The Windows Interface: An Application Design Guide

Microsoft® WINDOWS™
SOFTWARE DEVELOPMENT KIT

Microsoft® Windows™

Version 3.1

The Windows™ Interface An Application Design Guide

For the Microsoft® Windows™ Operating System

Microsoft Corporation

Document No. PC28921-0692

Contents

Introduction

This design guide provides guidelines for developing user interfaces for applications that run in the Microsoft® Windows™ graphical environment. It describes the components of the Windows user interface and explains design principles for software developers and designers of Windows-based applications.

Purpose

The purpose of this design guide is to promote visual and functional consistency within and across Windows-based applications. This has several advantages: When the interface for applications is consistent, users can move from one application to another with ease and speed. Consistency facilitates the learning process and minimizes the need for training when new applications are introduced into the workplace, resulting in increased productivity. Consistency also alleviates the confusion introduced by applications with divergent interfaces and eliminates the associated costs in efficiency and training. It gives users a sense of stability, which increases their confidence in the reliability of an application and in all applications with the same interface.

Scope

Most of the elements and techniques described in this guide are incorporated in the Windows operating system, version 3.1; some of these may not be available in earlier versions of the Windows environment. See your Microsoft Windows Software Development Kit (SDK) documentation for specific information on what is supported.

Because of the evolving nature of applications, this guide cannot provide specific recommendations for every possible interface issue. If an application requires elements or techniques not discussed in this guide, designers or developers may extend the existing guidelines in accordance with the principles in Chapter 1. They should review these guidelines as the minimum requirements for consistency with other Windows-based applications, and should evaluate their applications accordingly.

This guide focuses on recommendations that are specific to the development of Windows-based applications. These guidelines were developed to be generally compatible with guidelines that may be appropriately applied from the IBM® Common User Access (CUA) version 2.0 definition, published in *IBM Common User Access: Advanced Interface Design Guide* (Boca Raton, FL: IBM, 1989); but they are not intended to describe user interface requirements for CUA compliance.

The guidelines include recommendations that are generally applicable to a variety of Windows version 3.0 applications, as well as recommendations relating to new Windows version 3.1 features.

The information in this guide is tailored to applications developed for English-speaking countries. However, many of the guidelines are generally applicable to applications developed for non-English-speaking countries as well.

Implementation

The Microsoft Windows Software Development Kit (SDK) contains code samples for implementing many of the features described in this guide. See the SDK documentation for more information.

Support for Input Methods

This guide describes techniques for the mouse and keyboard, and includes a discussion of the pen which is gaining increasing acceptance as an input device. While a mouse is the preferred means of interaction in many cases, applications should also provide keyboard access to the interface for users who have systems without a mouse or who prefer using the keyboard. For applications that rely on the keyboard for entering data (for example, database, word-processing, or spreadsheet programs), keyboard access might be the preferred method of interaction for many users who do not wish to remove their hands from the keys. On the other hand, you may find that keyboard access to many features in graphics applications (such as drawing or painting programs) is a hindrance rather than a help. For this reason, you may wish to avoid keyboard access wherever its use is cumbersome.

Recommendation Levels

As stated earlier, the purpose of this design guide is to promote visual and functional consistency within and across Windows-based applications. The information presented is provided as a tool for those who would like to use it. Developers may choose to adopt any number of the guidelines in their own user interface designs. There is no conformance requirement, expressed or implied, in this set of guidelines.

The following definitions are provided to give designers an idea of the importance of specific guidelines:

- Guidelines that are labeled "recommended" represent the common way specific features, functions, operations, or behaviors should be implemented for the greatest degree of consistency. All recommended items need not be included in an application; however, if the items exist, the guidelines describe the preferred means of implementation. For example, although F6 is the recommended key for switching between panes, all applications need not include panes. Some of the recommended guidelines may be defined by the system (for example, ALT+ESC); others may be defined by the application (for example, F1).
- Guidelines that are labeled "optional" are common extensions that an application may implement. If these features or operations are implemented, the stated guidelines are the preferred means of implementation. The File and View menus illustrate the difference between recommended and optional guidelines: The File menu is recommended for all applications that provide access to data through files. The View menu is optional, but its contents follow preferred implementation guidelines.
- Guidelines that are labeled "suggested" state a particular direction that applications should follow, to the extent that the guidelines apply and do not conflict with other uses in the application.

Notation for Keys and Key Combinations

- Key names appear in small capitals; for example, CTRL or SHIFT.
- Simultaneous key combinations are linked by plus signs; for example, CTRL+B or CTRL+SHIFT+B. This notation indicates that the user should hold down the CTRL key while pressing the B key; or hold down the CTRL and SHIFT keys while pressing the B key.
- Sequential key combinations are linked by commas; for example: ALT,F. This notation indicates that the user should press and release the ALT key, and then press and release the F key.

Principles and Methodology

Chapter 1

1.1 Principles of User Interface Design

Because applications continually evolve, it is impossible to provide specific recommendations that cover every possible interface issue. Applications should follow the general principles in this section, even when they include elements and techniques not covered in this guide.

1.1.1 User Control

One of the most important principles of user interface design is that the user should always be in control of the application, not vice versa. This principle has several implications.

First, applications should always be as interactive as possible. The user should not have to wait a long time for processing to be completed. In general, applications should avoid modes that severely restrict the interactions available to the user at any given time. If modes must be used, they should be visually obvious (for example, the pointer can change shape), easy to learn, and easy to get out of.

A second implication of the principle of user control concerns customization. Because users' abilities and preferences vary, users should be able to customize aspects of the interface (including aesthetic qualities, like color and function) such as the content and structure of menus. However, designers should provide good defaults and should not depend on the user customizing these settings. (Customization methods are discussed in Chapter 11, section 11.2.)

Finally, the interface should facilitate the user's tasks rather than calling attention to itself. The best interface is often the one that is hardly noticed. Users want to accomplish tasks, not to use computers; they want to write letters, calculate profits, manage projects, and prepare presentations—not to slide scroll boxes, open drop-down lists, pull down menus, and navigate among dialog boxes.

1.1.2 Directness

The interface should give users direct and intuitive ways to accomplish their tasks. The object-action paradigm supports this principle. The user performs tasks by selecting an object (such as an icon, a window, or some text), then selecting an action (such as move, close, or underline) for that object.

Manipulating objects directly, although not appropriate in all situations, is often easier than typing complex commands. For example, it is much easier to move a window by dragging it with the mouse than by visually estimating new coordinates and then typing them into a dialog box.

1.1.3 Consistency

Two broad categories of consistency are particularly important: consistency with the real world, and consistency within and among applications. First, applications should build on the user's real-world experience by exploiting concrete metaphors and natural mapping relationships. The use of familiar concepts and metaphors reduces the amount of new material that users must learn and thereby makes applications easier to use. Second, each application should be conceptually, linguistically, visually, and functionally consistent within itself and with other applications. Such consistency benefits users. It also benefits designers and developers, who can produce well-designed applications more quickly by reusing standard interface elements.

Occasionally the goals of cross-platform consistency and within-platform consistency may conflict. For example, two platforms may provide different interfaces for accomplishing the same function. In such cases, within-platform consistency should be given priority, because most users only work within one platform.

1.1.4 Clarity

An application interface should be visually, conceptually, and linguistically clear. Visual elements should be immediately comprehensible, ideally because they relate to real-world analogues, and should be arranged so that their functions are comprehensible. Conceptual metaphors should be simple and realistic. Interface text should be clear, unambiguous, and free of jargon.

1.1.5 Aesthetics

Both aesthetic appeal and visual clarity can be substantially enhanced by attention to basic graphic design principles concerning spatial grouping, contrast, and three-dimensional representation. The best interfaces combine powerful yet accessible functionality with a pleasing appearance.

1.1.6 Feedback

The user should receive immediate and tangible feedback for actions within an application. For example, when a person picks up a pencil, tactile and visual sensations provide feedback that the pencil has been touched. Similarly, if the user of an application selects a data object with the mouse, the application should provide visual feedback that the object has been selected. Graphical feedback is particularly effective, but textual and auditory feedback are also useful (see Chapter 3, section 3.6).

1.1.7 Forgiveness

Users like to explore an application and learn by trial and error. Such self-motivated learning can be extremely effective, but users may not always be aware of potential dangers. Even with the best-designed interface, users make mistakes—both physical mistakes (accidentally pointing to the wrong command or data) and mental mistakes (making a wrong decision about which command or data to select). The interface should accommodate user exploration and mistakes without pain or penalty, should minimize the opportunities for errors, and should handle errors gracefully. Error messages should not imply that the user is at fault; instead, they should state the problem objectively and offer possible solutions (see Chapter 3, section 3.6.1.2).

1.1.8 Awareness of Human Strengths and Limitations

By the age of five, we are wonderfully adept at many linguistic and visual tasks that stymie even the most advanced computer systems. Nevertheless, we also have unavoidable limitations in perception, memory, and reasoning. Applications should respect these limitations rather than forcing the user to overcome them. For example, the user should not be required to calculate information (such as the day of the week corresponding to a certain date) that can be provided by the application. Similarly, the user should not be required to recall complex sets of options or commands. Instead, the available choices should be presented explicitly; recognizing items is much easier than recalling them.

1.2 Design Methodology

For maximum effectiveness, design principles must be used in conjunction with a design methodology that puts the user at the heart of the design process, encompasses the broad context within which the application will be used, and leaves room for iterative testing and redesign of the interface.

Even the most creative and experienced designers cannot always produce the right interface design on the first try. Indeed, sometimes experience in user interface design can be a barrier to finding the right solution; to the extent that designers are intimately familiar with an interface, they are removed from the viewpoint of the new or casual user. It is essential to test new designs on real users—not only on colleagues from down the hall. Schedules should include time to redesign the interface in light of usability test results. Time spent in usability testing is time well spent; it is far better to uncover interface problems early in the design stage rather than after a product has been launched.

In designing and testing an interface, it is crucial to keep in mind the larger context within which the application will be used. For example, will it run on a stand-alone computer or as part of a network? Will it be used alone or with other applications? In the increasingly networked and multitasking computer environment of the 1990s, applications that allow for easy exchange of data with other users and other applications will have an enormous advantage over traditional stand-alone applications. The interface should reflect and facilitate this new integration by providing data exchange techniques that are consistent from one application to another.

1.3 Selected Bibliography

Chew, Jane Carrasco, and John Whiteside, eds. *Empowering People: CHI '90 Conference Proceedings.* New York, NY: ACM Press, 1990.

Helander, Martin, ed. *Handbook of Human-Computer Interaction.* Amsterdam: North-Holland, 1988.

IBM Corporation. *Common User Access: Advanced Interface Design Guide.* Boca Raton, FL: IBM, 1989.

Laurel, Brenda, ed. *The Art of Human-Computer Interface Design.* Reading, MA: Addison-Wesley, 1990.

Nielsen, Jakob, ed. *Coordinating User Interfaces for Consistency.* Boston: Academic Press, 1989.

Norman, Donald A. *The Design of Everyday Things.* New York: Basic Books, 1988.

Robertson, S.P., G.M. Olson, and J.S. Olson, eds. *Reaching Through Technology: CHI '91 Conference Proceedings.* Reading, MA: Addison-Wesley, 1991.

Shneiderman, Ben. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Reading, MA: Addison-Wesley, 1987.

Tufte, Edward R. *Envisioning Information.* Cheshire, CT: Graphics Press, 1990.

In addition, ACM/SIGCHI publishes proceedings on computer-human interactions on a regular basis.

Fundamental Input Elements

Chapter 2

The user interface is grounded in a relatively small set of fundamental input elements that define the user's interaction with the computer. The two basic elements are the mouse and the keyboard.

2.1 Mouse Input

The most important mouse operations are pointing, clicking, and dragging. These operations may be combined with modifier keys (SHIFT, CTRL, and ALT).

2.1.1 Basic Operations

- **Button 1.** This is the selection button. Most mouse actions rely on mouse button 1.[1]
- **Button 2.** This button is used to bring up context-specific actions and options. Assigning alternate operations to button 2 may confuse the user.

Additional mouse buttons may be supported but are typically not available. Therefore, a third (or additional) button should be assigned to operations or functions already in the interface. For example, a third mouse button (the middle button) can be used as an additional way to distinguish selection from direct (drag) manipulation.[2]

Table 2.1 lists the basic mouse input elements and gives examples of their use. For information on using modifier keys in conjunction with the mouse, see Chapter 3, section 3.1.2.

[1] By default, button 1 is the left button, but the Microsoft Windows Control Panel allows the user to switch the left/right mapping.

[2] A third mouse button can be simulated with a two-button device by using both buttons or the ALT key+ button 2 combination.

Table 2.1 Mouse Input

Operation	Definition	Common Usage (Using Button 1)
Point	Move pointer (“hot spot*”) to desired screen location.	Navigates; prepares for selection or for operation of control.
Press	Press and hold the button.	Identifies object to be selected.
Click	Press and release button without moving mouse.	Selects insertion point or item; operates control; activates inactive window or control.
Double-click**	Press and release button twice within specified interval, without moving mouse.	Shortcut for common operations, for example, activates icon, opens file, selects word.
Drag	Press button and hold while moving mouse.	Identifies range of objects; moves or resizes items.
Double-drag	Press button twice and hold while moving mouse.	Identifies selection by larger unit (for example, words).

* The hot spot is the position in the mouse pointer that marks the exact screen location that will be affected by a mouse action.

** There are no recommended assignments for triple-clicking or additional multi-clicking operations. Applications may define their own assignments, but remember that these operations are often difficult for users to master.

A mouse action is proposed when the mouse button is pressed down, and confirmed when the mouse button is released. For example, if the pointer is over a menu item, the item is highlighted when the button is pressed down and initiated when the button is released. Similarly, during drag operations the user gets visual feedback (for example, highlighting) while the button is down, but the operations are not accepted until the button is released. While this is the general rule, there may be occasional exceptions. For example, scrolling is initiated as soon as the user presses the mouse button over a scroll arrow; the action auto-repeats as long as the mouse button is down.

2.1.2 Guidelines for Using Mouse Operations

Mouse operations should not require extraordinary hand-eye coordination from the user. For example:

- If an object is so small or thin that pointing or clicking to select it would require extremely precise mouse positioning, provide a “hot zone” around the object to increase the area where clicking will select the object.
- The rapid button-pressing required by double-clicks and double-drags is difficult for some users; never use these input techniques as the only means of carrying out essential operations such as opening files.

- Do not require the user to point at a moving target, except in games.

2.2 Keyboard Input

Keyboard input involves pressing types of keys: text keys, editing keys, mode keys, navigation keys, and shortcut (function) keys.

2.2.1 Text Keys

Text keys can be defined as the alphanumeric (a-z, 0-9), punctuation, symbol, TAB, and ENTER keys, and the SPACEBAR. In applications that have text-entry modes, pressing a text key in text mode causes the corresponding character to appear on the screen. (TAB and ENTER may not be visible except in certain views.) In non-text-entry modes, these keys can be used as shortcuts for other operations, such as choosing tools from a toolbox or selecting an item with a matching first letter from a list. In addition, TAB is used for navigation (see section 2.2.4), ENTER is used to press the default button in dialog boxes, and the SPACEBAR is used as the default Select key for explicit keyboard selection (see Chapter 3, section 3.1.3).

PC keyboards include two keys labeled ENTER: the normal ENTER and the keypad ENTER. These keys have the same label, so their default functions should be the same.

2.2.2 Editing Keys

Table 2.2 lists the editing keys and their functions. Unless otherwise indicated, these functions apply to text editing.

Table 2.2 Editing Keys

Key	Recommended Function
DEL	▪ If there is a selection: Deletes entire selection. ▪ If there is an insertion point and no selection: Deletes character to *right* of insertion point.
BACKSPACE*	▪ If there is a selection: Deletes entire selection. ▪ If there is an insertion point and no selection: Deletes character to *left* of insertion point.
INS	Toggles between Insert mode (new text characters push old ones to right) and Overtype mode (new text characters overwrite old ones).

* Characters deleted by the DEL and BACKSPACE keys are not placed on the clipboard. For additional information, see Chapter 3, section 3.7.

2.2.3 Mode Keys

Mode keys change the actions of the other keys. The two kinds of mode keys are toggle keys and modifier keys.

2.2.3.1 Toggle Keys

A toggle key turns a particular mode on or off each time it is pressed and released. For example, the INS key toggles between Insert mode (in which new text characters push old ones to the right) and Overtype mode (in which new characters overwrite old ones).

Table 2.3 lists the principal toggle keys. Function keys may also be used to toggle modes.

Table 2.3 Toggle Keys[3]

Key	Function
INS	Toggles between Insert mode (new text characters push old one to right) and Overtype mode (new text characters overwrite old ones).
CAPS LOCK	Pressing alphabetic key yields uppercase.
NUM LOCK	Numeric keypad keys yield numbers rather than direction.
SCROLL LOCK	Navigation keys scroll data without moving cursor; existing selections are preserved.
F8	Toggles Extend mode. In this mode, selection behaves as if the SHIFT key is locked down for all direction keys and mouse actions (see Chapter 3, section 3.1.1.5).
SHIFT+F8	Toggles Add mode, which allows disjoint selection through the keyboard. In Add mode, navigation keys move the focus without affecting existing selections, and pressing the SPACEBAR toggles the selection state of an item (see Chapter 3, section 3.1.3.2).

[3] Although these are the recommended assignments, applications need not support all of these keys.

2.2.3.2 Modifier Keys

The modifier keys are SHIFT, CTRL, and ALT. Like toggle keys, modifier keys also change the actions of other keys. Unlike toggle keys, however, the mode established by a modifier key remains in effect only while the key is pressed down; in other words, the mode is "spring-loaded," and the user must actively maintain it. Spring-loaded modes are preferable to self-maintaining modes, because the active maintenance required for a spring-loaded mode prevents the user from forgetting that a mode is in effect. Accordingly, if you need to switch modes from the keyboard, modifier keys are preferable to toggle keys, as long as the required actions within the mode can be quickly and easily accomplished while one hand is occupied by holding down the modifier key.

Table 2.4 lists the most common functions of the modifier keys. For more detailed descriptions, see the sections in Chapter 3 covering the functions mentioned in the table. Typically, the modifier key is pressed at the beginning of the operation and held down during the operation if its release cancels or changes the operation.

Table 2.4 Modifier Keys

Key	Typical Functions
SHIFT	▪ With alphanumeric keys, yields uppercase or the character inscribed on the top half of the key.* ▪ With mouse click or navigation keys, extends or shrinks the contiguous selection range. ▪ With function keys, alters meaning of action (for example, F1 brings up the Help application window, pressing SHIFT+F1 enters Help mode).
CTRL	▪ With mouse click, selects or deselects an item without affecting previous selections. ▪ With alphabetic keys, yields shortcuts. ▪ With navigation keys, typically moves cursor by a larger unit than the unmodified key.
ALT	With alphabetic key, navigates to the menu or control marked with that key as a mnemonic.

* With CAPS LOCK on, yields lowercase characters.

2.2.4 Navigation Keys

The navigation keys are HOME, END, PAGE UP, PAGE DOWN, the four arrow keys (LEFT, RIGHT, UP, and DOWN), and TAB. Table 2.5 lists the functions of these keys, singly and in combination with various modifier keys. The CTRL+key combination is generally used to move by a larger increment than the unmodified key. Navigation operations may also be assigned to keys in addition to those listed here (for example, to function keys).

Table 2.5 Navigation Keys

Key	Unmodified Key Moves Cursor To...	CTRL+Key Moves Cursor To...
HOME	Beginning of line. (Leftmost position in current line.)	Beginning of data. (Top left position in current field or document.*)
END	End of line. (Rightmost position occupied by data in current line.)	End of data. (Bottom right position occupied by data in current field or document.**)
PAGE UP	Screen up. (Previous screen, same horizontal position.)	Screen left/beginning. (Top of window; or, moves left one screen.)
PAGE DOWN	Screen down. (Next screen, same horizontal position.)	Screen right/end. (Bottom of window; or, moves right one screen.)
LEFT ARROW	Left one unit.§	Left one (larger) unit.§§
RIGHT ARROW	Right one unit.§	Right one (larger) unit.§§
UP ARROW	Up one unit/line.#	Up one (larger) unit.##
DOWN ARROW	Down one unit/line.#	Down one (larger) unit.##
TAB	Dialogs: Next field; may move left to right or top to bottom at designer's discretion; after last field, wraps to first. (SHIFT+TAB moves in the reverse order.)	(Not defined.)

* If there is no left dimension, the key combination may also be used to move to the top position.

** If there is no right dimension, the key combination may also be used to move to the bottom position.

§ For text, this moves between characters.

§§ For text, this is generally used to move between words (to the beginning of the next or previous word). Other usage may also be appropriate, as long as applications follow the general principle that CTRL+navigation key moves by a larger unit than the unmodified key.

# Generally maintaining the same position.

## For text, this is generally used to move between paragraphs. Other usage may also be appropriate, as long as applications follow the general principle that CTRL+navigation key moves by a larger unit than the unmodified key.

2.2.5 Shortcut Keys

Shortcut keys or key combinations can be used to provide more rapid access to frequently performed operations. Function keys and CTRL+letter combinations are often used as shortcuts. You may also use ALT+function key combinations, with the exceptions noted in section 2.2.5.3. Note that ALT+letter combinations are not recommended as shortcut keys because they provide standard keyboard access to menus and controls.

2.2.5.1 Function Key Shortcuts

Table 2.6 lists recommended PC function key assignments. Function keys that do not have recommended assignments are available for use by applications.

Table 2.6 Recommended PC Function Key Assignments

	(No modifier)	SHIFT	CTRL	ALT
F1	Help (Chapter 11, section 11.3.1).	Enter Help mode (Chapter 11, section 11.3.1).	(No recommended assignment.)	(No recommended assignment.)
F2, F3	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)
F4	(No recommended assignment.)	(No recommended assignment.)	Close document window (Chapter 5, section 5.4.1).	Close application window (Chapter 5, section 5.4.1).
F5	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)
F6*	Move clockwise to next pane of active window (Chapter 4, section 4.3.5.2.2).	Move counterclockwise to next pane of active window (Chapter 4, section 4.3.5.2.2).	Move to next document window; top window moves to bottom of stack (Chapter 4, section 4.3.5.2.3). (Adding SHIFT reverses action: previous window moves to top.)	Move to application's next open non-document window (Chapter 4, section 4.3.5.2.3). (Adding SHIFT reverses order of movement.)
F7	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)
F8	Toggle Extend mode, if supported (Chapter 3, section 3.1.1.5).	Toggle Add mode, if supported (Chapter 3, section 3.1.3.2).	(No recommended assignment.)	(No recommended assignment.)
F9	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)
F10	Toggle menu bar activation. (Supported for CUA 2.0 compatibility.)	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)
F11, F12	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)	(No recommended assignment.)

* In addition to the keys discussed here, applications may define their own specialized navigation keys.

If some target users of an application are unlikely to have keyboards that provide function keys, the application should avoid using function keys as the only means of performing essential operations.

