipboard's previous contents. The place where the selection used to be becomes the new selection; the visual implications of this vary among applications. For example, in text, the new selection is an insertion point, while in an array, it's an empty but highlighted cell. If the user chooses Paste immediately after choosing Cut, the document should be just as it was before the cut; the Clipboard is unchanged.

When the user chooses Cut, the application doesn't know if it's a deletion or the first step of a move. Therefore, it must be prepared for either possibility.

Copy

Copy is the first stage of a copy operation. Copy puts a copy of the selection in the Clipboard, but the selection also remains in the document.

Paste

Paste is the last stage of a copy or move operation. It pastes the contents of the Clipboard to the document, replacing the current selection. The user can choose Paste several times in a row to paste multiple copies. After a paste, the new selection is the object that was pasted, except in text, where it's an insertion point immediately after the pasted text. The Clipboard remains unchanged.

Clear

When the user chooses Clear, or presses the Clear key on the numeric keypad, the application removes the selection, but doesn't put it on the Clipboard. The new selection is the same as it would be after a Cut.

Show Clipboard

Show Clipboard is a toggled command. Initially, the Clipboard isn't displayed, and the command is "Show Clipboard". If the user chooses the command, the Clipboard is displayed and the command changes to "Hide Clipboard".

Select All

Select All selects every object in the document.

Font-Related Menus

Three standard menus affect the appearance of text: Font, which determines the font of a text selection; FontSize, which determines the size of the characters; and Style, which determines aspects of its appearance such as boldface, italics, and so on.

Font Menu

A font is a set of typographical characters created with a consistent design. Things that relate characters in a font include the thickness of vertical and horizontal lines, the degree and position of curves and swirls, and the use of serifs. A font has the same general appearance, regardless of the size of the characters. The Font menu always lists the fonts that are currently available. Figure 2Ø shows a Font menu with some of the most common fonts.

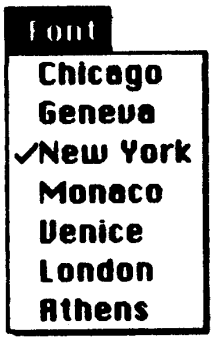


Figure 2Ø. Font Menu

FontSize Menu

Font sizes are measured in points; a point is about 1/72 of an inch. Each font is available in predefined sizes. The numbers of these sizes for each font are shown outlined in the FontSize menu. The font can also be scaled to other sizes, but it may not look as good. Figure 21 shows a FontSize menu with the standard font sizes.

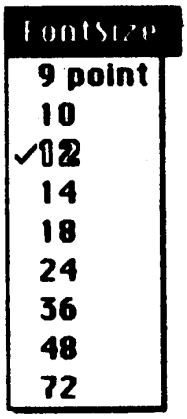


Figure 21. FontSize Menu

If there's insufficient room in the menu bar for the word FontSize, it can be abbreviated to Size. If there's insufficient room for both a Font menu and a Size menu, the sizes can be put at the end of the Font or Style menu.

Style Menu

The commands in the Style menu are Plain Text, Bold, Italic, Underline, Outline, and Shadow. All the commands except Plain Text are accumulating attributes; the user can choose any combination. They are also toggled commands; a command that's in effect for the current selection is preceded by a check mark. Plain Text cancels all the other choices. Figure 22 shows these styles.

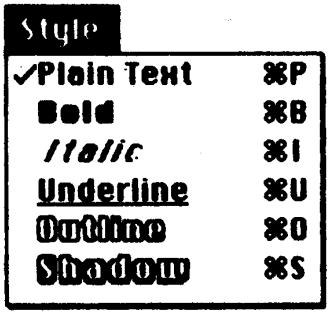


Figure 22. Style Menu

TEXT EDITING

In addition to the operations described under "The Edit Menu" above, there are other ways to edit text that don't use menu items.

Inserting Text

To insert text, the user selects an insertion point by clicking where the text is to go, and then starts typing it. As the user types, the application continually moves the insertion point to the right of each new character.

Applications with multiline text blocks should support word wraparound, according to the definition of a word given above. The intent is that no word be broken between lines.

Backspace

When the user presses the Backspace key, one of two things happens:

- If the current selection is one or more characters, it's deleted.
- If the current selection is an insertion point, the previous character is deleted.

In both cases, the deleted characters don't go into the Clipboard, and the insertion point replaces the deleted characters in the document.

Replacing Text

If the user starts typing when the selection is one or more characters, the characters that are typed replace the selection. The deleted characters don't go into the Clipboard, but the replacement can be undone by immediately choosing Undo.