2.2.5.2 Control Key Shortcuts

CTRL+letter combinations may also be used as keyboard shortcuts. Table 2.7 lists recommended CTRL+letter shortcuts that are equivalent to Edit menu commands. If an application does not provide the function corresponding to a particular recommended key combination, the application should avoid assigning that key combination to any other function.

Table 2.7 Recommended CTRL+Letter Shortcuts[4]

Key	Function
CTRL+Z	Undo
CTRL+X	Cut
CTRL+C	Copy
CTRL+V	Paste

Table 2.8 lists additional suggested shortcuts.

Table 2.8 Suggested CTRL+Letter Shortcuts

Key	Function
CTRL+N	New
CTRL+O	Open
CTRL+P	Print
CTRL+S	Save
CTRL+B	Bold*
CTRL+I	Italic*
CTRL+U	Underline*

* These shortcuts are suggested for text-formatting applications, in the context for which they make sense. Applications may use other modifiers for these operations.

[4] The shortcuts for Undo, Cut, Copy, and Paste are new in Windows version 3.1. For backward compatibility, we recommend that applications designed to run under Windows version 3.0 also support the old shortcuts: Undo = ALT+BACKSPACE, Cut = SHIFT+DEL, Copy = CTRL+INS, and Paste = SHIFT+INS. However, the old shortcuts should not be documented in user manuals or listed on the Edit menu.

2.2.5.3 Guidelines for Assigning Shortcut Keys

When selecting function keys or key combinations as shortcuts for actions in an application, designers should observe the following guidelines:

- Assign single keys for frequently performed, small-scale tasks. For example, if the application contains a command to split a window, a single function key (such as F6) can be used to move from one pane to another.
- Assign SHIFT+key combinations for actions that extend or are complementary to the actions of the key or key combination used without the SHIFT key. For example, if F6 moves from one pane to another in a clockwise direction, SHIFT+F6 moves through the panes in a counterclockwise direction.
- Assign CTRL+key combinations for infrequent actions, or for tasks that represent larger-scale versions of the tasks assigned to the unmodified key. For example, in text, LEFT ARROW moves left by one character, whereas CTRL+LEFT ARROW moves left to the beginning of the current word (see Table 2.5). Similarly, F6 moves clockwise from one window pane to the next, whereas CTRL+F6 moves clockwise from one document window to the next (see Table 2.6). Following the guideline for SHIFT listed above, SHIFT+CTRL+F6 moves counterclockwise through the document windows.
- Avoid assigning ALT+letter combinations; they are used as mnemonic access characters for menus and dialog box controls. You may use ALT+function key combinations, except for ALT+F4 and ALT+F6, which have the recommended assignments listed in Table 2.6. In addition, the ALT+TAB, ALT+ESC, and ALT+SPACEBAR assignments are reserved for system use.
- Function keys should not be the first choice for shortcuts. When deciding between a function key shortcut (with or without a modifier key) and a modifier+letter shortcut, remember that function key shortcuts are easier to localize but may be harder to remember because they are mapped to functions arbitrarily. Modifier+letter shortcuts offer more mnemonic possibilities and are easier to reach on the keyboard. Applications should use function key shortcuts in addition to modifier+letter shortcuts, or for operations that do not have good mnemonics.

2.2.6 Escape Key

The ESC key is generally used to terminate a function in process or to cancel a direct manipulation operation. It is also used to press the Cancel button in a dialog box. Thus, ESC is in some sense the opposite of ENTER, which presses the default button.[5] Applications may also allow the user to press ESC to cancel or to interrupt lengthy operations, such as printing.

[5] If Cancel is the default button (has a bold border), the ENTER key is used to press it.

General Techniques

Chapter 3

Just as the visual components of the interface should be consistent from application to application, the techniques employed by the user to interact with interface components should also remain consistent. If visual consistency is reinforced with procedural consistency, users can develop habits that transfer from one application to another.

In general, mouse and keyboard access to an application should be parallel. However, techniques should be optimized for the input device. Well-tailored techniques should prevail over strict parallelism between mouse and keyboard techniques.

3.1 Selection

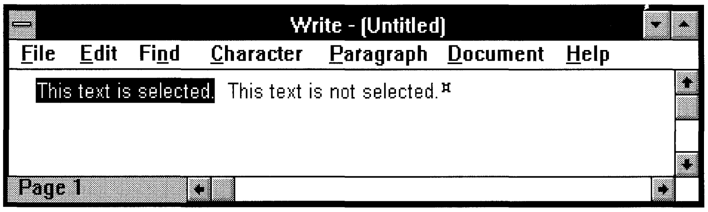
For the object-action paradigm (see Chapter 1, section 1.1.2) to work properly, the user must be able to select the object to which an action will apply. Applications therefore need to provide a means for the user to select data.

3.1.1 Concepts of Selection

Many selection concepts—highlighting, types of selection, adjusting selections, deselection, and selection modes—apply to both mouse and keyboard selection.

3.1.1.1 Indicating Selections: Highlighting

When data is selected, it appears highlighted. The appearance of highlighted text depends on the abilities of the system and the display. On monochrome displays, reverse video (XOR) should be used to indicate selected data (see Figure 3.1). On gray-scale displays, the selection should be marked with a shade of gray. On color displays, a highlight color should be used. Graphics may be highlighted in these same ways or by the addition of rectangles with resizing handles; these two methods may also be combined.



This text is selected. This text is not selected.

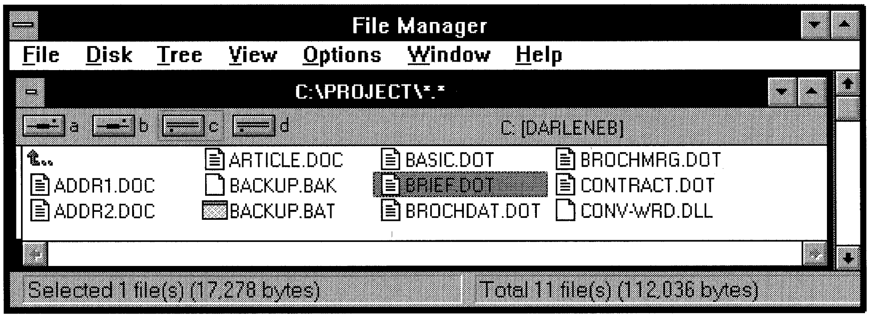
Figure 3.1 Indicating Selection by Reverse Video

Text in dialog boxes should be highlighted with the same methods used in data windows. For toolboxes and other collections of 3-D buttons, the depressed-button graphic should be used to indicate selection. For value set controls that do not contain 3-D buttons, an outline frame in the highlight color should surround the chosen value.

As a general guideline, selection highlighting should be visible only while the window is active. Leaving the selection highlighted in an inactive window is acceptable in special cases, when it is useful to the user. For example, if the user chooses a command with a dialog box that affects the selection, the selection can remain highlighted while the dialog box is open.

3.1.1.2 Types of Selection

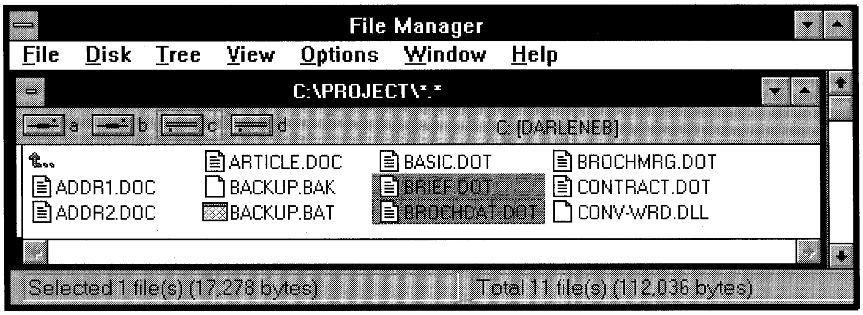
Selections may be classified as single (involving only one item) or multiple (involving several items). Multiple selections may be further classified as contiguous or disjoint, depending on their spatial ordering. These selection types are illustrated in Figures 3.2, 3.3, and 3.4.



```
..              ARTICLE.DOC   BASIC.DOT      BROCHMRG.DOT
ADDR1.DOC       BACKUP.BAK    BRIEF.DOT      CONTRACT.DOT
ADDR2.DOC       BACKUP.BAT    BROCHDAT.DOT   CONV-WRD.DLL

Selected 1 file(s) (17,278 bytes)        Total 11 file(s) (112,036 bytes)
```

Figure 3.2 Single Selection



```
..              ARTICLE.DOC   BASIC.DOT      BROCHMRG.DOT
ADDR1.DOC       BACKUP.BAK    BRIEF.DOT      CONTRACT.DOT
ADDR2.DOC       BACKUP.BAT    BROCHDAT.DOT   CONV-WRD.DLL

Selected 1 file(s) (17,278 bytes)        Total 11 file(s) (112,036 bytes)
```

Figure 3.3 Contiguous Multiple Selection

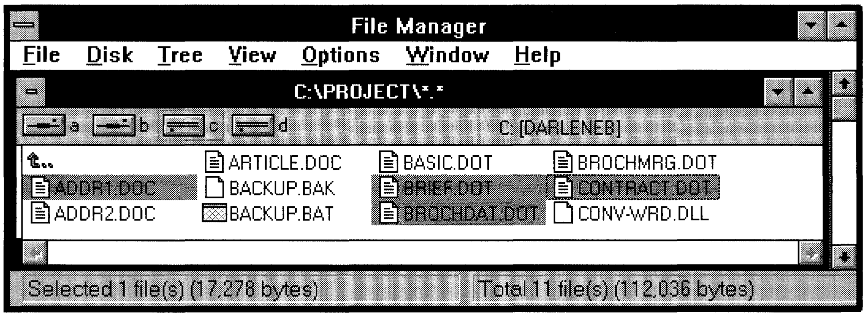
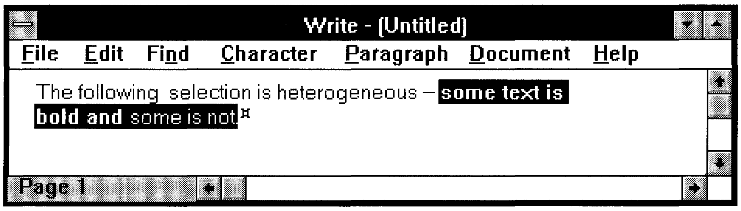


Figure 3.4 Disjoint Multiple Selection

Multiple selections may also be classified as homogeneous or heterogeneous, depending on the properties of the selected items. For example, if a text selection contains both plain and bold text, the selection is heterogeneous with respect to character attributes (see Figure 3.5). The user makes homogeneous and heterogeneous selections in the same way. In some cases, however, controls that indicate property settings for a selection should reflect the difference between the two types (see Chapter 6, sections 6.1.2 and 6.2).

Mouse and keyboard techniques for single and multiple selection are explained in sections 3.1.2 and 3.1.3.



The following selection is heterogeneous – **some text is bold and** some is not

Figure 3.5 Heterogeneous Selection

3.1.1.3 Adjusting Selections

Adjusting a selection means changing its size relative to its original starting point, or anchor point. The opposite end of the selection is the active end of the selection. To extend the selection, the user moves the active end away from the anchor point. To shorten the selection, the user moves the active end toward the anchor point. When the active end reaches the anchor point, the selection is an insertion point. From the insertion point, the user can extend the selection in either direction, but not in both directions at the same time.

3.1.1.4 Deselection

To deselect data, the user generally clicks to make a new selection. Deselecting does not delete the data. Deselection is overridden when the user applies disjoint selection techniques. Multiple-selection list boxes (see Chapter 6, section 6.3.2) also override this default deselection behavior.

In multiple (contiguous or disjoint) selections, items can be deselected individually or as a group. To deselect a single item with the mouse, the user uses CTRL+click.[1] This combination toggles the selection state of the clicked item while preserving the selection state of all other items. For example, if an item is selected, CTRL+click deselects it without affecting other items. If the item is not selected, CTRL+click selects it. To deselect all currently selected items, the user clicks to make a new selection.

3.1.1.5 Selection Modes

Selection techniques that set a special selection mode are generally not recommended, except in special contexts. A selection mode should only supplement standard selection techniques. If selection modes are used, a visual cue should indicate that the mode is active. For example, if the mode affects a data window, a mode indicator should be displayed in the status bar (if there is one).

Extend mode is an example of a selection mode. In Extend mode, selection behaves as if the SHIFT key is locked down for all direction keys and mouse actions. The key for toggling Extend mode is F8.

3.1.2 Mouse Selection

Mouse selection relies on the basic techniques of clicking and dragging. In general, clicking selects a single item or location, and dragging selects a range consisting of all items between the button-down and button-up locations. If dragging is already used by the application for item movement, outline selection (see section 3.1.2.4) may be used to select a range of items.

[1] See section 3.1.3 for keyboard deselection techniques.

3.1.2.1 Mouse Selection Techniques for Item-Oriented Applications

In most item-oriented applications, selections made with the mouse can be contiguous (range selection) or disjoint (selected items are separated by nonselected items). The user clicks mouse button 1 without a modifier to make contiguous selections. Disjoint selection requires the use of the CTRL key with mouse button 1. Selections can be adjusted by using the SHIFT key with mouse button 1.[2]

Table 3.1 describes how these techniques work.

Table 3.1 Mouse Selection

Action	Clicked Item Is...	Resulting Selection =	Focus Moves To...	Anchor Moves To...
Click	Selected	Clicked item	Clicked item	Clicked item (anchor = focus)
SHIFT+click	Set to selection state of anchor item (which currently may or may not be selected)	Extend anchor item's selection state to clicked item (and all items in between). (If the anchor was not selected, SHIFT+click selects only the current item.)	Clicked item	No movement (anchor remains where it was)
CTRL+click	Toggled	Existing selection +/- clicked item	Clicked item	Clicked item (anchor = focus)
SHIFT+CTRL +click	(Same as SHIFT+click)	Existing selection +/- new range, as defined for SHIFT+click	Clicked item	(Same as SHIFT+click)

[2] Graphic objects in drawing applications have no well-defined ordering, so selection adjustment is not defined for such objects. In these applications, SHIFT+click may be defined as equivalent to CTRL+click (= disjoint selection).

3.1.2.2 Mouse Selection Techniques for Text-Oriented Applications

Table 3.2 shows the recommended techniques for contiguous text selection with the mouse.

Table 3.2 Mouse Selection in Text

Action	Resulting Selection=	Focus Moves To...	Anchor Moves To...
Click or drag	All text from button-down to button-up (for a click, this is just an insertion point)	Beginning of selection	Button-down
SHIFT+click or SHIFT+drag	All text between anchor and button-up location*	Beginning of selection	No movement
CTRL+click or CTRL+drag	Entire sentences from button-down to button-up	Beginning of first selected sentence	Beginning of selection**
Double-click or Double-drag	Entire words from button-down to button-up	Beginning of first selected word	Beginning of selection**

* Optional enhancement: If the original selection unit was a word (selected by double-click or double-drag) or sentence (selected by CTRL+click or CTRL+drag), the selection can be extended beyond the button-up location to the end of the unit.

** In some applications, the anchor point encompasses the entire selection; a subsequent SHIFT+click extends the selection from the most distant end. The Windows Write program puts the anchor point at the beginning of the selection.

3.1.2.3 Margin Selection in Text and Arrays

Margin selection is a convenient way to select large sections of data with a single click. In text, margin selection is used to select lines, paragraphs, or entire documents; in data arrays, it is used to select rows and columns. The margin selection area can be the row and column labels of a data array or the left margin area between the left window frame and the left edge of text.

Table 3.3 shows the actions that should be assigned to the mouse for margin selection in text and in data arrays.

Table 3.3 Margin Selection in Text and in Data Arrays*

Action in Margin	Resulting Selection=	Focus Moves To...	Anchor Moves To...
Click or drag	Entire lines, rows, or columns next to pointer from button-down to button-up	Beginning of selection	Beginning of selection
SHIFT+click or SHIFT+drag	Entire lines, rows, or columns between anchor and button-up location	Beginning of selection	No movement
CTRL+click or CTRL+drag			
in text documents	Entire document	Beginning of document	Encompasses entire document
in data arrays	Previous selections + entire rows or columns next to pointer from button-down to button-up	Beginning of new selection	Beginning of new selection
Double-click or Double-drag			
in text documents	Entire paragraphs from button-down to button-up	Beginning of first selected paragraph	Beginning of first selected paragraph**
in data arrays	(Same as click or drag)	(Same as click or drag)	(Same as click or drag)

* In this table, for text documents, the beginning of a selection is defined as the part nearest the beginning of the document. For data arrays, the beginning of a selection is defined as the cell nearest the button-down point.

** In some applications, the anchor point encompasses the entire first selected paragraph; a subsequent SHIFT+click extends the selection from the most distant end of that paragraph. Windows Write puts the anchor point at the beginning of the first selected paragraph.

3.1.2.4 Outline Selection of Graphical Objects

Outline selection is an extended form of drag selection that is particularly useful for graphical objects when normal drag selection conflicts with moving objects with the mouse. This technique allows the user to drag an outline frame (rectangular or free-form) around an object, a set of objects, or a portion of an object. When the mouse button is released, existing selections are removed, and all objects falling completely within the frame (including the frame itself) are selected. When outline selection is used with bitmaps rather than with graphical objects, only the bits falling within the frame are selected.

3.1.3 Keyboard Selection

Keyboard selection uses a selection cursor (such as a dotted outline box or an insertion point) and a form of selection emphasis (see section 3.1.1.1) to indicate the data that will be affected by any action the user initiates. There are two types of keyboard selection techniques: implicit and explicit.

- In implicit selection, the item under the selection cursor becomes selected automatically when the selection cursor moves; thus, the selection cursor and emphasis travel together. In general, implicit selection is associated with contiguous selection.
- In explicit selection, the selection cursor moves independently of the selection emphasis. That is, the item under the selection cursor is selected and given the selection emphasis only when the user explicitly selects it with the Select key.[3]

The Select key may also be used to deselect selected items; in other words, it toggles selection states. In general, explicit selection/deselection is associated with disjoint selection, in which selected data can be separated by nonselected items, and a new selection can be added to existing selections at any point in the data.

3.1.3.1 Keyboard Techniques for Contiguous Selection

In text-oriented applications, the user can select a single insertion point with the keyboard by simply navigating to the desired location. The anchor point is set at the new location. A range of characters is selected by using the SHIFT key in conjunction with navigation keys. While holding down the SHIFT key, the user can press any navigation key (or any combination of keys defined for navigation, such as CTRL+END; see section 3.3 for details). The cursor then moves to the location implied by the navigation key, and all characters between the anchor point and the destination are selected. The anchor point does not move.

The SHIFT+navigation technique works similarly for item-based applications, with one difference: Whereas in text, all characters between the anchor point and the destination are selected, in item-based applications, the fate of the items between the anchor and the destination depends on the selection state of the anchor item. If the anchor item is currently selected, all items in the range become selected; if the anchor item is not currently selected, all items in the range become deselected.

[3] By default, the Select key is the SPACEBAR, unless this assignment conflicts with the needs of an application in a specific context.

3.1.3.2 Keyboard Techniques for Disjoint Selection

The keyboard technique for making disjoint selections relies on using SHIFT+F8 to enter Add mode. In this mode navigation keys move the focus without affecting existing selections or the current anchor point. Pressing the SPACEBAR toggles the selection state of the item at the new location and sets a new anchor point there. SHIFT+navigation (while still in Add mode) can then be used in the usual way to extend the selection from the new anchor point, without disturbing previous selections. Pressing SHIFT+F8 again toggles the application out of Add mode.

Disjoint selection may also be permitted in list boxes, to let the user select multiple items from the list (see Chapter 6, section 6.3.2).

3.2 Focus

The "focus" represents the part of the interface that will receive the next piece of input if no navigation occurs before the input is generated. For mouse input, the focus always coincides with the pointer (button down) location. For keyboard input, the focus depends on the context:

- In text, the focus is shown by an insertion point, which indicates where newly typed characters will be inserted.
- In spreadsheets, the focus is shown by a highlighted cell border, which marks the cell that will receive typed input and that will be affected by commands.
- In menus and dialog box controls, the focus is shown by an active control indicator (described below), which identifies the control that will be affected by the Select key or by typing.

The active control indicator can be an insertion point, highlighting, or a dotted box:

- The insertion point is used for text boxes.
- Text highlighting is used in conjunction with the insertion point when the text box is initially activated (for example, with ALT+mnemonic). If the user then clicks the mouse over the text, the highlighting disappears, and the insertion point is placed at the click location.
- The dotted box is used for option buttons, check boxes, and command buttons; the box should enclose the label of the control (see Figure 3.6). The dotted box is also used for list boxes. In single-selection lists, the box should enclose the list item that is currently selected or that will be selected if the user presses the Select key. In multiple-selection lists, the box should enclose the item whose selection state will be toggled by pressing the Select key. In extended-selection lists, the box should enclose the most recently selected item, unless the list is in Add mode. In Add mode, the box should enclose the item whose selection state will be toggled by pressing the Select key. (For further information on these types of list boxes, see Chapter 6, section 6.3.)

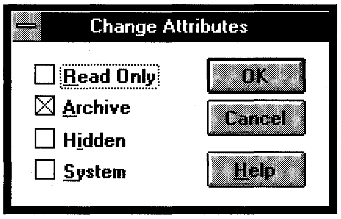


Figure 3.6 Use of Dotted Box to Indicate Active Control

When a window is reactivated, the focus and the selection should be displayed in the same locations as when the window was last active.

3.3 Navigation

The user can move the input focus by navigating on the screen.

3.3.1 Mouse Navigation

Navigation with the mouse is simple: When the mouse is moved left or right on the user's desktop, the pointer moves left or right on the screen; when the mouse is moved away from or toward the user, the pointer moves up or down. By moving the mouse, the user can move the pointer to any location on the screen. Mouse navigation typically does not affect existing selections.

3.3.2 Keyboard Navigation

Keyboard navigation is more complicated than mouse navigation. Several keys, such as HOME, END, and the arrow keys, are dedicated to keyboard navigation with respect to data items, as discussed in Chapter 2, section 2.2.4. Unlike mouse navigation, keyboard navigation changes the selection, unless Scroll Lock mode is in effect (see Chapter 2, section 2.2.3).

Keyboard navigation to controls relies primarily on mnemonic access characters and on the TAB, ENTER, and ESC keys. With mnemonic access, when the user presses ALT plus the mnemonic letter in a control label, the focus moves to the control, and the control is selected or operated. For example, "F" is the mnemonic assigned to the File menu, so ALT+F[4] selects and opens that menu.

[4] ALT,mnemonic (for example, ALT,F) is also supported.

The ALT+mnemonic method can also be used to navigate among controls in dialog boxes. If the control that currently has the focus does not capture character input (that is, if the control is not a text box, spin box, list box, or combo box[5]), mnemonics can be used without ALT. For example, the user can select the No command button in a dialog box by pressing ALT+N or just N. If the user presses the mnemonic for a dimmed control, the focus remains unchanged, and no action is taken on the control.

When the ALT+mnemonic method is used to reach and operate a control, the focus usually remains on the control after the operation (unless the operation closes the dialog). However, if the focus is on a list box and the user uses ALT+mnemonic to reach and operate a button associated with the list, the focus should return to the list after the button is pressed. For example, if the user adds or deletes individual items from a list box by selecting an item, then pressing the Add or Delete command button, the focus should return to the list after each Add or Delete operation so the user can select another item.

Dialog boxes also permit the use of the TAB, ENTER, and ESC keys for navigation:

- TAB moves the active control indicator among the available controls without operating them (the SPACEBAR can then be used to operate the control that currently has the focus).
- The ENTER key chooses the default command button.[6]
- The ESC key chooses the Cancel or Close command button if the dialog contains one of these buttons.

Table 3.4 lists recommended keyboard navigation techniques.

[5] See Chapter 6 for descriptions of these dialog box controls.

[6] The OK command button is typically the default, but a different button may be assigned instead if it is a more likely choice for the dialog. Users can also change the default button through keyboard navigation.

Table 3.4 Keyboard Access to Controls

Key	Action
ALT+mnemonic	Navigates to and selects or operates control. The mnemonic letter can be pressed without ALT if the current focus does not capture character input.
TAB	Moves focus to next control. The order of movement is generally from left to right and from top to bottom.*
SHIFT+TAB	Moves focus to preceding control. The order of movement is generally from right to left and from bottom to top (that is, the reverse of TAB).*
SPACEBAR	▪ Selects or operates a command button, option button, or check box that has the focus (equivalent to a mouse click on the control). ▪ In text box, combo box, or spin box, inserts a space character.
Arrow keys	▪ In text box, moves insertion point left and right. ▪ In group of option buttons, selects next button, wraps around at top and bottom of group. ▪ In group of check boxes, moves to next box without changing its state, wraps around at top and bottom of group. ▪ In list box, selects next item; stops at top and bottom of list. ▪ In spin box, increases or decreases value; wraps around at highest and lowest values.
DOWN ARROW	Opens a closed drop-down list box or drop-down combo box that currently has the focus.** If the drop-down control is already open, DOWN ARROW navigates within the list (see Chapter 6, section 6.3.1.3).
ALT+DOWN ARROW	Toggles state (collapsed or expanded) of active drop-down list box or drop-down combo box.***
ALT+UP ARROW	Toggles state (collapsed or expanded) of active drop-down list box or drop-down combo box.***
Alphanumeric keys	▪ As a mnemonic, see "ALT+mnemonic" at the beginning of this table. ▪ In list box, selects next item beginning with the character. ▪ If typed as the first character in the text field of a combo box, scrolls list to the first item beginning with the character.
ENTER	Presses default command button, if one is implemented. Otherwise, presses selected command button.
ESC	Presses the Cancel or Close button, if one is implemented.

* Unless there is a more logical order defined within the context of the operation.

** This is a new recommendation. Windows version 3.1 includes a style bit to enable the new behavior.

*** The use of F4 to open and close drop-down controls is no longer recommended.

3.3.3 Keyboard Access to Control Bars

ALT+mnemonic access can also be used in control bars (for example, ribbons and rulers; see Chapter 4, section 4.2.8), for labeled non-button controls, and for textual buttons (that is, buttons labeled with ordinary text rather than graphics).

- Purely graphical buttons (for example, for drawing tools) do not require mnemonic access. Keyboard access to these buttons is defined by the application.
- If the button label contains an underlined letter, the ALT+mnemonic method must be implemented. (In addition, applications may define CTRL+letter shortcuts.)
- Formatting buttons in ribbons and rulers often contain graphical text and thus are neither purely textual nor purely graphical. These buttons usually represent commands that are available through shortcuts, even when the control bar is not visible. These shortcuts typically use CTRL+letter (see Chapter 2, section 2.2.5.2). If the control bar happens to be visible, the command invoked by the shortcut also depresses the button. In a sense, then, the button is accessed by CTRL+letter. This behavior is acceptable; it is not necessary to add ALT+mnemonic as an additional access method.

Toolboxes are primarily a mouse-oriented feature, so keyboard access to toolboxes is not crucial and need not be implemented. For advanced users, however, keyboard access can be useful. Experienced graphics designers use a two-handed mouse/keyboard technique for operating the tools without interruption: the left hand presses keys to switch between tools while the right hand remains on the mouse to use the current tool. For example, the right hand draws a rectangle with the rectangle tool, the left hand presses a key to switch to the paint bucket tool, and the right hand fills the rectangle. Keyboard access to toolboxes can also be used to facilitate automated software testing.

The following methods are recommended for applications that choose to implement keyboard access to toolboxes:

- If the user will not typically access tools when the focus is in a text field, assign single-letter mnemonics to tools (for example, T for a text tool, R for a rectangle tool). Pressing the letter chooses the corresponding tool and depresses its button. The focus need not be in the toolbox when the letter is pressed. This method provides the most rapid access.
- If the user will need to access tools when the focus is in a text field, use CTRL+letter mnemonics instead of unmodified letters.

- The preceding methods provide rapid, direct access to tools, but they are somewhat obscure, because the mnemonic letters are not visible anywhere and so must be memorized. The following method provides a more standard way of accessing tools through the keyboard; it requires knowledge of standard Windows techniques for switching between non-document windows or between panes:
  - If the toolbox is a separate palette window, use the standard method of switching between non-document windows (ALT+F6) to activate the window.
  - If the toolbox is a pane (or a fixed child window) within the main application window (for example, as in Windows Paintbrush), use the standard method of switching between panes (F6 or SHIFT+F6) to move the focus to the pane. (TAB may also be used, as in the File Manager.)
  - Once the toolbox is active, use the arrow keys to choose a tool. When the arrow keys are used to move from one tool button to another, the new button is depressed and the old button returns to the up position.

3.4 Transfer Interface

The current interface for transferring objects and data from one location to another relies on a combination of techniques, such as direct manipulation (see the next section) and Edit menu commands (for example, Cut, Copy, Paste, and Paste Link; see Chapter 5, section 5.4.3, and Chapter 9, section 9.3.2).

3.5 Direct Manipulation

Direct manipulation is frequently used for moving an object from one location to another: The object is dragged with the mouse and dropped into the new location. For example, some file management programs use drag-and-drop to move files between directories on the same disk. Similarly, many word-processing applications use drag-and-drop to move tab stops on rulers.

Drag-and-drop can also be interpreted as a Copy or Link operation, if the context makes those operations more appropriate than a Move. For example, in the File Manager, dragging a file icon to a directory on a different disk causes the file to be copied rather than moved. Dropping a file icon from the File Manager into the Program Manager creates a link in the Program Manager. When more than one interpretation of drag-and-drop is possible for a given context, the application should provide a way to override the default interpretation by using modifier keys in conjunction with dragging.

In some cases, drag-and-drop is interpreted as "use the drop target to process (for example, open or print) the thing that was dropped." For example, dropping a document icon on the minimized Print Manager means "print the document."[7] In the File Manager, dropping a document icon on the associated application icon means "open the document for editing within the application." Dropping a document icon onto any non-document area of an open application window—for example, onto the title bar, menu bar, status bar, application workspace in a multiple-document interface (MDI) application, and so on—has the same meaning. Suppose, however, that the document icon is dropped into the document area of an open application window (for example, the text area of a mail memo or the drawing area of a paint program); in other words, the document icon is dropped into an open document. In this case, if the open document supports object embedding (see Chapter 9), the dropped document is copied (embedded) into the open document and displayed as an icon.[8]

Direct manipulation is particularly useful in pen-based systems, because manipulation of objects with the pen is even more direct than manipulation with the mouse. For a discussion of direct manipulation in pen applications, see Chapter 10, section 10.2.1.2.

[7] This feature is new in Windows version 3.1.

[8] What actually happens is that the file represented by the document is encapsulated in a "package"; this package is then embedded into the open document. To the user, however, the iconic document simply appears to have been embedded into the open document. For more information on packages, see Chapter 9.

Table 3.5 summarizes current recommendations for drag-and-drop operations with unmodified mouse button 1, originating in Windows version 3.1 File Manager.

Table 3.5 Unmodified Button 1 Drag/Drop Operations Originating in the File Manager

Dragged Object	Drop Target	Result
Document icon (representing, for example, a text document, spreadsheet, or drawing)	Open document	If target document supports embedding, embed dropped document and display it as an icon. Otherwise do nothing.
	Non-document area of open application window (for example, title bar, menu bar, status bar, MDI application workspace)	If dropped document is readable by application, open document within application window.
	Print Manager	Print document: 1. Launch associated application and bring window to top. 2. Load file. 3. Display dialog if Print command within application would normally lead to dialog; user can change settings if necessary. 4. If user presses Cancel, close application immediately without printing; if user presses OK, print file and then close application.
Any file icon (representing, for example, a document or application)	Application icon in File Manager	Start application, using dropped document as initial file. For example, if text document is dropped on word-processing application, open document for editing.
	Program Manager group window (open or closed)	Create program item in Program Manager that points to the file.

3.5.1 Differentiating Selection from Direct Manipulation

There are a variety of techniques for differentiating selection of an object from direct (drag) manipulation. It is difficult to come up with universal conventions that can be applied to all types of selected objects. In addition, it is common practice to use the left mouse button (button 1) to do this. This means that selection and drag manipulation may overlap. To distinguish these operations, it is recommended that a drag handle appropriate to the object be displayed when the object is selected. This can be a frame around the object or on the object. In many cases, it can simply be the object itself (for example, in icons). Some form of visual feedback (that is, a change in the pointer or in the object) should be provided on the button-down transition as a clue to the user that the operation will not result in selection, but instead in some form of direct manipulation. This visual feedback should be maintained during the operation, until the button-up (release) transition.[9]

Objects are not limited to displaying their drag manipulation handles when selected; it depends on the object. For example, window title bars and size borders act as manipulation handles that are always present. However, for many objects in context, the drag handle should only be displayed when the object is selected to avoid visual clutter.

3.6 Providing Feedback

Applications should keep the user informed about the current state of the application by providing feedback. Feedback can inform the user that a particular mode has been entered, acknowledge a command, point out an error, track the progress of an operation, and so on. Visual (either graphical or textual) feedback is the most common, but auditory feedback is also useful.

3.6.1 Visual Feedback

The user interface is primarily visual, therefore visual feedback is particularly effective. This type of feedback can consist of graphics, text, or both.

3.6.1.1 Graphical Feedback

Many of the interface techniques discussed previously in this chapter involve graphical feedback. Text selection, for example, is confirmed by the appearance of a highlight color; shifts of the focus are shown by movements of a dotted box, selection highlight, or insertion point. Another useful type of graphical feedback involves changing the pointer to reflect the current state of the application.

[9] This does not preclude changing the pointer further during the operation, if the nature of the destination or pressing a modifier key might change the operation while the button is down.

3.6.1.1.1 Pointers The mouse is linked with a graphic on the screen called the pointer. By positioning the pointer and clicking the buttons on the mouse, the user selects data, icons, commands, and controls to initiate and complete actions. The shape of the pointer changes according to the current action or the current pointer position on the screen. Applications should use only as many pointer shapes as needed to inform the user about current status and position; too many shapes can confuse users.

Tables 3.6 through 3.9 list the suggested pointer shapes and their uses. Applications may supplement these pointers with modifier keys or add visual effects, such as animation. Animation is an effective technique for drawing the user's attention and conveying information, but should be used with caution. Remember that the main function of a pointer is to communicate information, so the movement of the pointer should not be distracting.

Table 3.6 Suggested Selection Pointers

Shape	Screen Location	Selects...
↖	Over items or controls	Items and controls*
↗	Left margin of document or cell	Lines, rows, cells
I (I-beam)	Text	Insertion point or characters
⬇	Top of table column	Column

* Also used for resizing and moving objects; see Tables 3.7 and 3.8.

Table 3.7 Suggested Pointers for Resizing

Shape	Screen Location	Resizes...
↖	On resize handles	Graphics*
↔	Along column gridlines	Column width
↕	Along row headings	Row height
⇳	Top or bottom window border	Window vertically
⬄	Left or right window border	Window horizontally
⤢	Lower left or upper right window border	Window diagonally
⤡	Upper left or lower right window border	Window diagonally

* Also used for selecting and moving objects; see Tables 3.6 and 3.8.

Table 3.8 Suggested Movement Pointers

Shape	Screen Location	Movement
↖	On item	Unconstrained*
↕	On item	Vertical
↔	On item	Horizontal
✥	On item	Vertical or horizontal

* Also used for selecting and resizing objects; see Tables 3.6 and 3.7.

Table 3.9 Other Suggested Pointers

Shape	Screen Location	Use
⌛	Anywhere	Indicates that a lengthy operation is in progress
↖?	Any region or control	Click activates associated help
🔍	Inside window	Zooms
✥	Inside window	Indicates that direction keys will move or resize window
⊘	An object that can't have other objects dropped on it	Indicates that dropping is not allowed*
⇳	Split box in vertical scroll bar	Splits window horizontally
◂‖▸	Split box in horizontal scroll bar	Splits window vertically

* Typically, the pointer does not change over valid drop targets (unless the nature of the operation changes). For additional information, see section 3.6.1.1.2.

3.6.1.1.2 Feedback for Drag-and-Drop Operations For drag-and-drop operations, applications should provide the following types of visual feedback:

- As the pointer moves, the object, its outline, or some reasonable representation should move along with the pointer.
- The user should get feedback, if possible, over the target area. For example, possible destination locations can be highlighted or otherwise emphasized as the object is dragged over them.
- The pointer should change to the shape shown in Table 3.9 over invalid drop targets.

- As an optional but recommended extension, the pointer can be changed during the drag operation to reflect the operation (for example, Move, Copy, or Link) that will result if the object is dropped in the current location.

3.6.1.1.3 Progress Indicators When an operation that takes more than two or three seconds is in progress and the user cannot continue working in that application until the operation finishes, the application should display the hourglass pointer over the inaccessible window to indicate that the user must wait. If the user moves the mouse pointer to a second, accessible window, the normal pointer for that window should appear. If possible, the first application should let the user access the system or work in another application.

If an operation takes longer than five seconds and prevents access to the window during that time, the application should display a progress indicator in addition to the hourglass pointer.[10] Progress indicators reassure the user that the program is working on the task and indicate how much of the job has been completed. A progress indicator must be dynamic (applications should not use static messages or pictures) and the feedback from the progress indicator must be unmistakable and obvious.

The best progress indicators are graphical. For example, progress can be represented by a long rectangular bar that is initially empty but is gradually filled with color from left to right as the operation proceeds (see Figure 3.7). Animated cursors can also be used as graphical progress indicators.

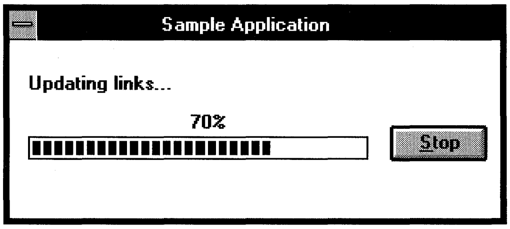


Figure 3.7 Graphical Progress Indicator

If graphical indicators take too long to update, percentage-complete messages can be used as a supplement or replacement. If the extent of the task is not known in advance, or a particular stage in the task takes a long time to complete (thereby freezing the percentage), an elapsed-time message can be used instead of the percentage-complete message. If the task is very long, applications may consider breaking it down into subtasks and using progress indicators for each subtask.

[10]Unless posting and updating the progress indicator takes longer than the process itself.

When possible, progress indicators should include a way of pausing and resuming lengthy operations, in case the performance of the rest of the system is adversely affected. The recommended way to provide pause/resume capability is through a command button whose label alternates between Pause and Resume. If the nature of a particular operation does not permit pause/resume functionality but does allow an irreversible interruption of the operation, the progress indicator should contain a command button labeled Cancel or Stop. The Cancel button interrupts the operation and returns the application and data to its state before the operation was invoked. If a return to that state is not possible, Stop should be used instead of Cancel. The Stop button interrupts the operation but does not reverse any changes that the operation has already caused.

3.6.1.1.4 Flashing for Attention An application can flash for attention when it is inactive but needs to display a message dialog. If the application is a window, it flashes its title bar. If it is minimized, it flashes its icon. When the user activates the application, the message dialog is displayed. The flash rate should be based on the cursor blink rate (which can be adjusted by the user through the Control Panel).

The flashing technique has two advantages. First, it preserves the user's control over the work flow by allowing the response to the message to be postponed. Second, by preventing the sudden display of the message dialog while the user is typing, this technique ensures that the dialog is not accidentally closed by keypresses stored in the type-ahead buffer.

As an additional warning mechanism, the flash is accompanied by one or two beeps.[11] This audible warning also allows applications to attract the user's attention when their title bars or icons are not visible.

3.6.1.2 Textual Feedback

Graphical feedback can be highly effective and can enhance the visual appeal of applications, but sometimes it is not precise enough to convey details. In such cases applications should provide textual feedback in the form of brief messages. Such messages are usually provided in the message bar or status bar at the bottom of the screen (if provided; see Chapter 4, sections 4.2.6 and 4.2.7) or in modal dialog boxes called message dialogs (see Chapter 7, section 7.1.4).

Application designers should observe the following guidelines when writing message text:

- State the information, problem, or error clearly. Avoid technical descriptions or explanations, and use straightforward, easily understood terminology.
- Limit the message to two or three lines. Status bar messages should not exceed one line.

[11]Note that the warning beep can be turned off by the user.

- If possible, suggest a remedy for error situations. For example, if the user enters a measurement that is too large, the error message should not only say "Measurement is too large" but should also ask the user to enter a number within a specified range (for example, "Please enter a number between 1 and 10").
- Avoid phrasing that blames the user or implies user error. For example, use "Cannot find <*file name*>" instead of "File name error." Avoid the word "error" altogether.
- Use "Cannot" instead of "Can not" or "Can't."
- Use "Not enough _______ to ________" (for example, "Not enough space on drive C: to save file SAMPLE.TXT") instead of messages such as "Disk is full, save not completed," "Insufficient memory," or "Low on memory."
- Do not use colons in a message. For example, use "Cannot read <*file name*>" instead of "Cannot read: <*file name*>."
- If a message is accompanied by a message number, place the number at the end of the message text.
- Left-align multiple-line messages.

3.6.2 Auditory Feedback

The most common auditory feedback is the system beep, which can alert the user to minor and obvious errors (for example, invalid keypresses or mouse clicks). For example, if the user scrolls to the top of the data and clicks the up scroll arrow or presses the UP ARROW key, the system can beep instead of displaying a message. The beep can also be used with other forms of notification, such as flashing (discussed in section 3.6.1.1.4) or message dialogs.

System beeps should be used sparingly for the following reasons. First, many users (as well as non-users within hearing distance of the computer) find beeps annoying. Second, frequent use of the system beep jades the user to its sound. Third, beeps are ephemeral messages that leave no trace. If the user is momentarily away from the computer or if working conditions prevent the user from hearing the beep when it occurs, the beep will fail to convey its message. Finally, the user can turn the warning beep off, so it is not a reliable source of feedback.

3.7 Editing Text

Applications should implement the following text-editing actions to let users edit text in windows and dialog boxes:

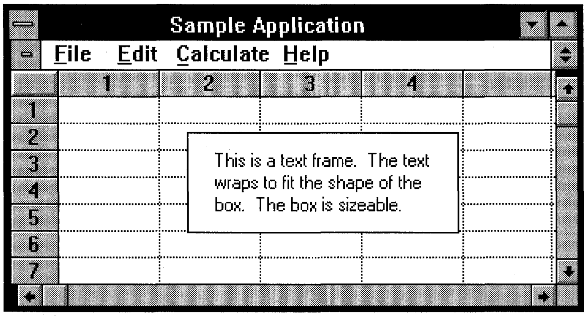
- When text is being inserted with only an insertion point, the insertion point moves one character to the right for each character the user types, and new characters appear to the left of the insertion point.
- With any selection, inserting text (by typing or by using the Paste command) removes the selected text and reduces the selection to an insertion point. The insertion point then acts as described above. The deleted text does not go into the clipboard.
- With any selection, deleting text (by means of BACKSPACE, DEL, or Cut) removes the selected text. The Cut command and its shortcut (CTRL+X) put the deleted text on the clipboard; the BACKSPACE and DEL keys do not.
- When the selection is only an insertion point, BACKSPACE deletes the character to the left and moves the insertion point to the left. The deleted character does not go onto the clipboard.
- When the selection is only an insertion point, DEL deletes the character to the right of the insertion point, which does not move. The text to the right of the insertion point moves to the left to fill the place of the deleted character. The deleted character does not go onto the clipboard.
- Whenever text is deleted, close the gap. (This operation is sometimes called "auto-joining.")
- The following behavior is suggested when the user selects one or more words by double-clicking or double-dragging: If there is a space after the last word in the selection, the space is also selected. If the user then deletes the selection, the fate of this trailing space depends on how the selection was deleted.
  - If the user deleted the selection by using BACKSPACE, DEL, or Cut, the trailing space is also deleted. As a result, the words before and after the deleted selection are separated by one space instead of two. In most cases, this result is precisely what the user wants.
  - If the user deleted the selection by typing one or more new words, the trailing space is not deleted. Instead, it is retained to separate the last new word from the first word following the selection. Otherwise, the user would have to reinsert the space so that the two words do not run together.
- As an acceptable extension to the basic text-editing model, applications can implement a user-selectable mode that provides nondestructive typing and backspacing when text is selected.

3.8 Moving Objects

Objects are moved by clicking and dragging either within their filled area or on their border with mouse button 1. Users should not be required to click on other regions (for example, special borders or handles) to move objects.

3.9 Text Frames

Text frames are sizable fields into which the user can type text (see Figure 3.8).[12] They are generally rectangular, but other shapes may also be used. When the user resizes the text frame, the text is rewrapped to fit within the new borders of the frame.



This is a text frame. The text wraps to fit the shape of the box. The box is sizeable.

Figure 3.8 Text Frame

The pointer appears as an arrow over an unselected text frame; the user can click to select the frame. A selected frame has resize handles. While the frame is selected, the pointer changes to an I-beam over the text, to an arrow over the border, and to a resize pointer over a resize handle. These selection rules follow the general principle for hierarchical selection of objects and their contents: The first click selects the container, and the second click selects its contents.

When the pointer is an I-beam, the user can follow standard text selection techniques to select the text within the frame. Formatting commands apply whenever the text frame or the text within it is selected. If the user selects text from the frame, then moves the pointer over the border of the frame, the pointer changes back to an arrow to allow the frame to be selected.

[12]A text frame is a dynamic form of an edit field and should not be confused with the text box discussed in Chapter 6, section 6.4.

The user can move a text frame by dragging its border. If the frame is not currently selected, the user can also move the frame by clicking and dragging within it.