Intelligent Cut and Paste

An application that lets the user select a word by double-clicking should also see to it that the user doesn't regret using this feature. The only way to do this is by providing "intelligent" cut and paste.

To understand why this feature is necessary, consider the following sequence of events in an application that doesn't provide it:

1. A sentence in the user's document reads: "Returns are only accepted if the merchandise is damaged." The user wants to change this to: "Returns are accepted only if the merchandise is damaged."

2. The user selects the word "only" by double-clicking. The letters are highlighted, but not either of the adjacent spaces.

3. The user chooses Cut, clicks just before the word "if", and chooses Paste.

4. The sentence now reads: "Returns are accepted onlyif the merchandise is damaged." To correct the sentence, the user has to remove a space between "are" and "accepted", and add one between "only" and "if". At this point he or she may be wondering why the Macintosh is supposed to be easier to use than other computers.

If an application supports intelligent cut and paste, the rules to follow are:

- If the user selects a word or a range of words, highlight the selection, but not any adjacent spaces.

- When the user chooses Cut, if the character to the left of the selection is a space, discard it.

- When the user chooses Paste, if the character to the left of the current selection isn't a space, add a space. If the character to the right of the current selection isn't a punctuation mark or a space, add a space. Punctuation marks include the period, comma, exclamation point, question mark, apostrophe, colon, semicolon, and quotation mark.

This feature makes more sense if the application supports the full definition of a word (as detailed above under "Selecting a Word"), rather than the definition of a word as anything between two spaces.

These rules apply to any selection that's one or more whole words, whether it was chosen with a double-click or as a range selection.

Figure 23 shows some examples of intelligent cut and paste.

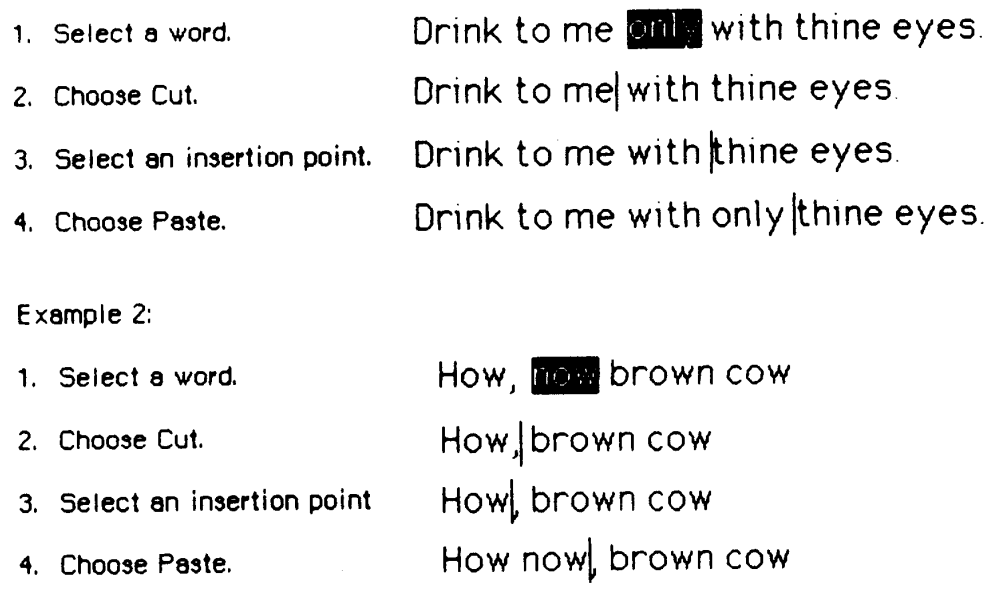


Figure 23. Intelligent Cut and Paste

Editing Fields

If an application isn't primarily a text application, but does use text in fields (such as in a dialog box), it may not be able to provide the full text editing capabilities described so far.

It's important, however, that whatever editing capabilities the application provides under these circumstances be upward-compatible with the full text editing capability. The following list shows the capabilities that can be provided, going from the minimal to the most sophisticated:

- The user can select the whole field and type in a new value.
- The user can backspace.
- The user can select a substring of the field and replace it.
- The user can select a word by double-clicking.
- The user can choose Undo, Cut, Copy, Paste, and Clear, as described above under "The Edit Menu". In the most sophisticated version, the application implements intelligent cut and paste.

An application should also perform appropriate edit checks. For example, if the only legitimate value for a field is a string of digits, the application might issue an alert if the user typed any nondigits. Alternatively, the application could wait until the user is through typing before checking the validity of the contents of the

field. In this case, the appropriate time to check the field is when the user clicks anywhere other than within the field.