The Cut command, the Copy command, and the DEL key apply to the text frame as a whole if the frame is selected but does not contain a selection or an insertion point. If text is selected within the frame, the commands apply to the text. If there is no selection but there is an insertion point, Cut and Copy are dimmed, and DEL deletes the character after the insertion point.

When a text frame contains selected text or an insertion point, its resize handles are not displayed. Clicking in the empty space below the text places an insertion point at the end of the text.

Windows

Chapter 4

Windows are the fundamental interface objects through which data, commands, and controls are organized and presented to the user.

4.1 Screen Window Types

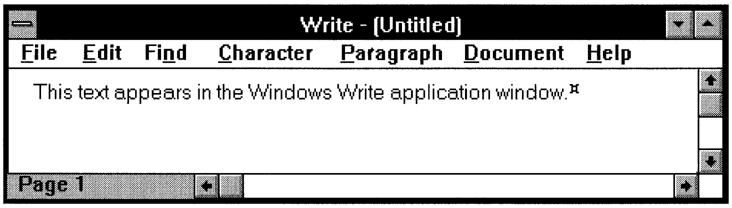
There are three types of screen windows: application windows, document windows, and dialog boxes.[1]

4.1.1 Application Windows

Application windows are movable and sizable, and they constitute the fundamental visual framework for data and commands in an application. Virtually all activity in an application takes place within the application window, with three exceptions:

- If the application window has been resized so that one of its dialogs or menu drop-downs will not fit inside it, the dialog or menu drop-down may appear partially outside the window.
- Movable dialogs may be moved outside the application window.
- The Help window is actually an independent application window and may be moved outside the primary application window.

Figure 4.1 shows an example of an application window. Application windows should always include a sizable frame and a title bar that contains at least the title of the application, a Control-menu box, and Minimize and Restore (or Maximize) buttons. Application windows may also include some or all of the other components discussed in section 4.2.



This text appears in the Windows Write application window.¤

Figure 4.1 Application Window

[1] Dialog boxes are different in many ways from other types of windows and will be discussed separately in Chapter 7. Toolboxes and palettes are examples of control bars and may appear in any type of window. These are discussed in section 4.2.8.

If an application that works with documents operates only on a single view of a single document at any given time, the document is displayed in the application window, and the document title is displayed in the title bar after the application title, separated from it by a hyphen (for example, "Write - REPORT.WRI"). Applications that allow users to open multiple views or multiple documents simultaneously can display the different views or documents in document windows, using the multiple document interface (MDI) instead of the single document interface (SDI).

4.1.2 Document Windows (MDI)

The multiple document interface (MDI) allows an application to manage multiple documents, or multiple views of the same document, within the main application window (also known as the "workspace"). These views or documents are displayed in separate windows called "document windows." Document windows may also be referred to by names that describe the contents of the windows more specifically, such as "group windows" (in the Windows Program Manager), "directory windows" (in the Windows File Manager), or "worksheet windows," "chart windows," and "macro windows." Figure 4.2 illustrates an MDI application, the Windows Program Manager, in which two group windows, Main and Accessories, are open.

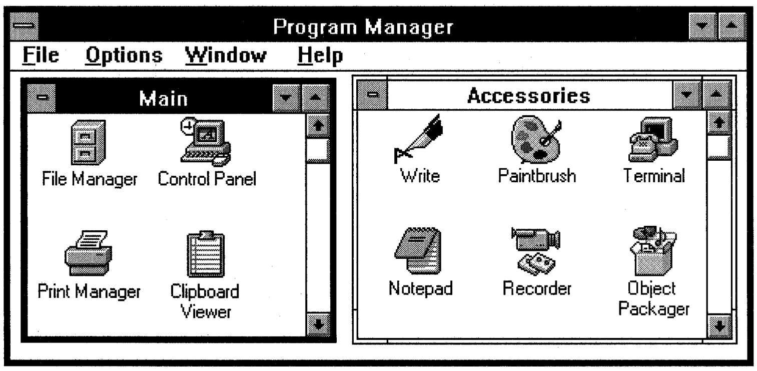


Figure 4.2 MDI: Windows Program Manager with Two Group Windows Open

4.1.2.1 Characteristics of Document Windows

A document window can be manipulated in the same way as the application window. Because document windows are movable and sizable, they have a title bar and a sizable window frame. All document windows must appear within the borders of the application window. If the user reduces the size of the application window so that it is smaller than a document window, the document window should be clipped.

A document window title bar should contain a caption that displays the name of the document in the window, a Control-menu box, and a Maximize button. Optionally, applications can allow document windows to be minimized; in this case, a Minimize button should appear on the title bar of the document window, to the left of the Maximize button (see Figure 4.2). Minimized windows are represented as icons. Applications should define a particular space within the application window (typically at the bottom) where the icons are placed by default, even if the user is allowed to move the icons elsewhere within the application window.

The Control menu for a document window parallels the Control menu for the application window. The document window Control menu can be accessed from the keyboard by typing ALT,HYPHEN (or ALT+HYPHEN). The document's Control menu can also be accessed through arrow keys, like other menus. Control menus are discussed further in Chapter 5, section 5.4.1.

If an MDI application has a menu bar, it should appear within the application window, along with any application controls (such as ribbons or toolboxes) that apply to all document windows. Placing these controls in the application window makes them available to all document windows. Scroll bars, however, are not shared among document windows; each document window should have its own scroll bar. The application window has a scroll bar if icons representing minimized document windows are hidden below the bottom of the application window, or if the application window contains a maximized document window that requires scrolling. The title bar of the application window should identify the application name but not the current document name, unless that document window is maximized.

4.1.2.2 Maximizing Document Windows

The multiple document interface allows the user to maximize the current document window to increase the amount of data that can be viewed. Maximization has the following effects, which are illustrated in Figure 4.3:

- The data from the document window is displayed in the application window.
- The document window and all associated controls disappear, except for the document window Control-menu box, which is displayed to the left of the first menu in the application window menu bar.

- The document title is placed in the application window title bar after the application name, separated from it by a hyphen (exactly as in an SDI application).
- A Restore button for the document window is added at the extreme right of the menu bar.

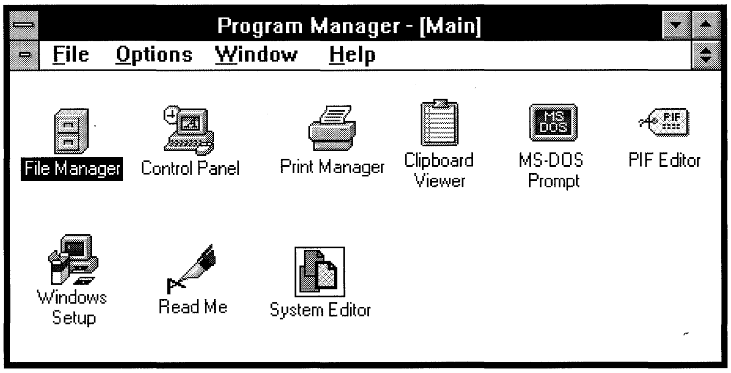


Figure 4.3 MDI: Windows Program Manager with Maximized Group Window

If the user maximizes one document window and then switches to another, the second window should also be maximized. Similarly, restoring one document window should restore the others to their pre-maximized sizes and locations.

4.1.2.3 Saving Window Configurations

If the user switches from an MDI application to another application (leaving the first application running) and then switches back, the arrangement of all document windows should be preserved, and the window that was active before the switch should be reactivated. If the user restarts an MDI application after exiting it, the application should provide a way to restore the workspace to its previous configuration, with all document windows arranged exactly as before. If the application cannot automatically preserve the information for restoring the previous configuration, a "Save Workspace" command may be added to the File menu. (This command generally follows the conventions of the Save command.) The user opens this file after restarting the application to restore the previous configuration. If the user opens other document windows before restoring the configuration, the new windows should be left open when the configuration is restored.

4.1.3 Launching Files Associated with MDI Applications

Suppose that one instance of an MDI application is running, and the user opens a file associated with the application by one of the following methods: by double-clicking an icon in the File Manager or Program Manager; by using the File Open command in the File Manager or Program Manager; or by dragging the file icon onto the application icon in the File Manager. In these cases, should a new instance of the MDI application be started, or should the file be loaded into a new document window within the existing instance?

Plausible scenarios favor each alternative. For example, if the user double-clicks on one file and then on another one immediately afterward, it is likely that he or she wants to load both files into the same MDI window. On the other hand, suppose that the user left the MDI application running earlier in the day with several files loaded into it. If the user later double-clicks on another file associated with the same application, it is not at all clear that he or she wants the new file loaded into the old instance; the new file may be part of a completely unrelated task.

Worse yet, suppose that the MDI application is a spreadsheet and that the newly opened file is an autoexecute macro that will operate on any other files that happen to be in the same MDI workspace. If the newly opened macro file were loaded into a spreadsheet window left over from some previous unrelated task, the results could be disastrous. Workspace configuration files are another type of file that the user almost certainly would not want to open inside an existing MDI window.

The cases mentioned so far—leftover unrelated instances, autoexecute files, and workspace configuration files—all argue that files opened from the Program Manager or the File Manager should be opened within new rather than existing instances of the application. An additional argument for the new-instance behavior is that all SDI applications already behave that way because they have no other alternative. Furthermore, the new-instance behavior is more consistent with the probable future movement away from an application-oriented window grouping model and toward a task-oriented model. Finally, the user who wants to open multiple files within the same MDI window can always do so by using the File Open command within the MDI window (rather than within the File Manager or the Program Manager). Because this method is available from the MDI application, it is not as important to provide same-instance functionality from the File Manager or from the Program Manager.

For these reasons, it is suggested that files opened from the File Manager or the Program Manager start a new instance of the associated application, except in the following cases:[2]

- File already open: If the file is already open, its window should be surfaced. If the file is already open inside an MDI application that contains several open document windows, its window should be brought to the front.
- Memory problems: If the file is associated with an MDI application that does not use memory efficiently when multiple instances are running, the application should make an intelligent decision about what to do. For example, the application could check available memory to see whether running one more instance would be likely to cause problems. If so, the application could display a warning and, if possible, offer the user a choice between starting a new instance or loading the file into an existing instance.

4.1.4 "Always on Top" Windows

Windows version 3.1 includes support for creating windows that remain on top of other windows.[3] This feature is useful for keeping control bar (palette) windows or modeless dialogs on top of the windows that they modify. However, the facility must be used carefully because these windows also sit on top of other application windows. Therefore, an application that uses this feature should change this attribute or hide supplemental windows when the main application window is inactive (or minimized).

It is suggested that windows that always exist on top of others be given a gray shadow (along the right and bottom edges) to visually distinguish them from other windows.

If the "always on top" feature is used on the main window of an application, the user should be allowed to change the attribute so that the window will not always float on top of other application windows. It is recommended that such applications include an "Always on Top" entry in the window's Control menu. Selecting this option should place a check mark next to the menu item and should set the attribute for the window. Selecting it again should remove the check mark and reset the window to overlap with other windows.

[2] One additional case will not be supported in Windows version 3.1, but is likely to be supported in future releases: Suppose that the file is part of a multiple selection in the File Manager. Any files in the selection that are associated with the same MDI application should be opened within the same MDI window, with each file in a separate document window. This behavior will provide a quick way to open a group of related files within the same window, without using the File Open command inside the MDI application.

[3] This feature should be used carefully because it is not explicity supported in Windows versions prior to 3.1.

4.2 Screen Window Components

All windows should include a frame, a title bar[4], and a control-menu box. In addition, windows may include some or all of the following components: menu bar, scroll bar, message bar, status bar, and control bars. Some of these components are illustrated in Figure 4.4.

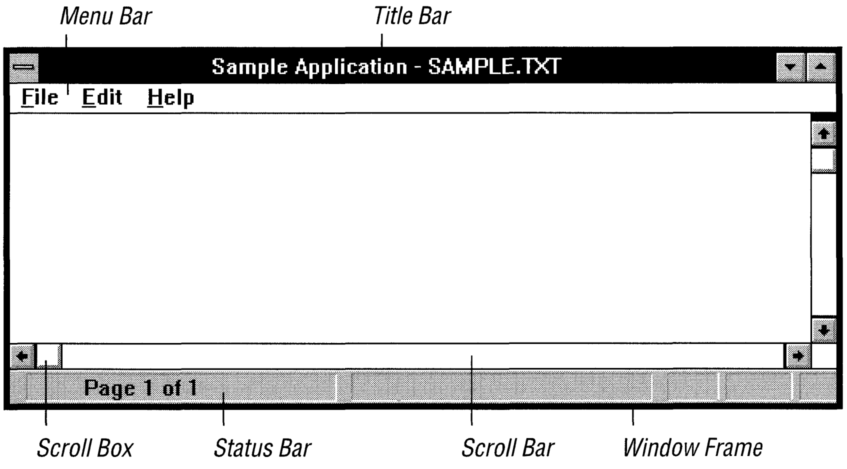


Figure 4.4 Screen Window Components

4.2.1 Window Frame

All windows have frames, except when they are maximized and fill the entire screen.[5] The frame defines the window boundary and distinguishes each window from others that it overlaps.

The frames of dialog windows are not sizable. The frame of a modal dialog is a one-pixel-thick line (default color black), inside which is a thicker border that shares the color of the title bar (default color blue). The frame of a modeless dialog is a single-pixel black line without an inner border. This difference in frames provides a visual cue for determining whether a dialog box is modal or

[4] Modal dialog windows in many current applications do not include title bars, but it is recommended that title bars be added to all dialogs. The title bar serves two purposes. First, it clarifies the purpose of the dialog by including the name of the command that invoked the dialog. Second, the title bar allows the user to move the dialog to reveal data obscured by the dialog. Adding a title bar to a dialog requires only trivial changes to program code.

[5] When the window is maximized to fill the entire screen, its frame is coextensive with the screen edge.

modeless; the distinction was previously provided by the presence or absence of a title bar (see footnote 4). For more information on types of dialog boxes, see Chapter 7, section 7.1.

Most other windows have sizable frames that consist of parallel one-pixel-thick lines inside of which is a thicker border. (Colors and border thickness are based on the settings in the Windows Control Panel.) When the pointer is moved over the border, it changes into the appropriate resizing pointer (see Chapter 3, Table 3.7). The user can then drag the frame with the mouse to resize the window. During the drag operation, a gray "ghost" frame moves under the pointer; when the mouse button is released, the window is redrawn at the new size. Dragging on the corners resizes the window horizontally and vertically at the same time; dragging on the horizontal parts of the frame resizes the window vertically, and dragging on the vertical parts of the frame resizes the window horizontally.

4.2.2 Title Bar

As shown in Figure 4.4, a title bar can contain a title, a Control-menu box (also known as a System-menu box), a Minimize button, and a Maximize button (if the window is not maximized) or a Restore button (if the window is currently maximized) . If the window is not maximized, the user can drag the title bar with the mouse to move the window to a new position. During the drag operation, a gray "ghost" window frame moves under the pointer; after the mouse button is released, the window is redrawn at the new location.

4.2.2.1 Title

The title is a unique label that identifies the application window. This identification is especially important when the user is working with multiple applications. At a minimum, the title should contain the name of the application with the first letter capitalized (for example, "Write"). The title may also contain additional text. For example, if the application operates on data files, the application name should be followed by a hyphen and the current file name (in uppercase letters and including any extension). The storage place for the file should not be included, because it may be long and it can intimidate inexperienced users who have no use for the information.[6] If no file is currently active, the title should contain a placeholder document name, such as (Untitled), Document*n*, Sheet*n*, Chart*n*, and so on, where *n* indicates a number (as in "Document1"). The placeholder should be in mixed case to provide a visual cue that it is not an official file name.

[6] Users can obtain this information from the Save As dialog box.

If a dialog box is displayed as the direct result of a command, its title should match the command name, unless the command name does not contain enough information. For example, in the Windows Write program, the dialog box produced by the File menu Open command is entitled Open, but the File menu Repaginate command is entitled Repaginate Document. If an intermediary message dialog is displayed before the dialog box (for example, to warn the user to save changes before executing the command), the application identifies itself in the title of the message dialog.

4.2.2.2 Control Menu

The Control menu contains commands for manipulating—changing the size, changing the position, and closing—the main window. These commands are described in Chapter 5, section 5.4.1. The Control menu is primarily an aid for keyboard users; the mouse has direct access to the actions the commands perform, but can also be used to select the commands from the Control menu. For example, the mouse user can double-click the Control menu to close the application window, instead of choosing the Close command from the Control menu.

4.2.2.3 Minimize, Maximize, and Restore Buttons

The Minimize, Maximize, and Restore buttons are graphical equivalents of the corresponding commands on the Control menu. Clicking the Minimize button reduces the main window to its minimum size (usually an icon) and hides all associated windows, including floating palettes or toolboxes. Clicking the Maximize button enlarges the main window to its maximum size; on many screens, the maximized main window fills the entire screen. When the user clicks the Maximize button or chooses the Maximize command from the Control menu, the Restore button replaces the Maximize button. When the user clicks the Restore button or chooses the Restore command, the Maximize button replaces the Restore button. Fixed-size windows do not have Maximize buttons.

4.2.3 Menu Bar

When a menu bar is used to provide access to commands in an application, it should be placed directly under the title bar of the application window. Document windows and dialogs do not generally have menu bars. The menu bar in the application window contains the titles of the menus provided by the application. The menus themselves contain items representing commands. Applications that support more than one document type may replace one menu bar with another according to the type of document displayed in the active window. (For additional information on menus, see Chapter 5, section 5.2.)

4.2.4 Scroll Bars

Applications should provide scroll bars for all windows in which the size of the data may exceed the size of the window. Scroll bars allow the user to move data through a window with the mouse, thereby revealing previously hidden portions of the data. A window can have vertical scroll bars, horizontal scroll bars, or both. If the user would never scroll in either direction, the associated scroll bar can be omitted. If a window with a scroll bar becomes inactive, the scroll bar should be left intact.

Inactive scroll bars in lists are discussed in Chapter 6, section 6.3.1.1.

4.2.4.1 Scroll Arrows

Scroll arrows appear at each end of the scroll bar, pointing in opposite directions away from the center of the scroll bar. The scroll arrows point in the direction that the window "moves" over the data, as if the data is fixed. When the user clicks a scroll arrow, the data in the window appears to move in the opposite direction of the arrow by an appropriate amount (for example, one line for text applications and one row or column for spreadsheets). When the window cannot be scrolled any farther in one direction or the other, the associated scroll arrow should be dimmed.

4.2.4.2 Scroll Box

The scroll box (also called the elevator, thumb, or slider) moves along the scroll bar to represent how far the current view of the document is from the top (for vertical scroll bars) or from the left edge (for horizontal scroll bars). For example, if the current view is at the middle of the document, the scroll box in the vertical scroll bar is in the middle of the scroll bar.[7] This behavior of the scroll box makes it a useful indicator for keyboard users as well as mouse users. The user can also drag the scroll box along the scroll bar to move to a different view of the document. If possible, the view should be updated continuously as the scroll box is dragged. However, if the view cannot be updated continuously with sufficient speed, it can be updated in a single jump at the end of the drag operation.

If documents in a particular application (for example, a spreadsheet) by default have substantial but mostly unused length and width, the behavior of the scroll box can be modified as follows: When the scroll box is dragged to the bottom of a vertical scroll bar or to the right of a horizontal scroll bar, the document scrolls to the end of the currently used portion of the document (that is, the portion containing data) rather than to the actual limit of the document. Pressing the scroll arrow next to the scroll box continues the scrolling into the unused portion of the document.

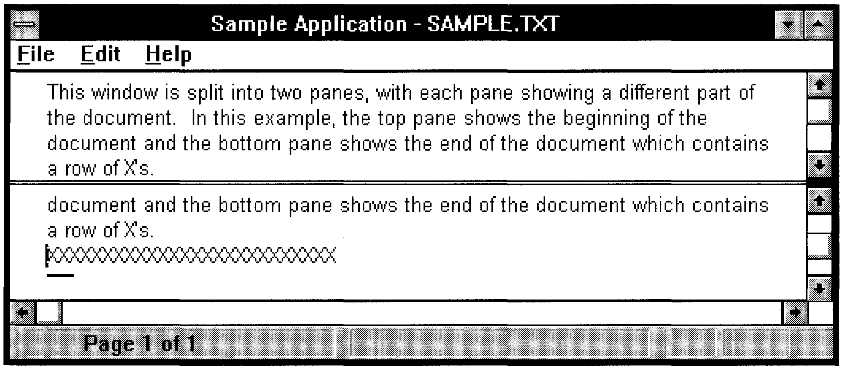
[7] As an optional extension to this model, applications may implement a proportional scroll box whose size indicates what portion of the document is visible in the window.

4.2.4.3 Scroll Bar Shaft

Clicking inside a vertical scroll bar shaft scrolls the data forward by the height of the window (if the click location is below the scroll box) or backward by the height of the window (if the click location is above the scroll box). To help users keep their place within a document, the new screen should preserve at least one line of data from the old screen. In other words, if the data is scrolled forward, the top line on the new screen should be the line that was at the bottom of the old screen; if the data is scrolled backward, the bottom line on the new screen should be the line that was at the top of the old screen. Similar recommendations apply to horizontal scroll bars.

4.2.5 Split Box and Split Bar

Applications can allow the user to split the application window into two or more separate viewing areas, called panes, as shown in Figure 4.5. For example, a user can split a window and then examine two parts of a spreadsheet at the same time. A split window can also display different views of the same data (for example, text view and outline view). Applications can let the user split the window into as many panes as is useful and practical. All panes, however, should be kept within the window.



This window is split into two panes, with each pane showing a different part of the document. In this example, the top pane shows the beginning of the document and the bottom pane shows the end of the document which contains a row of X's.

document and the bottom pane shows the end of the document which contains a row of X's.
XXXXXXXXXXXXXXXXXXXXXXXXXX

Figure 4.5 Application Window Split into Panes

For all splittable windows, applications should provide a split box. (For example, this could be implemented as a solid box located at the top of the vertical scroll bar and at the left end of the horizontal scroll bar beyond the tip of the scroll arrow.) The user can drag the split box with the mouse to split the window into two separate panes at the desired split position. Double-clicking on the split box is an optional shortcut for splitting a window in the middle. Keyboard techniques for splitting windows are discussed in section 4.3.4.

Splitting the window sets up a split bar between the panes. The split bar is a double line with a blank pixel between the lines. If the application allows only one split in a single direction, the split box appears only at the end of the split bar when the window is split. If the application allows multiple splits in a single direction, the split box appears at its original position, as well as at the end of each split bar.

When the pointer is positioned over the split bar or the split box, it changes to the split pointer (see Chapter 3, Table 3.9). After the window is split, the user can drag anywhere on the split bar or the split box with the mouse to adjust the sizes of the two panes proportionally. If the window is split both horizontally and vertically, the user can adjust multiple panes at once by dragging the intersection of the split bars. Dragging the split bar or box to either end of the workspace closes the pane in the direction of the drag. For example, if the user drags the split bar or box to the top of the window, the upper pane closes. Double-clicking on the split bar or box is an optional shortcut for closing the pane.

Applications should establish a minimum height and width for each pane. For example, the minimum height might be the amount needed to display the three parts of a scroll bar—two scroll arrows and a scroll box. The minimum width might be the amount needed to display the title bar controls and the shortest recognizable part of the title.

When the user splits a window, the application should display scroll bars to scroll each pane perpendicular to the direction of the split. For example, if the user splits a window horizontally (one pane above the other), each pane should have its own vertical scroll bar. The user can operate these scroll bars independently of each other, in the same way as for a window that is not split. The panes should share a single scroll bar in the direction parallel to the split border, unless users are likely to require independent scrolling of the panes in this direction. For example, if the split is horizontal, both panes should share one horizontal scroll bar (if any) that appears at the bottom of the lower pane.

For additional information about scrolling in panes, see section 4.3.6.

4.2.6 Message Bar

The message bar is an optional component at the bottom of an active window that lets an application request information from the user or display status information about a selection, a command, or a process. The message bar is also a convenient place to explain menu and control bar items as the user highlights each item (see Figure 4.6) and to display help information. Messages longer than the message bar should be displayed in message dialogs (see Chapter 7, section 7.1.4).

The standard placement of the message bar is at the bottom of the window, but applications may allow users to select another location. The application may also provide an option for suppressing the message bar, so this area should not be used to present essential information or messages that require acknowledgment by the user.[8]

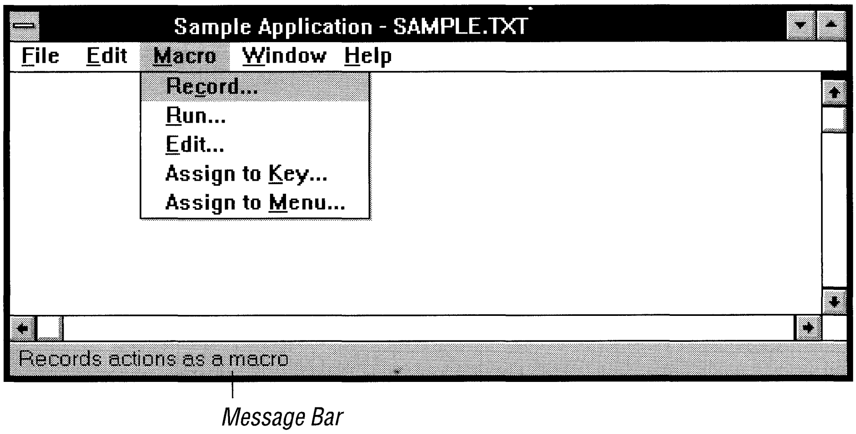


Figure 4.6 Message Bar

4.2.7 Status Bar

A more elaborate form of the message bar is the status bar (see Figure 4.7), an optional window component that displays information about the current state of the application.

In addition to brief messages, the status bar may include information such as the current cursor location and any current keyboard-initiated modes for selection (for example, Extend mode) and typing (for example, Overtype and Caps Lock). The "normal" modes, such as Insert or non-Caps-Lock mode, are indicated in the status bar by the absence of the indicator for the opposite mode.

[8] These message bar options, if provided, should be available from the dialog or menu used for other viewing options.

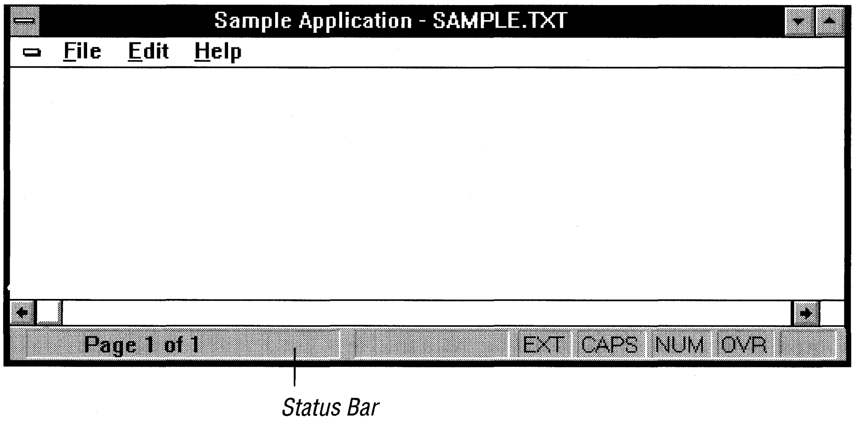


Figure 4.7 Status Bar

Table 4.1 lists the mode indicators that can optionally be used on the status bar. Applications can spell these out if there is enough room on the status bar. Additional keyboard modes can be indicated at the right end of the status bar, after the spaces reserved for the more common modes.

Table 4.1 Suggested Mode Indicators for Use in Status Bars[9]

Mode	Indicator
Extend Selection	EXT
Caps Lock	CAPS
Num Lock	NUM
Scroll Lock	SCRL
Overtype	OVR
Recording Macro	REC

Note: The mode indicators are listed in the order in which they should appear (from left to right) on the status bar.

Because the status bar takes up space that could be used to display data, applications should always provide a way for the user to control whether the status bar is displayed. Applications that contain a View menu should include a Status Bar toggle item on that menu.[10]

[9] Indicators that are not supported by the application can be omitted.

[10] The application should remember the display state for status bars, message bars, and control bars and restore them the next time the application is invoked.

4.2.8 Control Bars: Ribbons, Rulers, Toolboxes, and Palettes

Applications may implement control bars to provide quick and convenient access to frequently used choices and commands. In word-processing applications, ribbons and rulers are examples of control bars. In painting programs, toolboxes and color/pattern palettes are sometimes implemented as control bars, although they often appear as independent movable windows. Figures 4.8 and 4.9 show examples of control bars.

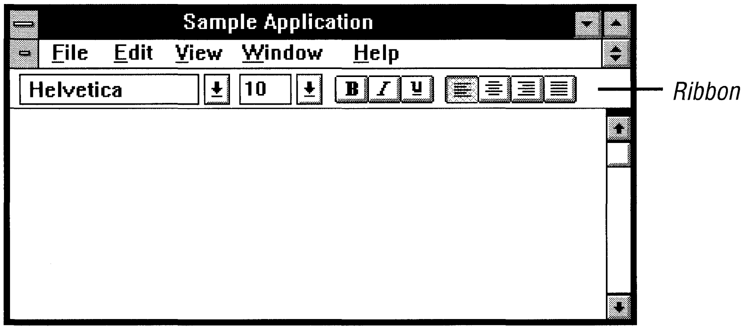


Figure 4.8 Control Bars Ribbon

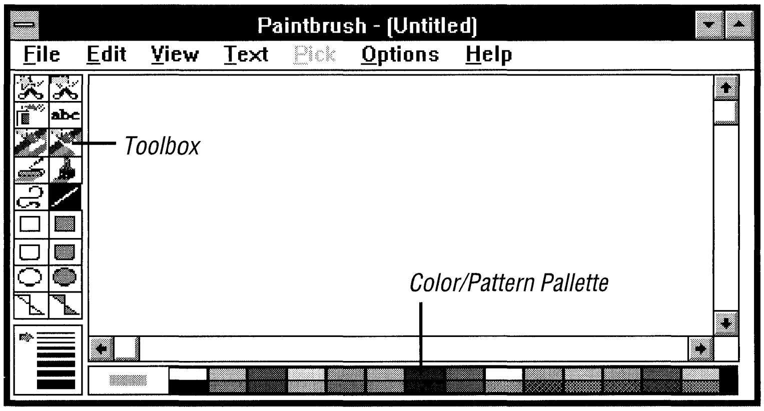


Figure 4.9 Control Bars: Toolbox and Color/Pattern Palette in Paintbrush

Control bars may occupy a fixed position within the application window, or they may be placed in a supplemental window or a dialog box and thus become movable. A movable control bar is always in front of the window to which it applies (see section 4.1.4). Thus, a control bar is never hidden by its associated parent window, although it may be hidden by one of its peers.

Movable control bars should include a miniature title bar and a Control-menu box. The title bar need not contain a title; its main purpose is to allow the mouse user to drag the control bar to a new location. The Control menu should contain the Move command to allow keyboard users to move the control bar. In addition, the Control menu may also include the Close command. Control menus for control bars typically do not include the other commands (for example, Maximize) that are found on Control menus for application or document windows (see Chapter 5, section 5.4.1).[11]

Like status bars, control bars take up space that could otherwise be used to display data, so applications should provide commands to specify which control bars are displayed. Commands for movable control bars should appear on the Window menu; commands for fixed-position control bars that are part of the main application window should appear on the View menu.

Mnemonic access to controls on control bars is discussed in Chapter 3, section 3.3.3.

4.3 Window Operations

Basic window operations include moving, resizing, and closing windows; splitting windows into panes; switching between windows or panes; and scrolling data in windows and panes. Mouse techniques for many of these operations rely on the window components described previously and have already been discussed in the sections devoted to those components. Keyboard techniques for many of these operations rely on Control menu commands and are discussed in Chapter 5, section 5.4.1. This section briefly summarizes all the major window techniques, but focuses on those not covered elsewhere—namely mouse techniques that do not involve specific window components and keyboard techniques that do not rely on Control menu commands.

4.3.1 Moving Windows

To move a window with the mouse, the user drags the title bar of the window. To move a window with the keyboard, the user chooses the Move command from the Control menu. This causes a gray "ghost" window frame to appear on top of the regular frame. The pointer changes to the window movement pointer (the last pointer shown in Chapter 3, Table 3.8). The arrow keys can be used to move the ghost frame to the desired location. Pressing ENTER completes the command and redraws the window at the new location.

[11]Size can be included if the window is sizable.

4.3.2 Resizing Windows

Mouse techniques for resizing windows rely on direct manipulation of window components (the window frame and the Minimize, Maximize, and Restore buttons) and have already been discussed in sections 4.2.1 and 4.2.2.3. Keyboard techniques rely on the Control menu resizing commands: Minimize, Maximize, Restore, and Size (see Chapter 5, section 5.4.1). These commands can also be accessed with the mouse.[12]

4.3.3 Closing Windows

To close a window with the mouse, the user can double-click on the Control-menu box. Windows can also be closed by selecting the Close command from the Control menu (see Chapter 5, section 5.4.1); this command can be selected with the mouse or with the keyboard.

4.3.4 Splitting Windows into Panes

The mouse user can split a window into panes by dragging the split bar to the desired location, as described previously in section 4.2.5. The keyboard user can split a window into panes by choosing the Split command from the View menu.[13] This results in a split bar being placed across the middle of the window, with a split box at the right end, inside the scroll bar. Pressing arrow keys moves the split bar up or down; pressing ENTER sets the split at the current position of the split bar.

4.3.5 Switching Windows and Panes

Either the mouse or the keyboard can be used to switch from one application window to another, from one child window to another child window within the same application, or from one pane to another pane within the same window. The reactivation of a window or pane does not affect any pre-existing selection there; the selection and focus are restored to the state that existed when the window or pane was last active.

[12]Some applications let users size windows by dragging the area at the intersection of the vertical and horizontal scroll bars. This behavior is a carryover from Windows version 1.0 and is neither encouraged nor supported in Windows version 3.1, particularly because this area is likely to be used for different purposes in the future.

[13]For MDI applications, the Split command can also be placed on the document window's Control menu or on the Window menu.

4.3.5.1 Mouse Techniques

- To switch from one window or pane to another, the user simply clicks on the desired window or pane.
- To switch to an application window that has been minimized, the user double-clicks on the icon; this reactivates the window and restores it to its previous size.
- To switch to an application window that is completely obscured by other windows, the user can double-click on the screen background (outside any application windows) to obtain a dialog that contains a list of running applications. (The same dialog is available from the Switch To command on the Control menu.) The user can activate the desired application by first clicking on its name to select it, and then pressing a confirmation button in the dialog. Double-clicking on the name of the application immediately activates the application without requiring confirmation.

4.3.5.2 Keyboard Techniques

Users can also switch between application windows, window panes, and document windows with the keyboard.

4.3.5.2.1 Switching Application Windows with the Keyboard Several keyboard techniques are available for switching between application windows (or between icons that represent minimized application windows):

- CTRL+ESC (equivalent to the Switch To command on the Control menu) invokes an explicit list of available applications.
- ALT+ESC and ALT+TAB (with or without SHIFT) treat the application windows as a stack of cards. The visible window is at the top and is called the active window. When a window becomes "active," it moves to the front of the stack. The keyboard user can manipulate the windows in the following ways:
  - Move the front window to the back of the stack (ALT+ESC) or the back window to the front (ALT+SHIFT+ESC).
  - Shuffle through the windows from front to back (ALT+successive TABs), or from back to front (ALT+SHIFT+successive TABs), and pick one to put at the front of the stack.

Table 4.2 describes these window-switching techniques in detail.

Table 4.2 Keyboard Techniques for Switching Application Windows

Key	Action
ALT+ESC	Moves top (active) application to bottom of stack, thereby deactivating the application. The application that was second in the stack is now on top. If its window was open, it is automatically activated; if it was minimized, it can be opened and activated by pressing ALT+SPACEBAR to display the Control menu and choosing Restore.
ALT+SHIFT+ESC	Opposite of ALT+ESC: Moves *bottom* application to *top* of stack. The application that was on top is now second and is deactivated.
ALT+TAB	Shuffles one step from front to back through stack of windows, displaying next application's name. Releasing ALT places named window on top of the stack, thereby activating it; if the window was minimized, it is restored to its previous size. All other windows remain in their previous relative stack positions.
ALT+SHIFT+TAB	Opposite of ALT+TAB: Shuffles one step from *back to front* through stack of windows, displaying next application's name. Releasing ALT activates named window. All other windows remain in their previous relative stack positions.
CTRL+ESC	Invokes a dialog that allows the user to switch to any application in a list of currently running applications.

4.3.5.2.2 Switching Window Panes with the Keyboard

Switching between panes follows either a clockwise or a counterclockwise rotation. Table 4.3 lists the relevant techniques.

Table 4.3 Keyboard Techniques for Switching Panes

Key	Action
F6	Moves clockwise to next pane of active window.
SHIFT+F6	Moves counterclockwise to next pane of active window.

4.3.5.2.3 Switching Windows within an Application with the Keyboard

Switching between windows within an application is conceptually similar to switching between panes. Consequently, the keyboard assignments for switching windows are extensions of the keys that switch panes. Table 4.4 lists these key assignments. Users can also choose a window from the numbered list provided in the Windows menu (see Chapter 5, section 5.5.2.1) through standard keyboard techniques.

Table 4.4 Keyboard Techniques for Switching Windows Within an Application

Key	Action
ALT+F6	Moves to the application's next open non-document window, such as a modeless dialog box or movable control bar.
ALT+SHIFT+F6	Same as ALT+F6, but moves through the non-document windows in the reverse order.
CTRL+F6*	Moves top document window to bottom of stack, thereby activating the formerly second document window. (Analogous to ALT+ESC for application windows.)
CTRL+SHIFT+F6*	Reverse of CTRL+F6; moves *bottom* document window to *top* of stack. (Analogous to ALT+SHIFT+ESC for application windows.)

* MDI applications only.

4.3.6 Scrolling Data in Windows and Panes

When a window or pane is too small to display all the available data, the user must be able to scroll the data to bring previously hidden portions into view.

4.3.6.1 Mouse Scrolling

The mouse can scroll data either explicitly or automatically.

4.3.6.1.1 Explicit Scrolling Mouse techniques for explicit scrolling rely on scroll bars and have already been discussed in section 4.2.4. Table 4.5 lists these techniques. Note that explicit scrolling with the mouse does not move the cursor or change the selection.

Table 4.5 Mouse Techniques for Scrolling

Mouse Action	Effect
Click vertical scroll arrow	View moves over data in direction of arrow by height of one data unit (for example, line of text or spreadsheet cell).*
Click horizontal scroll arrow	View moves over data in direction of arrow by width of one data unit (for example, character or spreadsheet cell).*
Click scroll bar shaft between scroll arrow and scroll box	View moves over data in direction of arrow by size of window. To help user maintain context orientation, make old and new window contents overlap by at least one data unit (for example, for forward scrolling, display last line of old contents at top of new contents).
Drag scroll box to new position in shaft	Move view to the location analogous to the location of the scroll box within the shaft (for example, middle of shaft represents middle of document). If performance permits, view should be updated continuously as box is dragged; otherwise, view can be changed in a single jump after drag is complete.

* If the data unit is very small (for example, a pixel in a graphics program), applications may choose to move the view by more than one unit, to make scrolling less tedious.

4.3.6.1.2 Automatic Scrolling The mouse scrolling techniques summarized in Table 4.5 all have scrolling as their explicit goal. Users may also scroll data as a secondary result of employing mouse techniques for object selection or object movement. This type of scrolling is called automatic scrolling. For example, in word-processing applications, if the user moves the pointer to the bottom boundary of the window during text selection, the data is scrolled upward (or equivalently, the view is moved downward) to allow the selection to be extended downward. Similarly, in graphics applications, if the user drags an object to the boundary of a window, the graphics document is automatically scrolled to let the user place the object in a part of the document that was not previously visible.

4.3.6.2 Keyboard Scrolling

The keyboard can also be used to scroll data through windows. An application can provide three types of keyboard scrolling:

- Normal scrolling moves data through the window and also moves the cursor.
- Automatic scrolling moves data or text when the user presses a cursor key with the cursor at the "boundary" of the visible data. (This type of scrolling is similar to the automatic mouse scrolling described previously; the only difference is that the cursor plays the role of the pointer.)
- Scroll Lock scrolling moves the data through the window without moving the cursor.

4.3.6.2.1 Normal Scrolling For normal scrolling with the keyboard, the user presses one of the navigation keys. These keys move the data through the window and reposition the cursor within the visible data. The application is responsible for establishing a new position for the cursor in the data when scrolling moves the current cursor position outside the view of the data. If scrolling leaves the current cursor position in view, the position need not change. The navigation keys are listed in Chapter 2, Table 2.5.

4.3.6.2.2 Automatic Scrolling When the cursor reaches the "boundary" of the visible data, pressing the arrow key that moves the cursor toward that boundary scrolls the data by some reasonable amount—typically, the height or width of one data unit. For example, if the user presses the down arrow key when the cursor is in the last line of the window, the window scrolls by one line. If a data unit is very small (for example, a single character in a line of text that is being scrolled horizontally), the application may scroll the data by a larger amount.

Mouse and keyboard techniques for automatic scrolling can also be used for lists and for text in text boxes.

4.3.6.2.3 Scroll Lock Scrolling Applications can also (optionally) implement scrolling that is modified by the SCROLL LOCK key. When the user presses SCROLL LOCK and then uses a navigation key, the data scrolls as described previously, but the position of the cursor does not change and existing selections are preserved. In Scroll Lock mode, scrolling with the keyboard is the equivalent of scrolling with the mouse, which also does not change the cursor position and preserves existing selections.

4.3.6.3 Linked Scrolling in Panes

In most cases, scrolling data in panes is the same as scrolling data in windows. If two panes have separate scroll bars for scrolling in a particular direction, scrolling one pane has no effect on the other. If two panes share a scroll bar for a particular direction, scrolling one pane scrolls the other by the same amount. In a few cases, however, it is useful to provide special scrolling linkages between panes.

Suppose that the user splits a window into two panes and displays a different view of the data in each pane. Scrolling the data in one pane should move the data in the other by the same amount, but because different views result in different spacing arrangements, this movement may not result in the same data being displayed in the two panes. Instead, scrolling should move the data in different amounts, if necessary, so that both panes display the same information. For example, in a word-processing program with an outlining facility, suppose the user splits the window into two panes and turns on outline view in the top pane. Scrolling the outline view to the fifteenth outline heading in the document would also scroll the text view to the fifteenth heading. The outline view would only need to scroll by a small amount (through the headings) to reach the fifteenth heading, whereas the text view would have to scroll through a larger amount (through the headings and the text) to reach the fifteenth heading.

Grid-oriented applications, such as spreadsheets and databases, may provide a special type of linked scrolling with a Freeze Panes command. Suppose that the user has a large table of data with row and column headings. If the row headings are placed in a pane at the left and the column headings are placed in a pane at the top, the Freeze Panes command links the panes so that the headings are always kept in view when the user scrolls the data within the table. For example, if the user scrolls the data pane downward, the column headings do not move (that is, they remain in view), but the row headings scroll down with the data. Similarly, if the user scrolls the data pane to the right, the row headings do not move, but the column headings scroll right with the data. Thus, the relevant row and column headings for any part of the data always remain in view. The command also prevents the data pane from being scrolled into the area shown in the heading panes. When selected, the Freeze Panes menu item changes to Unfreeze Panes to allow the command to be reversed.

Menus

Chapter 5

A menu is a list of items from which the user can choose; each item represents a command, either explicitly (for example, Cut) or implicitly (for example, Bold—that is, "apply bold formatting"). All applications that have commands should provide menus to give the user access to the commands.

5.1 Types of Menus

An application can implement three types of menus:

- Drop-down (also known as pull-down) menus on a menu bar.
- Pop-up (also known as contextual) menus.
- Cascading (also known as hierarchical) menus.

5.1.1 Drop-Down Menus

A drop-down menu is represented by a menu title (for example, File, Edit, Help) that appears in the menu bar of an application window (see Figure 5.1).

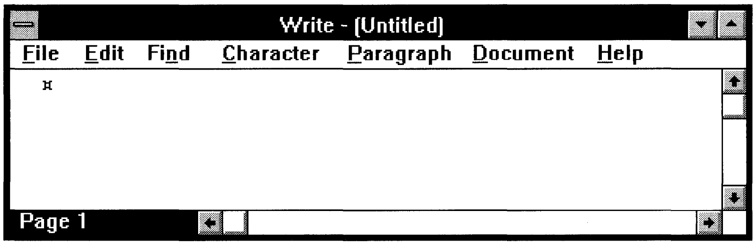


Figure 5.1 Closed Drop-Down Menus in Menu Bar

To display the menu, the user can either click on the menu title, or press ALT followed by the underlined mnemonic access character in the menu title.[1] Figure 5.2 shows an open drop-down menu. Drop-down menus are by far the most common type of menus in current applications, but applications may also use pop-up menus and cascading menus, as discussed in sections 5.1.2 and 5.1.3.

[1] ALT + underlined mnemonic access character is also supported.

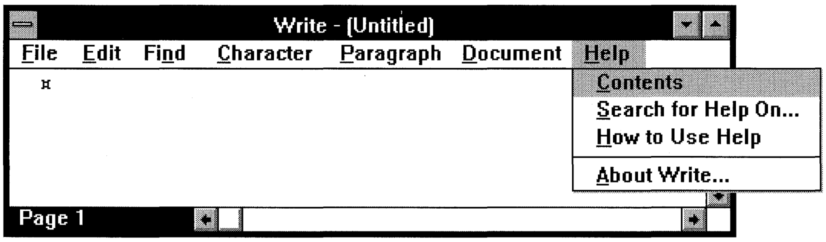


Figure 5.2 Open Drop-Down Menu

5.1.2 Pop-Up Menus

Pop-up menus are floating menus that appear when specifically invoked by the user. The items displayed on the menu depend on where the pointer was located when the button was pressed; hence, pop-up menus are also called contextual menus. For example, if the pointer was positioned over selected text, the menu might include text-specific commands; if the pointer was on the title bar of a window, the menu might include commands for moving and resizing the window.