DIALOGS AND ALERTS

The "select-then-choose" paradigm is sufficient whenever operations are simple and act on only one object. But occasionally a command will require more than one object, or will need additional parameters before it can be executed. And sometimes a command won't be able to carry out its normal function, or will be unsure of the user's real intent. For these special circumstances the Macintosh user interface includes two additional features:

- dialogs, to allow the user to provide additional information before a command is executed

- alerts, to notify the user whenever an unusual situation occurs

Since both of these features lean heavily on controls, controls are described in this section, even though controls are also used in other places.

Controls

Friendly systems act by direct cause-and-effect; they do what they're told. Performing actions on a system in an indirect fashion reduces the sense of direct manipulation. To give Macintosh users the feeling that they're in control of their machines, many of an application's features are implemented with controls: graphic objects that, when directly manipulated by the mouse, cause instant action with visible results.

There are four main types of controls: buttons, check boxes, radio buttons, and dials. These four kinds are shown in Figure 24.

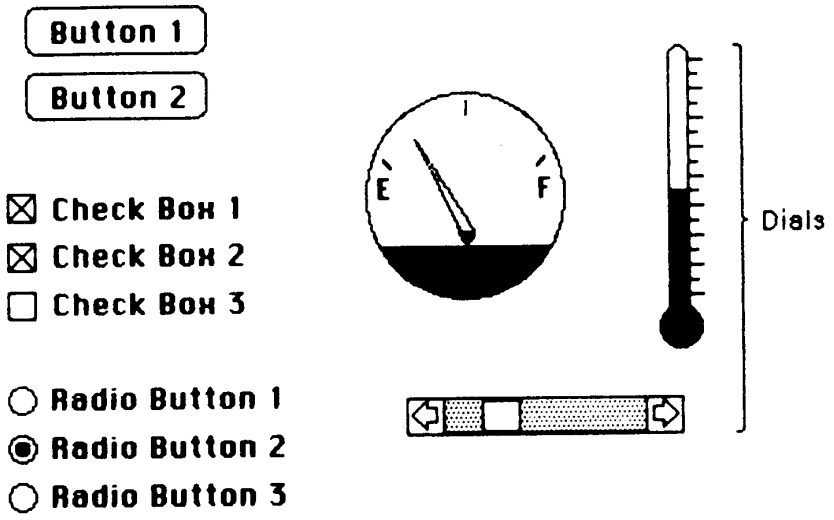


Figure 24. Controls

Buttons

Buttons are small objects, usually inside a window, labeled with text. Clicking or pressing a button performs the action described by the button's label.

Buttons perform instantaneous actions, such as completing operations defined by a dialog box or acknowledging error messages. Conceivably they could perform continuous actions, in which case the effect of pressing on the button would be the same as the effect of clicking it repeatedly.

Two particular buttons, OK and Cancel, are especially important in dialogs and alerts; they're discussed under those headings below.

Check Boxes and Radio Buttons

Whereas buttons perform instantaneous or continuous actions, check boxes and radio buttons let the user choose among alternative values for a parameter.

Check boxes act like toggle switches; they're used to indicate the state of a parameter that must be either off or on. The parameter is on if the box is checked, otherwise it's off. The check boxes appearing together in a given context are independent of each other; any number of them can be off or on.

Radio buttons typically occur in groups; they're round and are filled in with a black circle when on. They're called radio buttons because

they act like the buttons on a car radio. At any given time, exactly one button in the group is on. Clicking one button in a group turns off the current button.

Both check boxes and radio buttons are accompanied by text that identifies what each button does.

Dials

Dials display the value, magnitude, or position of something in the application or system, and optionally allow the user to alter that value. Dials are predominantly analog devices, displaying their values graphically and allowing the user to change the value by dragging an indicator; dials may also have a digital display.

The most common example of a dial is the scroll bar. The indicator of the scroll bar is the scroll box; it represents the position of the window over the length of the document. The user can drag the scroll box to change that position.

Dialogs

Commands in menus normally act on only one object. If a command needs more information before it can be performed, it presents a dialog box to gather the additional information from the user. The user can tell which commands bring up dialog boxes because they're followed by an ellipsis (...) in the menu.

A dialog box is a rectangle that may contain text, controls, and icons. There should be some text in the box that indicates which command brought up the dialog box.

Other than explanatory text, the contents of a dialog box are all objects that the user sets to provide the needed information. These objects include controls and text fields. When the application puts up the dialog box, it should set the controls to some default setting and fill in the text fields with default values, if possible. One of the text fields (the "first" field) should be highlighted, so that the user can change its value just by typing in the new value. If all the text fields are blank, there should be an insertion point in the first field.