Pop-up menus provide an efficient way to access commands. Because they are presented at the pointer's location, they eliminate the need for the user to navigate to a menu bar or control bar. Furthermore, pop-up menus are only displayed on demand, so they do not take up dedicated screen space.

5.1.2.1 Appearance and Location

A pop-up menu follows the guidelines for standard drop-down menus discussed in section 5.2, except that it does not have a title (see Figure 5.3).

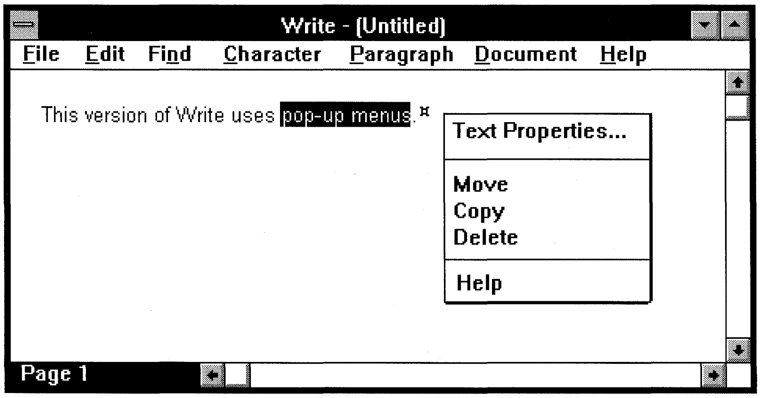

This version of Write uses pop-up menus.

Figure 5.3 Pop-Up Menu

It is suggested that the pointer's hot spot be initially positioned close enough that the user can move into the menu easily; but not so close that the pointer is positioned on an item, thus selecting a command on the menu inadvertently. If the pointer is positioned in such a way that the menu would appear clipped or off-screen, the menu is adjusted so that it appears fully on the screen.

Pop-up menus are primarily designed for mouse users. At the present time, there is no defined keyboard interface, except for UP ARROW, DOWN ARROW, ENTER, and the ESC key, which can be used to cancel a pop-up menu. Consequently, menu items should not have mnemonic access letters or shortcut keys. For information on accessing pop-up menus with the mouse, see section 5.3.1.1.2.

5.1.2.2 Additional Guidelines

- Pop-up menus are designed to provide an efficient method for accessing common, contextual commands. For this reason, a pop-up menu should not include too many commands or multi-level cascading menus. Single-level cascading menus are acceptable and should follow the guidelines in section 5.1.3.
- If the pointer is positioned over selected text that contains heterogeneous items (for example, both text and graphics), at the minimum the pop-up menu should provide the commands that represent the intersection of pop-up commands pertaining to all items.
- Applications should not place individual property settings on pop-up menus. Commands for bold, italic, font family, style, and size are better presented in property dialogs or in special property viewers like control bars. (However, a pop-up menu can contain commands that result in property setting dialogs.)
- Applications should use pop-up menus for frequently used commands and not simply repeat the menu items from the menu bar. Pop-up menus may also include commands that logically apply to the limited context of the selected object. For example, a pop-up menu for a text selection can include separate items for a character properties dialog and a paragraph properties dialog, although these would be better presented as different "pages" of a single text properties dialog. Another example is Undo. This command may not apply to the particular selection, but to the domain to which that selection belongs.

5.1.3 Cascading Menus

A cascading or hierarchical menu is a submenu (also known as a child menu) attached to the right side of a menu item (called the parent item, in the parent menu). Figure 5.4 illustrates a cascading menu. Menu items that lead to cascading menus are marked with a right-pointing triangle after the menu item name. Cascading menus can be added to drop-down menus, contextual menus, or even other cascading menus. In general, however, the use of multilayer cascading menus should be avoided, because they compound the difficulties discussed in section 5.1.3.2 for single-layer cascades. The user drops down a cascading menu by selecting the parent menu item with any of the usual techniques for menu item selection.

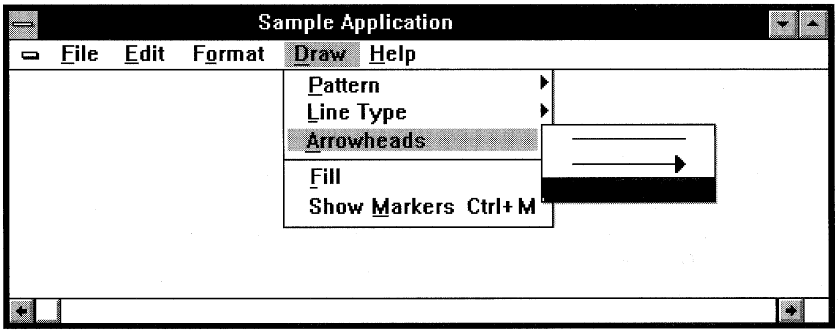


Figure 5.4 Cascading Menu

5.1.3.1 Advantages of Cascading Menus

- The top-level (parent) menus are simplified, because some commands are hidden in the cascading menu.
- More first-letter mnemonics are available. Mnemonics need only be unique within either the parent menu or the cascading menu, so there are fewer first-letter conflicts.
- High-level command browsing is easier. By dragging the mouse across the menu bar, the user can see important commands or command groups without being distracted by the details hidden in the submenus.

5.1.3.2 Disadvantages of Cascading Menus

- Access to submenu items is more difficult than access to top-level items.
  - Dragging to a submenu item requires extra coordination to negotiate the change in direction.
  - Accessing the submenu item by means of mouse clicks requires an extra click and a change in mouse direction.

- Keyboard access requires an extra keypress.
- Exhaustive command browsing is more difficult; the user cannot see all menu items by dragging the mouse across the menu bar.

Except for complex products with many commands, the disadvantages of cascading menus tend to outweigh the advantages. In general, cascading menus should be avoided whenever possible. If they are used, they are most appropriate for ordinary commands (for example, Insert Break -> Section), dialog box commands (for example, File Print -> Preview), and interdependent (one-of-N) settings (for example, Format Alignment -> Left). Cascading menus are less appropriate for independent (M-of-N) settings (for example, Format Character -> Bold), especially if users may want to choose more than one setting at a time (for example, Bold, Italic, and Underline). In such cases, it is preferable to place independent settings in a dialog or on a ribbon so that users who want to set multiple attributes can simply open the dialog once or directly access the ribbon buttons, rather than repeatedly working through two levels of menus.

5.2 Menu Components

Each menu consists of a menu title and one or more menu items.

5.2.1 Menu Titles

All drop-down and cascading menus should have menu titles. A pop-up menu typically does not have a title. Titles for drop-down and cascading menus should represent the entire menu and should reveal as clearly as possible the purpose of all items on the menu. In drop-down menus, the menu title is the term that appears on the menu bar. In cascading menus, the menu title is the name of the parent menu item.

Compound titles are acceptable (for example, Fontsize) but should be used sparingly because they may strike users as odd. The title may not contain numbers or spaces; spaces would increase the possibility of confusing a single two-word title with two one-word titles. Avoiding the use of spaces is especially important for systems that provide proportionally spaced system fonts.

As shown in Figure 5.1, each menu title should contain an underlined mnemonic access character that gives the user direct access to the menu through the keyboard. The access character should be (in suggested order of preference):

- The first letter of the menu title, unless another letter offers a stronger mnemonic link or the first letter conflicts with another menu title (for example, File and Format).

- A distinctive consonant in the title.
- A vowel in the title.

No two menu titles should use the same access character. If an application uses a non-Roman writing system (such as Kanji) but runs on a standard keyboard (such as an IBM® PC keyboard), the menu names can be given a Roman alphabetic character prefix as the menu access character.

Menu titles may be temporarily removed from the menu bar under the following conditions:

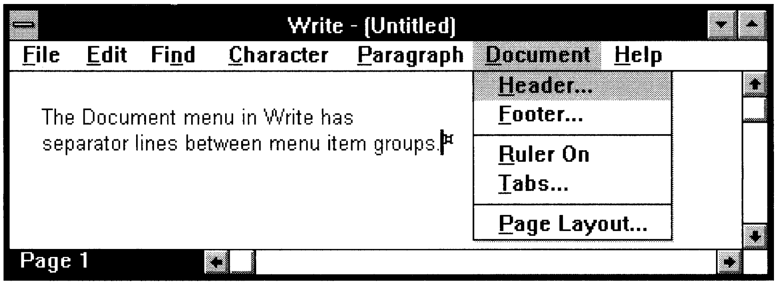
- Applications that support different document types may display different menu bars, depending on the document type that currently has the focus. In such applications, if no documents are open, there is no basis for deciding which menu bar to display. Accordingly, all menu titles should be removed from the menu bar, except for titles of menus containing commands that remain active even when no documents are open (for example, the Control and File menus).
- In applications that support only one document type, menu titles should not be removed from the menu bar when no documents are open. Instead, the titles of menus that do not contain any active commands should be dimmed.[2]

5.2.2 Menu Items

Menu items can be names of actions (for example, Cut or Paste), attributes (for example, Bold), documents (for example, "REPORT.DOC"), or windows (for example, "REPORT.DOC:2"). Menu items can also be graphical (for example, representing patterns or drawing tools). If an item has a keyboard shortcut, the shortcut should be listed on the menu next to the item. Shortcuts should be aligned with other keyboard shortcuts in the menu. By default, Windows provides left-alignment; however, applications may choose to right-align shortcuts to conserve space.

Whenever a menu contains items that fall into logical groups, the groups should be separated with a line (see Figure 5.5). The basic separator is a solid single line that spans the width of the menu. If a group has logical subgroupings, a dashed line can separate subgroups. People tend to remember information in chunks of three or four items. Accordingly, whenever possible, a menu should contain no more than four groups of items, with three or four items in each group. It is possible to implement scrolling menus that hold a very large number of items. However, such menus are not recommended, because the items are never visible at the same time and are therefore difficult to remember.

[2] Although the menu titles are dimmed (indicating unavailability; see section 5.2.2.3), they should still be selectable. This will allow the user to explore menu contents and to obtain help on a menu item.



The Document menu in Write has separator lines between menu item groups.

Figure 5.5 Drop-Down Menu with Separator Lines Between Groups

5.2.2.1 Types of Menu Items

Some menu items take effect as soon as they are chosen, but others require further information before the command can be completed. Menu items that require further information are called dialog box commands, because the additional information that they require is obtained through a dialog box. The names of dialog box commands should be followed by an ellipsis ("...").[3]

Choosing a menu item results in the initiation of a process. For most menu items (for example, those for opening files, checking spelling, and so on) this is the most natural interpretation. Some menu items, however, are best understood in terms of properties rather than processes. Menu items that change attributes of data or properties of the interface are called settings. The two types of settings are independent (M-of-N) and interdependent (one-of-N).

- Independent settings are the menu equivalents of check boxes in dialogs (see Chapter 6, section 6.2). For example, character attributes such as Bold, Italic, and Underline, if placed on a menu, would form a group of independent settings. Each of these can be changed without affecting the others, although they are related by virtue of being applicable to a single piece of text.
- Interdependent settings are the menu equivalents of option buttons in dialogs (see Chapter 6, section 6.1.2). For example, alignment properties such as Left, Center, and Right, when placed on a menu, would form a group of interdependent settings. Selecting one setting deselects the others because a particular paragraph can only have one type of alignment.

[3] Some commands may or may not lead to a dialog, depending on the state of the data or on the setting of an application option. Although it would be possible to add or remove the ellipsis dynamically for such commands, the benefit to users is probably too small to justify the extra implementation cost. Instead, the ellipsis should always be omitted in such cases. This rule ensures that users will never see an ellipsis and mistakenly expect a dialog (which typically offers opportunities to cancel a command) and then be surprised when the command is executed immediately.

Groups of related (independent or interdependent) settings should be separated from other items on the menu with a single horizontal line. A group of independent settings can also include an item that turns off all the other settings in the group.

When a setting is selected, a check mark should be placed to its left, as illustrated in Figure 5.6.

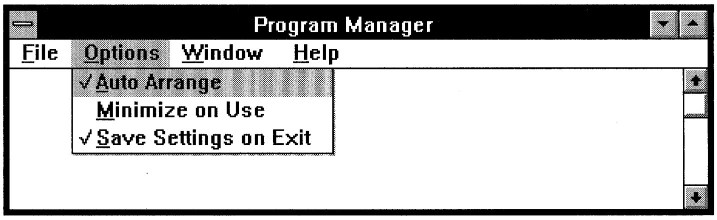


Figure 5.6 Current Settings Indicated with Check Marks

As an optional extension to this method, applications can use graphics other than check marks to distinguish between interdependent and independent settings.

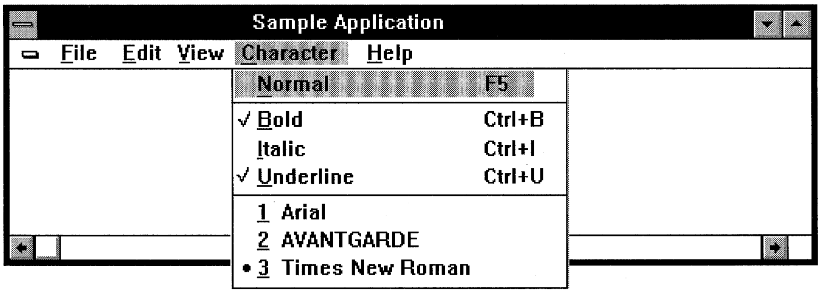


Figure 5.7 Independent and Interdependent Menu Settings

In some cases, it is appropriate to indicate the state of a setting by changing the name of the menu item rather than adding a graphic. As a rule of thumb:

- If the two states of a setting are clear and obvious opposites, a graphic mark should be used to show the states. For example, a View menu item named Ruler should show a check mark when "on" and no check mark when "off."

- If the two states of a setting are not obvious opposites, a pair of alternating menu item names should be used to indicate the two states. For example, a naive user might guess that the opposite of a menu item called "Full Duplex" is "Empty Duplex." Because of this ambiguity, applications should pair "Full Duplex" with the alternative name "Half Duplex," rather than using graphics to indicate the alternative states. The item name does not represent the current state; it indicates the state that will be obtained by choosing the item. The same mnemonic access character should be used for both names whenever possible. This can often be accomplished by incorporating the same word as part of both names. For example, "Full Duplex" and "Half Duplex" both contain "Duplex," so D can be used for the mnemonic access character (if other menu items present no conflicts).

Sometimes the state of a setting is indeterminate—for example, when a text selection contains words with different attributes. Applications should remove the graphic marks from all groups for which the settings are indeterminate.

Applications should avoid menu items whose names and functions change depending on whether the SHIFT key is held down while the menu is opened. The shifted-state functionality of such items is normally hidden, thus it often remains undiscovered and unused. If possible, the shifted-state command should be added to the menu as a regular menu item. If this cannot be done because of limited menu space, a cascading menu that includes the unshifted- and shifted-state commands can be added as a last resort. Designers should carefully weigh the drawbacks discussed in section 5.1.3.2 before using cascading menus.

5.2.2.2 Names of Menu Items

Each menu item should be represented by a descriptive name or graphic. Applications should follow these guidelines when naming menu items:

- Item names should be unique within a menu, but may be repeated in different menus to represent similar or different actions.
- Item names may be single words, compound words, or multiple words (for example, Save As).
- Each item name should have a mnemonic access character for users who choose commands with the keyboard. The guidelines for selecting a menu mnemonic access character for menu titles also apply to menu items, except that mnemonics for menu items can also be numbers. Numbers are appropriate for menu items that are graphics or that are part of a varying list of items. For example, if a menu contains a list of file names, the names may vary, but numeric access characters for each position in the list provide constant, consistent user access to the items in the list.

- An ellipsis ("...") should follow the names of commands that require more information before they can be completed.
- Item names should be arranged in a single column. Multi-column menus are generally not recommended, except for menus that contain a long list of items (for example, sizes or amounts). If multi-column menus are used, the items should be presented in a logical order (for example, numeric ascending order would be appropriate for a list of font sizes), with the sequence organized by columns rather than by rows. For example, a numeric sequence would begin with the lowest numbered item at the upper left and proceed down all items in the first column before continuing to the second and subsequent columns.

5.2.2.3 Unavailable Items

Menu items that cannot be chosen meaningfully in the current state of an application should be disabled. To preserve the stability of the menus, applications should not remove disabled commands. Instead, the names of the commands should be dimmed to inform the user that they are unavailable. Similarly, if all items on a menu are temporarily unavailable, the menu title should be dimmed but not removed. Users can open a dimmed menu to explore its contents and to obtain help on menu items. If a user chooses a dimmed command, the application may optionally provide a brief message explaining why the command is unavailable. For example, if there is no current selection and the user tries to choose the dimmed Cut command on the Edit menu, the application could provide the message "You must make a selection before choosing Cut."

5.3 Menu Operations

Users can display menus and choose menu items with the mouse, with the keyboard, or with both methods.

5.3.1 Mouse Methods

Mouse techniques for menus rely almost exclusively on mouse button 1. The single exception is that mouse button 2 is used to display pop-up menus. The following discussion assumes mouse button 1, unless otherwise indicated.

5.3.1.1 Displaying Menus

The user must display a menu before selecting an item from it. Mouse techniques for displaying menus vary according to whether the menu is a drop-down, pop-up, or cascading menu.

5.3.1.1.1 Displaying Drop-Down Menus To display a drop-down menu, the user uses the mouse to point to the menu title and presses the mouse button. This procedure highlights the title and opens the menu. If the user releases the button while the pointer is still on the menu title, the menu remains open so that the pointer can be moved to the desired menu item. Alternatively, the pointer can be moved to the item while the button is still held down. If the user moves the pointer to a second menu title before releasing the button, the first menu closes and the second menu opens. This lets users switch easily from one menu to another or see an overview of all menus by dragging the mouse across the menu bar. If the user drags the pointer out of a menu frame to any location other than a menu title, the menu remains open. However, if the user then releases the mouse button outside the menu frame, the menu closes without initiating any commands.

5.3.1.1.2 Displaying Pop-Up Menus To display a pop-up menu, the user positions the mouse pointer over an object and presses mouse button 2 (for details, see section 5.1.2.1). If the user releases the button at the button-down point (or within four pixels of the button-down point), the menu remains displayed. If the user moves the pointer and releases the button outside the menu, the menu is canceled (removed from the screen). This is similar to the behavior of drop-down menus.

A pop-up menu is also canceled when the user clicks mouse button 1 or 2 outside the menu. Consequently, there can be only one pop-up menu displayed at any time. Clicking mouse button 1 outside the menu removes the menu and sets the selection to the clicked location.[4] Clicking mouse button 2 outside the menu only removes the menu.

Generally, a pop-up menu is invoked when the pointer is over an object selected explicitly with mouse button 1. (Explicit selection techniques are discussed in Chapter 3, section 3.1.2.) However, mouse button 2 can be used to select an object and display its pop-up menu simultaneously. For example, if icon A is currently selected and the user presses mouse button 2 over icon B, icon B becomes selected and its pop-up menu is displayed. Likewise, if there is no current selection, pressing mouse button 2 both selects an object and displays its menu.

If an object is selected implicitly, pressing mouse button 2 displays its menu. Implicit selection unifies the act of selection and action into one, but does not directly change explicit selections. For example, dragging a scroll box implicitly selects an object (the scroll box) and identifies its action (move). It is possible to get a pop-up menu for the scroll box by pressing mouse button 2 while the pointer is over the scroll bar. This does not result in scrolling, but simply displays the menu.

[4] Note that this may reset an existing selection.

5.3.1.1.3 Displaying Cascading Menus To display a cascading menu, the user clicks on the parent item (this is the item with the right pointing triangle) or drags the mouse to that item. If the drag method is used, there is a brief pause before the submenu is displayed. This pause prevents the submenu from flashing when the user is dragging across the parent item on the way to another item. If the user releases the mouse on the parent item after the submenu appears, the submenu and the parent menu remain open. Alternatively, the user can drag the mouse into the submenu to choose an item from the submenu. If the user clicks on the parent item after the submenu has already been displayed, the submenu should disappear but the parent menu should remain open. The relationship of a submenu to its parent item parallels the relationship of a non-cascading menu to its menu title: The first click on the higher-level element opens the menu, and the second click closes the menu.

5.3.1.2 Choosing Menu Items

Mouse techniques for choosing menu items are the same for all types of menus. To select a menu item, the user releases the mouse button while the pointer is on the item. The release (that is, button up transition) is the second half of a mouse click. The press and release need not occur on the same item. In other words, the user may click on the item directly or may release the button there after dragging the mouse to the item from another location, either from the menu title or from another menu item. Whenever the mouse button is down and the pointer is over a menu item, the item is highlighted to indicate that it will be chosen if the mouse button is released.

If the menu is already open, the user can click a menu item to execute it. In drop-down and cascading menus, mouse button 1 is used for this purpose. In pop-up menus, the user can click mouse button 1 or 2.

5.3.2 Keyboard Methods

Table 5.1 lists keyboard techniques for drop-down and cascading menus. As explained in section 5.1.2.1, pop-up menus have no keyboard interface, except for UP ARROW, DOWN ARROW, ENTER, and the ESC key.

Table 5.1 Keyboard Techniques for Drop-Down and Cascading Menus

Key	Function
ALT	Toggles activation of menu bar (if inactive, activates it; if active, deactivates it and closes any open menu).
LEFT ARROW	▪ In menu bar, moves to previous menu; at extreme left, wraps to rightmost menu. ▪ In cascading menu, closes submenu but leaves parent menu open. ▪ In multicolumn menu, moves left one column.
RIGHT ARROW	▪ In menu bar, moves to next menu; at extreme right, wraps to leftmost menu. ▪ In multicolumn menu, moves right one column.
UP ARROW	▪ In open menu, selects item above current item; at top, wraps to bottom. ▪ In multicolumn menu, wraps to bottom of previous column.
DOWN ARROW	▪ If no menu is open, opens selected menu. ▪ In open menu, selects item below current item; at bottom of menu, wraps to top. ▪ In multicolumn menu, wraps to top of next column.
Mnemonic character	▪ When menu bar is active, chooses and opens menu with corresponding underlined mnemonic access character. ▪ When menu is open, chooses item with corresponding access character.
ENTER	Chooses selected menu item.
ESC	▪ If menu is open, closes menu but leaves menu bar active. ▪ If no menu is open but menu bar is active, deactivates menu bar.

A frequently used menu item may be assigned a keyboard shortcut (also known as an accelerator), which provides a single-step method of keyboard access, rather than the three-step method of ALT, menu title access character, and menu item access character. The shortcut key combination should be displayed at the first tab position after the name of the longest item in the menu that has a shortcut, and should be left-justified; spaces should not be used for alignment because they will not work properly with proportional system fonts. Key names should match those inscribed on the keyboard (for example, "CTRL" rather than "CONTROL"). CTRL and SHIFT key combinations should be displayed as "CTRL+*key*" (not "^+*key*"), "SHIFT+*key*", or "CTRL+SHIFT+*key*". As an optional alternative, applications can substitute graphical representations of key caps for the words "CTRL" and "SHIFT". If the application uses function keys for shortcut keys, the name of the key should appear as "F*n*", where *n* is the function key number. F is uppercase and there is no space between F and the number. Note that function key shortcuts (with or without modifier keys) are easier to localize than modifier+letter shortcuts.

When the user presses a shortcut key combination, the command is executed immediately; the menu that contains the command may appear highlighted but does not drop down.

Guidelines for assigning keyboard shortcuts are discussed in Chapter 2, section 2.2.5.

5.4 Standard Menus

Every application should include a set of standard menus, to give users a common starting point for each new application and to speed learning. The standard menus are Control, File, Edit, and Help. In the application window, the Control menu is represented at the left end of the title bar by a small box. For maximized document windows, it appears at the left end of the menu bar. On the menu bar, generally File appears first, followed by Edit (if supported). Help is generally the last menu on the bar. For application window Control menus, the access character is SPACE-BAR (that is, the user types ALT+SPACEBAR); for document window Control menus, the access character is HYPHEN. The access character for the File menu is F, Edit is E, and Help is H.

5.4.1 Control Menu

Every movable window (application, document, or dialog window) contains a Control menu at the left end of its title bar (see Figure 5.8). The Control menu is reserved for commands that provide keyboard control over the active window. Applications should avoid adding commands to the Control menu, unless no other menus are in the window and there are no suitable alternatives for providing access to the commands.

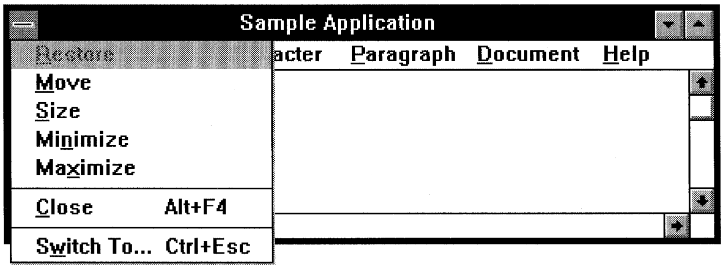


Figure 5.8 Control Menu

Control menus for dialog windows contain only Move and Close commands. The standard menu items on all other Control menus are Restore, Move, Size, Minimize, Maximize, and Close (in that order). Control menus for application windows also include a Switch To command for switching to other application windows.[5] Optionally, the Next command may also be included on the Control menu of MDI document windows to allow the user to switch to the next document window.

If one of the standard items is never used in an application, it may be removed from the menu. Tables 5.2 and 5.3 summarize Control menu items and keyboard shortcuts. Items in the first table appear on both application window and document window Control menus; items in the second table appear on only one type of Control menu. The items should appear in the order shown in the tables, with those in the first table preceding those in the second.

[5] The Control menu should not include a Run command. This command was included with applications delivered with runtime Windows, to allow the user to run other applications such as the Control Panel. With applications that depend on Windows version 3.0 or later, however, the Program Manager can be used to run other applications, so the Run command is no longer necessary.

Table 5.2 Control Menu Items Common to Application and Document Windows

Item	Function	Mouse Alternative*
Restore	Restores window from its maximum or minimum size to its previous intermediate size.	Click Restore button.**
Move	Gray "ghost" window frame appears on top of regular frame; pointer changes to window movement pointer. Arrow keys move ghost frame. ENTER completes command and redraws window at new location.	Drag title bar.
Size	Changes pointer to sizing mode pointer. Arrow keys move pointer to nearest border and change pointer to appropriate resizing pointer. Arrow keys then resize ghost window frame. ENTER completes command and redraws window at new size.	Drag window frame.
Minimize	Replaces window with its minimized representation (usually an icon). If window is an application window, all supplemental windows (document windows, dialogs, floating palettes, toolboxes, and so on) are hidden.	Click Minimize button.
Maximize	Expands window to maximum size.	Click Maximize button.**
Close§	Closes window; displays a dialog asking user whether to save data if window contains unsaved changes. If window is an application window, all supplemental windows (document windows, dialogs, floating palettes, and so on) are also closed.§§ If window is a dialog window, committed transactions are accepted and uncommitted transactions are not accepted.	Double-click Control menu.

Note: Mnemonic access characters for menu items are underlined.

* The mouse can also be used to select menu items from the Control menu.

** Some applications support a double-click on the title bar as an equivalent shortcut for Maximize or Restore. This shortcut may conflict with future system-defined behavior for title bars and should not be documented.

§ The recommended keyboard shortcut for Close is ALT+F4 for dialog and application windows and CTRL+F4 for document windows. CTRL+ESC is the shortcut for Switch To (see Table 5.3). No other keyboard shortcuts are currently recommended for commands in this table.

§§ As a rule, the Close command removes windows from the screen, whereas the Exit command (on the File menu) ends the active application. Close is functionally equivalent to Exit when it is used to close the main application window.

Table 5.3 **Additional Control Menu Items**

Item	Function	Mouse Alternative*
Switch To	Displays dialog that contains a list of running applications so user can switch to another application. This command is only available on application window Control menus. (Keyboard shortcut: CTRL+ESC.)	Click on desired application window; or double-click on desktop to show task list.
Next	Switches among open document windows and icons. This command is only available on document window Control menus. (Keyboard shortcut: CTRL+F6.)	Click on desired window.

Note: Mnemonic access characters for menu items are underlined.

* The mouse can also be used to select menu items from the Control menu.

5.4.2 File Menu

Applications that use data files should include a File menu, which provides all the commands the user needs to open, create, and save files. Figure 5.9 shows a standard File menu. If an application has a File menu, it should be the first menu on the menu bar (except for the Control menu for a maximized document window in MDI applications). The mnemonic access character for the File menu is F.

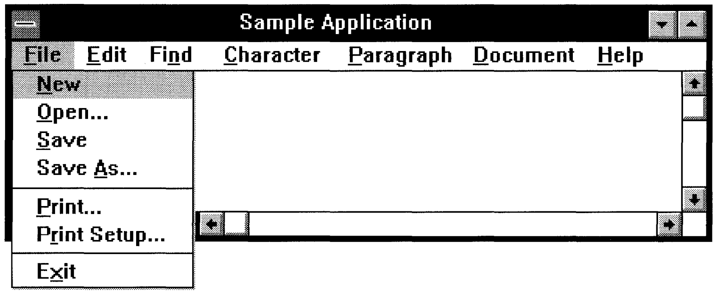


Figure 5.9 Standard File Menu

5.4.2.1 Standard Items

The File menu contains the following items in the order listed (other items may be interspersed): New or New (always the first item), Open, Save, Save As, Print or Print, Print Setup,[6] and Exit (always the last item). In applications that deal with multiple open document files (MDI), the Close command may optionally be added (after Open) to close the document file displayed in the active window. Table 5.4 lists the standard File menu items.

Table 5.4 Standard File Menu Items

Item	Function
New or New...*	Creates new document with default name such as Untitled, Document*n*, Sheet*n*, or Image*n*. Applications that can create different types of documents (for example, spreadsheets and charts) should display the standard dialog to allow user to choose type. Most common type should be default choice, and user-supplied file name should not be required; user should be able to create new document quickly by simply pressing OK after dialog appears.**
Open*	Leads to standard dialog that allows the user to open existing files, which may be located in different directories or on different storage devices.**
Save	Saves file displayed in active window. If file has never been saved, displays Save As dialog so that user can specify file name.
Save As	Displays standard dialog that allows user to save current file under a new name, in the same or different directory.
Print or Print...	Prints active document at currently selected printer; if appropriate, first displays standard dialog that allows user to set print options (for example, page range and number of copies) before printing.
Print Setup***	Displays standard dialog that allows user to switch from one printer connection to another and to specify settings for selected printer.
Exit*	Terminates application and closes all windows belonging to it. (In applications that do not include File menus, the Exit command should be the last command on the leftmost application menu, after the Control menu.)

Note: Mnemonic access characters are underlined. No standard keyboard shortcuts are currently recommended for File menu commands.

* If executing this command would cause information to be lost, the application should display a message dialog prompting user ("Save changes to <*file name*>?") before executing the command. This dialog should contain the Yes, No, and Cancel buttons for user response. For further information about message dialogs, see Chapter 7, section 7.1.4.

** If the New or Open command results in closing an existing file that has unsaved changes, the user should be queried to save the changes before the New or Open dialog is displayed.

*** Note that the name of this dialog is Print Setup, not Printer Setup. The latter name is now reserved for the system dialog that changes global printer settings.

[6] Not "Printer Setup"; see note to Table 5.4.

5.4.2.2 Common Optional Items

In addition to the standard items in Table 5.4, the File menu frequently includes optional items, such as the most recently used (MRU) list of files.

5.4.2.2.1 MRU List The MRU list provides quick access to recently used files, without requiring the user to work through a dialog box to specify the location of the files. When the user chooses a file name from the MRU list, the file is opened immediately. (If the file is already open, a new window should not be opened; instead, the existing window for the file should be brought to the front.) If present, the MRU list should precede the Exit command, which is always the last menu item so that it can be easily located.

The number of files on the MRU list may vary between applications but should remain constant within an application.[7] The appropriate length for the list depends on the length of the File menu but should not be less than three or more than eight. When the user opens a file (or saves a new document in a file), the file name is placed at the top of the list, next to the number 1, which can be used as a mnemonic access character. The numbered list continues through progressively less recently opened files, with the least recently opened file at the bottom. The MRU list entry may also display all or part of the pathname, but it is preferable to display only as much of the pathname as is necessary for the user to distinguish between similar file names. Internally, however, the full pathname of the file should be recorded so that the file can be opened immediately without asking the user for storage information.

5.4.3 Edit Menu

The Edit menu follows the File menu on the menu bar. The mnemonic access character for the Edit menu is E. Figure 5.10 shows the standard Edit menu items.

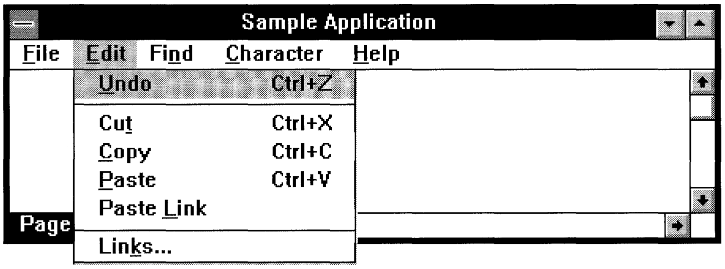


Figure 5.10 Standard Edit Menu

[7] Except for the first few times an application is used, when there may not be enough previously used files to fill the list.

5.4.3.1 Standard Items

The standard Edit menu provides commands for:

- Reversing the last action that altered the user's data (Undo).
- Moving, copying, and linking data and objects (Cut, Copy, Paste). These operations rely on the system clipboard, which is a common data buffer used to move data within and between applications.

Undo is always the first command on the Edit menu. The remaining standard items (which may be interspersed with others) appear in the following order: Cut, Copy, and Paste. Paste Link and Links are added in applications that support linking and embedding (see Chapter 9).

Table 5.5 lists the standard Edit menu items.

Table 5.5 Standard Edit Menu Items

Item	Shortcut	Function
Undo	CTRL+Z*	Reverses last action that altered user's data. (Applications may also provide more extensive Undo support.) Name of last action appears after "Undo" (for example, Undo Cut). If action can't be reversed, command is dimmed; optionally, name may also change to Can't Undo.
Cut**	CTRL+X*	Transfers selected data to clipboard and deletes data from current window or field.
Copy**	CTRL+C*	Copies selected data, objects, or references to clipboard; marks current selection for subsequent use in Paste Link operations.
Paste***	CTRL+V*	Pastes data, object, or reference from clipboard into document at current insertion point. Replaces current selection (if any).
Paste Link***	(none)	At insertion point in current document (that is, destination), creates link to item previously marked in source (by means of Copy).
Links...	(none)	Displays Links dialog for changing link properties and accessing linked objects.

Note: Mnemonic access characters are underlined.

* The shortcuts for Undo, Cut, Copy, and Paste are new in Windows version 3.1. For backward compatibility, we recommend that applications designed to run under Windows version 3.0 also support the old shortcuts: Undo = ALT+BACKSPACE, Cut = SHIFT+DEL, Copy = CTRL+INS, and Paste = SHIFT+INS. However, the old shortcuts should not be documented in user manuals or listed on the Edit menu.

** Command should be dimmed if there is no selection.

*** Command should be dimmed if the clipboard is empty.

5.4.3.2 Common Optional Items

Common optional items on the Edit menu include Paste Special, Repeat <*Action*>, Find, Replace, Clear, and Delete. (The Paste Special command is discussed in Chapter 9, section 9.3.2.4.)

5.4.3.2.1 Repeat <*Action*> Repeat <*Action*> repeats the most recent action. The name of the action appears after "Repeat" (for example, Repeat Paste). If present, the Repeat command should follow Undo on the Edit menu.

5.4.3.2.2 Find and Replace Applications that provide Find and Replace commands for text search and substitution should place those commands on the Edit menu (see Chapter 8, section 8.3).

5.4.3.2.3 Clear vs. Delete The Clear command applies only to container objects such as text boxes and grid cells; it removes the contents of the object without removing the object itself. The contents are not placed on the clipboard.

The Delete command is equivalent to the DEL key; it deletes the current selection without placing it on the clipboard. If the user selects a container object such as a text box or a grid cell, Delete removes both the object and its contents. The Clear command, on the other hand, deletes only the contents of a selected object. If the user selects only the contents of the object, Delete has the same effect as Clear; it removes the contents.

Delete is the more common of the two commands because it can apply both to containers and non-containers, and because it can be used to accomplish the same purposes as Clear. If an application provides an interface in which a container and its contents can easily be distinguished, the application need only implement the Delete command. If the distinction between the container and its contents is blurry and the user needs a quick way to clear the contents of the container, the application should also implement the Clear command.

5.4.4 Help Menu

The Help menu is always the rightmost menu of the menu bar (see Figure 5.11), immediately following the next-to-last item.[8]

[8] Some applications currently place the Help menu at the extreme right of the menu bar. The current recommendation is that the Help menu immediately follow the next-to-last menu item, for three reasons: (1) to increase the accessibility of the Help menu; (2) to decrease the likelihood of pressing the Maximize or Minimize buttons by mistake when trying to access Help; and (3) to make the space at the extreme right of the menu bar available in MDI applications for a Restore icon for maximized child windows.

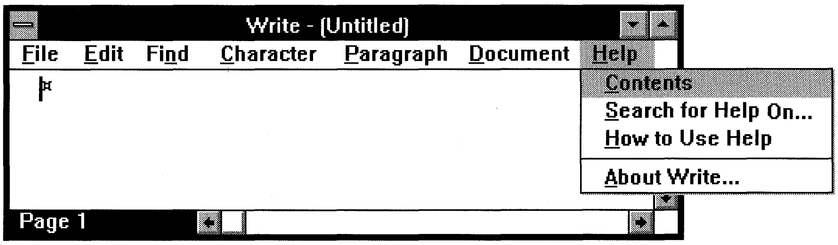


Figure 5.11 Help Menu in Windows Write

5.4.4.1 Standard Items

The Help menu contains the items Contents, Search for Help On, How to Use Help, and About <*Application-Name*> in the order listed. Other items may be interspersed.

Table 5.6 Standard Help Menu Items

Item	Function
Contents	Opens Help window and displays list of main topics.[9]
Search for Help On	Opens Help window and displays dialog that allows the user to search for Help topics containing specific keywords.
How to Use Help	Opens Help window and displays instructions for using Help.
About <*Application-Name*>	Displays standard dialog box containing application name, version number, copyright message, icon, serial number, and user's name; plus optional additional information such as amount of available workspace in memory or amount of storage space on active storage device. For more information on the About dialog, see Chapter 8, section 8.6.

Note: Mnemonic access characters are underlined.

[9] This item was called "Index" in Windows version 3.0. "Index" is now an optional item that leads to a full alphabetic index rather than a selective list of main topics.

5.4.4.2 Common Optional Items

The Help menu may also include additional items that access broad subcategories of Help (for example, Commands, Procedures, Keyboard), or that provide equivalents for commands used by other products in the same category. If the application includes an online tutorial, the menu can include a Tutorial item that starts the tutorial. This item should follow Search for Help on and precede How to Use Help; the mnemonic access character is T.

5.5 Common Optional Menus

In addition to the standard menus, applications frequently include the View and Window menus.

5.5.1 View Menu

The View menu includes commands for changing the user's view of the data in the window; these commands change only the view, never the data itself. For example, in a word-processing application, the View menu might include commands for switching between text and outline view; in a graphics application, the View menu might include Zoom In and Zoom Out commands.

The View menu may also include commands for controlling the display of interface elements, such as status bars and control bars that are a fixed part of the application window. (The display of movable control bars should be controlled by commands on the Window menu.) The View menu can also include commands for displaying specialized window panes (such as or annotation panes) or general-purpose panes. Applications that support general-purpose panes should provide a Split command on the View menu.[10] (For additional information on the Split command, see Chapter 4, section 4.2.5.) If present, the View menu typically follows the Edit menu.

5.5.2 Window Menu

The ability to open multiple document windows adds a requirement for additional menu commands that manipulate document windows as whole entities (for example, commands that place document windows in an orderly arrangement). MDI applications should include a Window menu for these commands. This menu should be the last menu before Help. A sample Window menu is illustrated in Figure 5.12.

[10]This command can also be placed on the Window menu.

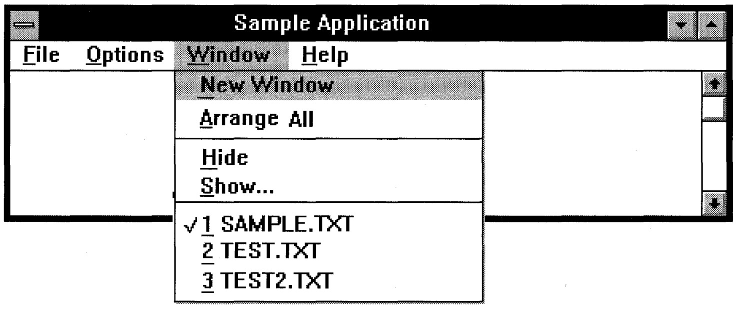


Figure 5.12 Sample Window Menu

5.5.2.1 Standard Items

Because document windows in MDI applications can overlap or even completely obscure each other, activating them by clicking on them with the mouse can become difficult or impossible. Consequently, the Window menu should contain a numbered list of open windows to allow the user to select and activate any window easily. When the user selects a document window from the Window menu, the currently active window is deactivated, and the selected window is activated and displayed on top of the other document windows.

The list of windows appears after all other menu items and is separated from them by a horizontal line. Each list item is preceded by a number, which serves as its mnemonic access character. An active window is indicated by a check mark before its number. The list shows the first nine open document windows. When more than nine document windows are open, the list contains a More command (with M as its mnemonic access character), which displays a dialog box listing all the open document windows, including those that were already shown on the Window menu. The application window, secondary windows, and dialog boxes never appear on the list. The order of windows in the list is the order in which the user opened the windows. When the user closes a document window, the list is renumbered.

5.5.2.2 Common Optional Items

Common optional items on Window menus include New Window, window arrangement commands, and the list of open windows. Applications that allow windows to be split into panes should include a Split command on the View menu. If the application has no View menu, the command should be placed on the Window menu.

5.5.2.2.1 New Window The New Window command creates a duplicate window that opens another view on the active document.[11] When multiple windows are open for a single document, the title bar of each window indicates the document name and the number of the window. For example, if a spreadsheet application has a chart window named ChartA open, and the user selects the New Window menu item, the first document window containing ChartA shows "ChartA:1" in its title bar. The second document window containing ChartA shows "ChartA:2" in its title bar, and so on.

5.5.2.2.2 Window Arrangement Commands Window arrangement commands are optional but highly recommended. As users open new document windows and resize old ones, the windows can overlap and eventually obscure or hide one another. Consequently, the user needs a way to arrange the windows so that all document windows are at least partially visible. This can be done in a variety of ways:

- The Tile menu item arranges the document windows in tiled style, with each window displayed in its own space within the application window.
- The Cascade command arranges the document windows in overlapping style, like offset note cards.
- The Arrange command (Arrange... or Arrange All) can be used for more general window arrangements.

You can also include arrangement commands that specifically affect minimized windows.

[11]The New Window command should not be confused with the File New command, which creates a new document.

Dialog Box Controls

Chapter 6

This chapter discusses controls that appear primarily in dialog boxes (and in control bars, which are essentially modeless dialogs): buttons, check boxes, lists, and edit controls. Applications may also contain custom controls, but widespread use of such controls defeats the benefits of consistency. Before deciding to use custom controls, application designers should carefully consider whether existing controls can be used instead.

6.1 Buttons

Buttons are graphical controls that initiate actions and change data object properties or the interface itself. Users can choose buttons by clicking them with the mouse. Users can also choose most buttons using the keyboard, either through mnemonic access characters (for example, ALT+O for an Options >> button in a dialog) or keyboard shortcuts (for example, ESC for the Cancel button in a dialog or CTRL+B for the Bold button on a control bar).

Three-dimensional buttons that set properties should be shown in the depressed position whenever the specified property is in effect. For example, if a control bar contains a graphical button for turning on the Bold text style, the button should be shown in the depressed position whenever the current selection is in the Bold style. Similarly, a button that causes a drop-down control to open should remain depressed as long as the control is open.

When a button is inactive (that is, when the user cannot choose it because the associated action or setting is unavailable), the button label should be dimmed.

The standard interface uses two types of buttons: command buttons and option buttons.

6.1.1 Command Buttons

A command button[1] is a rectangular shape containing a label that specifies the action or response represented by the button, for example, "Assign" or "Cancel". Figure 6.1 shows a dialog containing several command buttons. Button labels are usually textual, but in control bars (for example, ribbons, palettes, or toolboxes), graphical labels may also be used. Graphical labels are particularly appropriate when the function of the button cannot be concisely represented with a textual label but can be summarized by a single picture. For example, buttons for setting text alignment can be labeled with miniature representations of appropriately aligned text instead of long textual labels, such as "Align Text with Left Margin." Graphical labels are also useful in reducing clutter when many buttons are presented in a small space, as in toolboxes.

[1] Also known as a push button.