Editing text fields in a dialog box should conform to the guidelines detailed above, under "Text Editing".

When the user is through editing an item:

- Pressing Tab accepts the changes made to the item, and selects the next item in sequence.

- Clicking in another item accepts the changes made to the previous item and selects the newly clicked item.

Dialog boxes are either modal or modeless, as described below.

Modal Dialog Boxes

A modal dialog box is one that the user must explicitly dismiss before doing anything else, such as making a selection outside the dialog box or choosing a command. Figure 25 shows a modal dialog box.

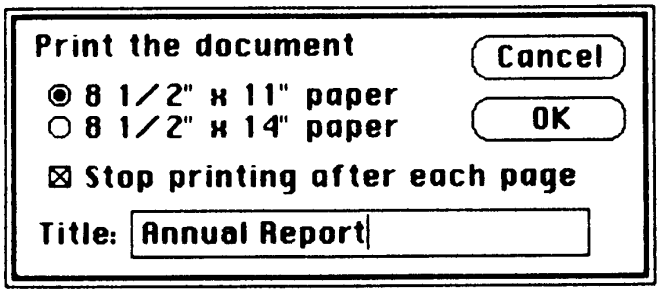


Figure 25. A Modal Dialog Box

Because it restricts the user's freedom of action, this type of dialog box should be used sparingly. In particular, the user can't choose a menu item while a modal dialog box is up, and therefore can only do the simplest kinds of text editing.

For these reasons, the main use of a modal dialog box is when it's important for the user to complete an operation before doing anything else.

A modal dialog box usually has at least two buttons: OK and Cancel. OK dismisses the dialog box and performs the original command according to the information provided; it can be given a more descriptive name than "OK". Cancel dismisses the dialog box and cancels the original command; it must always be called "Cancel".

A dialog box can have other kinds of buttons as well; these may or may not dismiss the dialog box. One of the buttons in the dialog box may be outlined boldly. The outlined button is the default button; if no button is outlined, then the OK button is the default button. The default button should be the safest button in the current situation. Pressing the Return or Enter key has the same effect as clicking the default button. If there is no default button, then Return and Enter have no effect.

A special type of modal dialog box is one with no buttons. This type of box is just to inform the user of a situation without eliciting any response. Usually, it would describe the progress of an ongoing operation. Since it has no buttons, the user has no way to dismiss it. Therefore, the application must leave it up long enough for the user to read it before taking it down again.

Modeless Dialog Boxes

A modeless dialog box allows the user to perform other operations without dismissing the dialog box. Figure 26 shows a modeless dialog box.

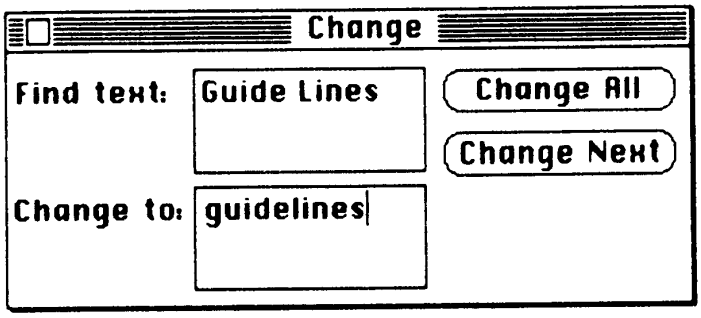


Figure 26. A Modeless Dialog Box

A modeless dialog box is dismissed by clicking in the close box or by choosing Close when the dialog is active. The dialog box is also dismissed implicitly when the user chooses Quit. It's usually a good idea for the application to remember the contents of the dialog box after it's dismissed, so that when it's opened again, it can be restored exactly as it was.

Controls work the same way in modeless dialog boxes as in modal dialog boxes, except that buttons never dismiss the dialog box. In this context, the OK button means "go ahead and perform the operation, but leave the dialog box up", while Cancel usually terminates an ongoing operation.

A modeless dialog box can also have text fields; since the user can choose menu commands, the full range of editing capabilities can be made available.

Alerts

Every user of every application is liable to do something that the application won't understand. From simple typographical errors to slips of the mouse to trying to write on a protected disk, users will do things an application can't cope with in a normal manner. Alerts give applications a way to respond to errors not only in a consistent manner, but in stages according to the severity of the error, the user's level of expertise, and the particular history of the error.

The two kinds of alerts are beeps and alert boxes.

Beeps are used for errors that are both minor and immediately obvious. For example, if the user tries to backspace past the left boundary of a text field, the application could choose to beep instead of putting up an alert box. A beep can also be part of a staged alert, as described below.

An alert box looks like a modal dialog box, except that it's somewhat narrower and appears lower on the screen. An alert box is primarily a one-way communication from the system to the user; the only way the user can respond is by clicking buttons. Therefore alert boxes might contain dials and buttons, but usually not text fields, radio buttons, or check boxes. Figure 27 shows a typical alert box.

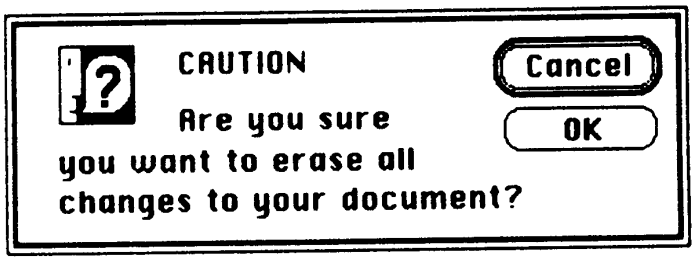


Figure 27. An Alert Box

There are three types of alert boxes:

- Note: A minor mistake that wouldn't have any disastrous consequences if left as is.

- Caution: An operation that may or may not have undesirable results if it is allowed to continue. The user is given the choice whether or not to continue.

- Stop: A situation that requires remedial action by the user. The situation could be either a serious problem, or something as simple as a request by the application to the user to change diskettes.

An application can define several stages for an alert, so that if the user persists in the same mistake, the application can issue increasingly more helpful (or sterner) messages. A typical sequence is for the first two occurrences of the mistake to result in a beep, and for subsequent occurrences to result in an alert box. This type of sequence is especially appropriate when the mistake is one that has a high probability of being accidental. An example is when the user chooses Cut when the selection is an insertion point.

How the buttons in an alert box are labeled depends on the nature of the box. If the box presents the user with a situation in which no alternative actions are available, the box has a single button that says OK. Clicking this button means "I have read the alert." If the user is given alternatives, then typically the alert is phrased as a question that can be answered "yes" or "no". In this case, buttons labeled Yes and No are appropriate, although some variation such as Save and Don't Save is also acceptable. OK and Cancel can be used, as long as their meaning isn't ambiguous.

The preferred (safest) button to use in the current situation is boldly outlined. This is the alert's default button; its effect occurs if the user presses Return or Enter.

It's important to phrase messages in alert boxes so that users aren't left guessing the real meaning. Avoid computer jargon.

Use icons whenever possible. Graphics can better describe some error situations than words, and familiar icons help users distinguish their alternatives better. Icons should be internationally comprehensible; they should not contain any words, or any symbols that are unique to a particular country.

Generally, it's better to be polite than abrupt, even if it means lengthening the message. The role of the alert box is to be helpful and make constructive suggestions, not to give out orders. But its focus is to help the user solve the problem, not to give an interesting but academic description of the problem itself.

Under no circumstances should an alert message refer the user to external documentation for further clarification. It should provide an adequate description of the information needed by the user to take appropriate action.

The best way to make an alert message understandable is to think carefully through the error condition itself. Can the application handle this without an error? Is the error specific enough so that the user can fix the situation? What are the recommended solutions? Can the exact item causing the error be displayed in the alert message?

DO'S AND DON'TS OF A FRIENDLY USER INTERFACE

Do:

- Let the user have as much control as possible over the appearance of objects on the screen--their arrangement, size, and visibility.
- Use verbs as menu commands.
- Make alert messages self-explanatory.

- Use controls and other graphics instead of just menu commands.

- Take the time to use good graphic design; it really helps.

Don't:

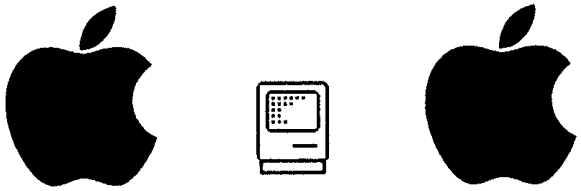
- Overuse modes, including modal dialog boxes.

- Require using the keyboard for an operation that would be easier with the mouse, or require using the mouse for an operation that would be easier with the keyboard.

- Change the way the screen looks unexpectedly, especially by scrolling automatically more than necessary.

- Make up your own menus and then give them the same names as standard menus.

- Take an old-fashioned prompt-based application originally developed for another machine and pass it off as a Macintosh application.



END OF DOCUMENT