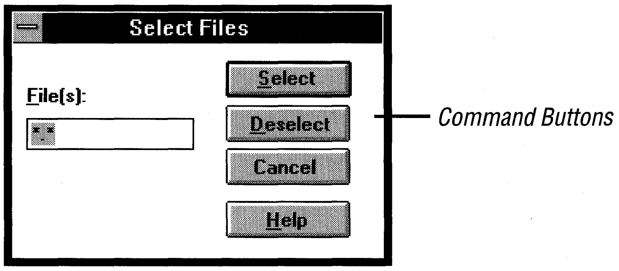


Figure 6.1 Command Buttons in Dialog Box

The user can choose a command button by clicking the mouse while the pointer is over the button. The action associated with the command button is initiated when the mouse button is released.[2] If the function of a command button changes depending on the state of the application, its label should change accordingly. For example, in transactional dialogs (also known as multiple-action dialogs), the user is allowed to perform several actions before closing the dialog. As soon as the user performs any action that cannot be canceled, the label of the Cancel button should change to Close, to reflect that the action cannot be undone.

A dialog box may contain several types of command buttons:

- Command buttons that initiate an action.
- GoTo[3] buttons, which close the current dialog box and open a related one.
- GoSub buttons, which open a related dialog box on top of the current dialog box without closing the current dialog box.
- Unfold buttons (such as Options >>), which expand the dialog to include additional options.

One command button in a dialog may be designated as the default. This button is pushed automatically when the user presses ENTER (see Chapter 7, section 7.3.2).

[2] Auto-repeat buttons (for example, scroll arrow buttons) are exceptions to this rule; their action is initiated as soon as the mouse button is pressed and continues until the mouse button is released.

[3] GoTo, GoSub, and Unfold are used descriptively; they are not the actual button labels. Applications should choose appropriate labels for these buttons.

6.1.2 Option Buttons

An option button[4] represents a single choice in a limited set of mutually exclusive choices. Accordingly, in any group of option buttons, the user can only select one at any time. Option buttons are represented by circles, as shown in Figure 6.2. When an option button choice is selected, the circle is filled; when the choice is not selected, the circle is empty. If the number of option buttons in a group exceeds four, the buttons can be replaced by a standard or drop-down list to save space. If space is not at a premium, however, option buttons provide easier access to choices.

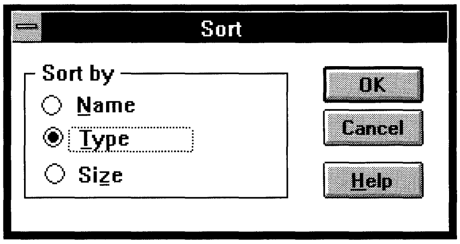


Figure 6.2 Option Buttons

A group of option buttons can be used to choose among a fixed set of attributes for a selection. Whenever the user makes a new selection, the option button group should indicate which attribute currently applies to the selection; that is, the option button corresponding to the current attribute should be filled, and the other option buttons should be empty. If the current selection is heterogeneous with respect to the set of attributes (that is, if more than one attribute is represented in the selection), all the option buttons in the group should be empty, as shown in Figure 6.3. Choosing any button applies the associated attribute to the entire selection.

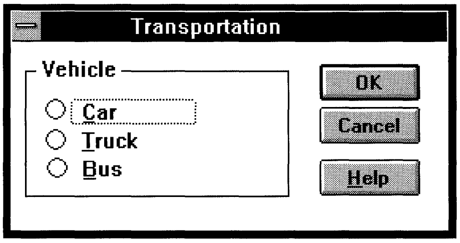


Figure 6.3 Option Buttons for Heterogeneous Selection

[4] Also known as a radio button.

In dialog boxes, double-clicking on an option button is an optional shortcut for selecting the button and choosing the default command button in the dialog, thus closing the dialog.

An application can also implement value sets, which are groups of adjacent rectangular option buttons in which the labels are contained inside the buttons. The labels may be graphical or textual. Value sets are particularly appropriate for options that can best be represented by graphical labels (for example, colors, patterns, or drawing tools), but they are also useful for options with textual labels that are short enough to fit within a small rectangle. When the user selects an item in a value set, the item should denote selection.

6.2 Check Boxes

Check boxes control individual choices that are either turned on or off. When the choice is turned on, the check box shows an X in it (see Figure 6.4). When the choice is turned off, the check box is blank. The user can toggle the state of a check box by clicking on the box or the label with the mouse or by pressing the Select key (SPACEBAR) when the check box has the focus.

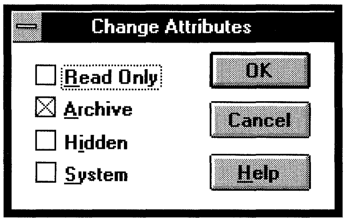


Figure 6.4 Check Boxes

Check boxes can be grouped, but grouping does not prevent the user from turning the check boxes on and off in any combination. Two exceptions to the independence of check boxes are allowed:

- If it is desirable to provide a quick way to turn off all check boxes in a group, an additional check box that performs that function can be added to the group.

- Suppose that a large group of related, but in most cases independent, options (for example, type style options in a character properties dialog) contains two dependent options (for example, Uppercase and Small Caps). Applications should consider using option buttons or drop-down lists in such cases, but those alternatives do have some disadvantages. Option buttons force a third choice to be added (for example, Normal) to indicate that neither of the two options applies. Drop-down lists not only force this third choice to be added, but they hide all but one of the choices and make them harder to access. Accordingly, in crowded and complex dialogs, it is permissible to use check boxes for two dependent options that are in the middle of other independent options.

Check boxes may be used to set properties of a selection. As with option buttons, check boxes should correctly reflect the properties of each new selection. If the selection is heterogeneous, the check box for each heterogeneous property should be filled with a gray pattern, as shown in Figure 6.5. Grayed check boxes can be cycled through three states. Clicking a grayed check box once turns on the associated property for the entire selection (and places an X in the box). Clicking the check box again turns off the associated property for the entire selection (and removes the X). Clicking the check box a third time returns the selection to its original heterogeneous state (and restores the original gray pattern in the box).

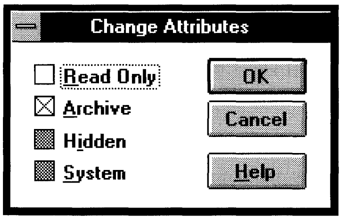


Figure 6.5 Check Boxes for Heterogeneous Selection

As with buttons, when a check box is inactive, its label should be dimmed.

6.3 List Boxes

List boxes are used to display choices for the user. The choices may be represented by text, color, or graphics (bitmaps or metafile graphic objects). Selected choices are highlighted according to the guidelines described in Chapter 3, section 3.1.1.1.

If a particular choice is not available in the current context, it should generally be omitted from the list. For example, if a certain point size is not available for the currently selected font, that size should not be displayed in the list. However, if it is important to communicate to the user both the existence and the current inaccessibility of a list item (for example, a file), the item can be dimmed rather than omitted. For example, in the standard File Save As dialog (see Chapter 8, section 8.1.2), the files in the current directory are displayed in the file name list so that users know which files already exist, but the names are dimmed and unselectable to reduce the likelihood of accidentally overwriting an existing file.

When a list box is inactive, its label should be dimmed. As an optional but recommended extension, applications can also dim textual list items when a list is inactive.

List boxes can be classified according to whether they permit the selection of one or multiple items. Single-selection lists are the most common.

6.3.1 Single-Selection List Boxes

The two types of single-selection list boxes are standard and drop-down.

6.3.1.1 General Characteristics

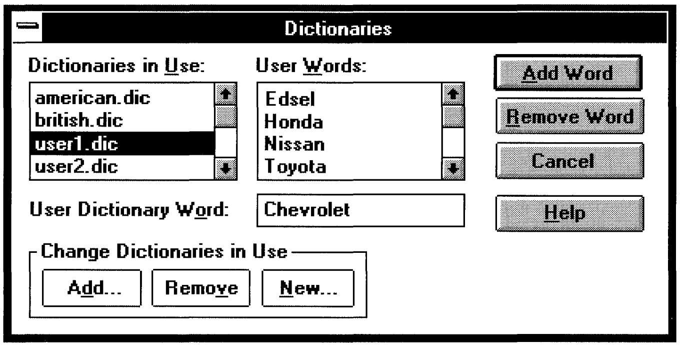
Because single-selection lists allow the user to select only one item, the items in the list are functionally similar to option buttons. Applications should use single-selection lists rather than option buttons in these cases: when the size or composition of a set of choices is variable, when the set of choices is large (more than four or five items), or when space or layout considerations make option buttons impractical.

The currently selected item in a single-selection list should be highlighted. The arrow keys can be used to move the selection highlight up and down in the list. The mouse may also be used to select items and to scroll the list with the scroll bar. If scrolling is not possible, either because the list is inactive or because all items in the list are already visible, then the scroll arrows should be dimmed, the scroll box should be removed, and the interior of the scroll bar shaft should be changed to the background color of the dialog box.[5] (For an example, see the illustration of the File New dialog in Chapter 8, Figure 8.3.) List boxes also support automatic scrolling (see Chapter 4, sections 4.3.6.1.2 and 4.3.6.2.2).

If the user presses a character key while a single-selection list box has the focus, the list scrolls to the first item that begins with that character (if there are any such items), and the item is selected. If no matching item is found, the list does not scroll, nor does the highlight move.

6.3.1.2 Standard Single-Selection Lists

Standard lists always remain the same size: tall enough to show from three to eight choices, depending on the available height in the dialog box, and several spaces wider than the average width of the items in the list. When the list contains an item that is too wide for the list, a horizontal scroll bar may be optionally placed at the bottom of the list. Figure 6.6 shows examples of standard lists.



american.dic
british.dic
user1.dic
user2.dic

Edsel
Honda
Nissan
Toyota

Chevrolet

Figure 6.6 Standard Single-Selection List

In dialog boxes, double-clicking on an item in a single-selection list is an optional shortcut for selecting the item and choosing the default command button in the dialog, thus closing the dialog. This behavior parallels the shortcut for option buttons described previously.

[5] This is a new recommendation in Windows version 3.1.

If the choices in a list represent possible attribute values for a selection, the current value should be selected when the list is first displayed. If the selection is heterogeneous, no value should be selected.

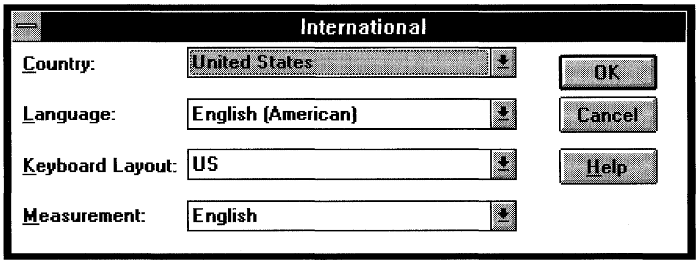
When space is limited, standard lists may be replaced by drop-down lists.

6.3.1.3 Drop-Down Single-Selection Lists

Like a standard single-selection list, a drop-down single-selection list has a fixed width, which should be several spaces wider than the average width of the items in the list. As in a standard list, a horizontal scroll bar may be added at the bottom of a drop-down list if a particular item is too wide for the list.

Unlike a standard list, however, a drop-down list has two possible heights. When closed, a drop-down list is only tall enough to show one item. When opened, a drop-down list should be large enough to show three to eight items, just like a standard list. If the drop-down list contains more than eight or a variable number of items, it should have a vertical scroll bar. A drop-down list may drop outside the dialog box, but if there is not enough space on the screen for the list to drop down, it should open upward.

Figure 6.7 shows several drop-down lists. Note that the drop-down arrow button abuts the end of the associated list box. The lack of a gap between the list and the drop-down arrow is a visual distinction between a drop-down list and a drop-down combo box, which does have a gap (see section 6.4.1.2).



Country: United States

Language: English (American)

Keyboard Layout: US

Measurement: English

Figure 6.7 Closed Drop-Down Single-Selection Lists

When a drop-down list is open (see Figure 6.8), the list extends to the right edge of the drop-down arrow button, to allow the user to drag into the list.

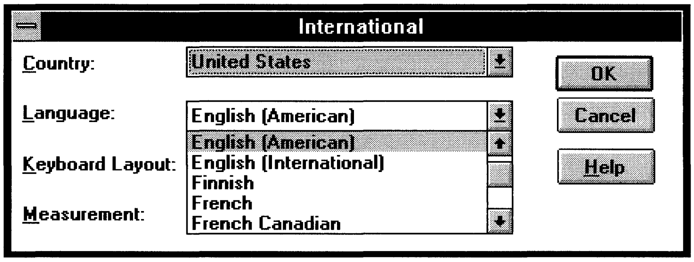


Figure 6.8 Open Drop-Down List

A drop-down list can be toggled between the closed and open state by:

- Clicking on the drop-down arrow.
- Pressing ALT+DOWN ARROW.
- Pressing ALT+UP ARROW.
- Pressing or clicking on the field at the top of the list.

When a list has the focus and is closed, pressing DOWN ARROW opens it. Scrolling is permitted only when the list is open.[6] Any new item selection made while the list was open is accepted and displayed in the list field.

An open list can be closed by:

- ALT+DOWN ARROW
- ALT+UP ARROW
- TAB or other navigation method
- ENTER (which does not close the dialog)

If the choices in a drop-down list represent possible attribute values for a selection and the selection is heterogeneous, no value should be displayed when the list is closed, and no value should be selected when the list is open.

[6] Windows version 3.0 allowed users to scroll and autoselect items from a closed drop-down list. This behavior was changed in version 3.1 to avoid delays associated with updating and processing information. For example, in a database application, changing the selection in a drop-down list may change the current query and thereby require a new database search. Similarly, scrolling the Drives drop-down list in the File Open dialog box would require the files list to be updated with each new drive selection.

6.3.2 Extended-Selection and Multiple-Selection List Boxes

Although most list boxes are single-selection lists, occasionally it is useful to let the user select more than one item. This functionality can be obtained with either extended-selection or multiple-selection list boxes.

Extended-selection lists should support the mouse and keyboard techniques for contiguous and disjoint selection described in Chapter 3, sections 3.1.2 and 3.1.3.[7] For example, pressing SHIFT+F8 turns on Add mode, which allows the focus indicator (the dotted box) to move independently of the selection highlight. Pressing SPACEBAR toggles the selection state of the item that has the focus and sets a new anchor point there, without deselecting other items. SHIFT+navigation or SHIFT+click propagates the selection state of the item at the current anchor point. Note that when no modifier keys or special modes are in effect, extended-selection lists behave just like single-selection lists.

Extended-selection lists are particularly useful under the following conditions:

- The user may want an action (for example, printing or deleting) to apply to more than one list entry at a time.
- Contiguous list entries are related in ways meaningful to the user. Because related entries are contiguous, extending a selection with the SHIFT key can easily pick out meaningful groups. This type of ordering frequently arises when the entries represent objects that have been created and named by the user.

When users want to select several entries from a list but the entries are not grouped in a way that makes extended selection useful, multiple-selection lists can be used. Whereas extended-selection lists provide easy range selection, multiple-selection lists are optimized for disjoint selection. The suggested appearance of items in a multiple-selection list includes a check box preceding each item, as shown in Figure 6.9.

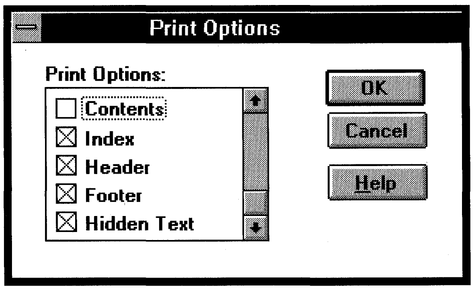


Figure 6.9 Multiple-Selection List[8]

[7] The directory windows in the Windows File Manager are examples of extended selection in lists.

[8] The multiple-selection list is not a predefined Windows program control.

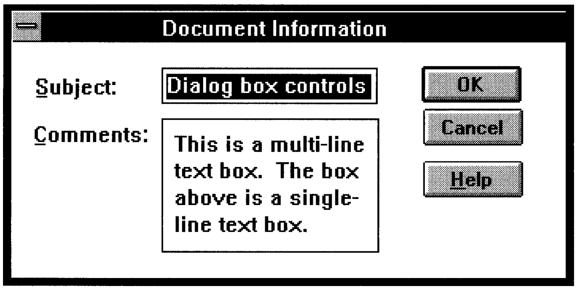
The two main reasons for using check boxes rather than check marks are:

- Check boxes are a more familiar and widely used part of the interface.
- Even if no items in the list have been selected, the boxes are always present to indicate that the list is a multiple-selection list. This advantage outweighs the slight increase in visual clutter caused by the always-present check boxes.

When the focus moves to the list, the dotted focus rectangle surrounds the first item name (that is, the first check box label) but not the check box itself. Pressing the SPACEBAR toggles the state of the check box without affecting other items. The user can also toggle a check box by clicking on it (or its label) with the mouse. The behavior of check boxes in multiple-selection lists matches the behavior of free-standing check boxes (see section 6.2): If an item is selected, its check box has an X; if it is not selected, its check box is empty. Reverse video is not used to indicate selection.

6.4 Text Boxes

Text boxes are edit controls into which the user types information, as shown in Figure 6.10.[9] The user can accept the current text, edit it, delete it, or replace it. The LEFT ARROW and RIGHT ARROW keys move the insertion point within the text in a text box; when combined with CTRL, the same keys move the insertion point to the beginning and end of the text. Mouse and keyboard selection of text within text boxes follows the standard methods described in Chapter 3, sections 3.1.2 and 3.1.3.



Dialog box controls

This is a multi-line text box. The box above is a single-line text box.

Figure 6.10 Text Boxes

[9] Text boxes are not the same as text frames, which are discussed in Chapter 3, section 3.9.

Most text boxes are only one line tall, but applications may also use multi-line text boxes, such as the Comments box in Figure 6.10. In multi-line text boxes, data that is too long to fit on one line may either wrap to the next line or extend beyond the right boundary of the box. Both single-line and multi-line text boxes should support automatic keyboard and mouse scrolling (see Chapter 4, section 4.3.6), to allow hidden data to be brought into view. Multi-line text boxes may also include scroll bars. To insert a carriage return in a multi-line text box in a dialog, the user can press CTRL+ENTER, because ENTER alone would perform its usual function of choosing the default button[10] and closing the dialog.

To activate a text box, the user either presses the access key combination or presses the TAB key to move the active control indicator into the text box. When a text box is activated in this way, its contents should be highlighted, and an insertion point should be placed inside the box at the end of the highlighted contents. If the text box is activated with a mouse click, an insertion point should be placed in the contents of the box as near as possible to the click location, but the contents should not be highlighted.

If a text box is inaccessible (for example, because it is associated with an unselected option), the label of the box should be dimmed.

In certain situations, fixed-length auto-exit text boxes can be used to speed up data entry. As soon as such a box is filled (that is, as soon as the last character is typed), the focus moves to the next control. For example, a five-character auto-exit text box could be used to facilitate entry of zip codes. As soon as the fifth digit is typed, the focus moves to the next control. Because the automatic focus shift can be unexpected and disconcerting, auto-exit text boxes should be used sparingly. In general, they are best limited to situations involving extensive data entry.

It is not unusual to link controls in a dialog box, such as a text box and a list, so that user interaction with one control also affects another. The standard File Open dialog box demonstrates the benefits of integrating a text box with a list box. This type of linking can be taken one step further by merging the two controls into one, called a combo box.

6.4.1 Combo Boxes

A combo box is a text box with an attached, integrated, and interdependent list. Combo boxes are useful when the application requires user input and can display a list of possible responses. The user can type a response in the text box if the correct one is not available in the list. Combo boxes can be classified into two categories, standard and drop-down, according to the type of list it includes.

[10]ENTER can be used if there is no default button.

6.4.1.1 Standard Combo Boxes

Standard combo boxes include a text box and a standard list, as shown in Figure 6.11. If the list is *never* expected to display more entries than can be shown at one time, the scroll bar may be omitted. The left border of the list is indented from the edit control by the width of a numeric digit in the character set. The entries in the list should be in alphabetic order unless there is a compelling reason to use a different order. For example, a list of file names should be in alphabetic order, but a list of dates should be in chronological order.

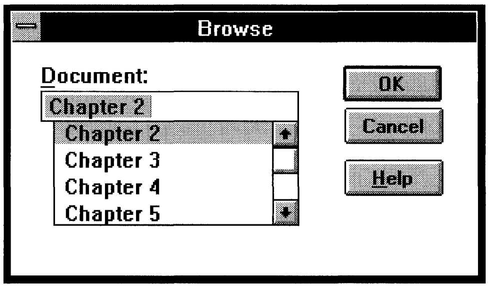


Figure 6.11 Standard Combo Box

When a combo box has the focus, the UP ARROW and DOWN ARROW keys move the selection up and down in the list and put the selected item into the text box. The LEFT ARROW and RIGHT ARROW keys move the cursor left and right in the edit field. The user can press character keys to enter characters in the text box. The list scrolls to the first item that begins with the characters in the text box. When the target item is brought into view, however, it is not automatically selected (as it would be in an independent list). Pressing the DOWN ARROW key selects the target item and places it in the text box.

When space is at a premium, standard combo boxes may be replaced by drop-down combo boxes.

6.4.1.2 Drop-Down Combo Boxes

A drop-down combo box includes a text box and a drop-down list. Figure 6.12 shows two drop-down combo boxes, one for changing fonts and the other for changing point size.

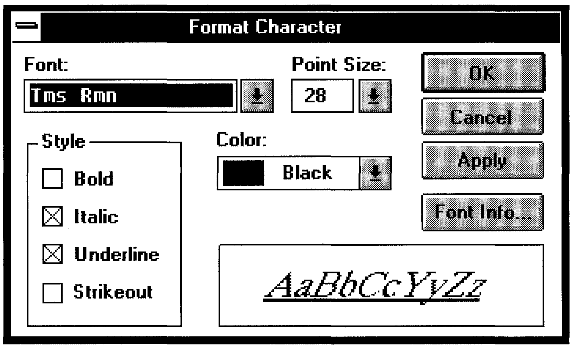


Figure 6.12 Two Drop-Down Combo Boxes

Note the gap between the end of the text box and the drop-down arrow button. This gap provides a visual distinction between a drop-down combo box and a drop-down list (such as the color control in Figure 6.12), which does not have a gap. When the list portion of a drop-down combo box is open, the list should extend to the right edge of the drop-down arrow button, to allow the user to drag into the list.

6.4.2 Spin Boxes

Spin boxes are specialized text boxes that accept only a limited set of discrete, ordered input values. A spin box consists of a text box with a pair of arrows (an upward-pointing arrow above a downward-pointing arrow) attached to the right side of the text box, as shown in Figure 6.13.

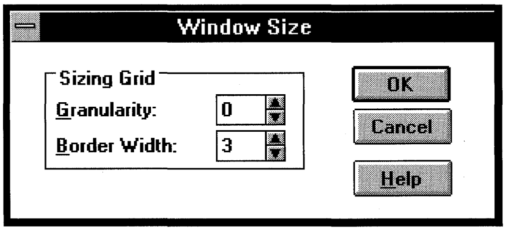


Figure 6.13 Spin Boxes

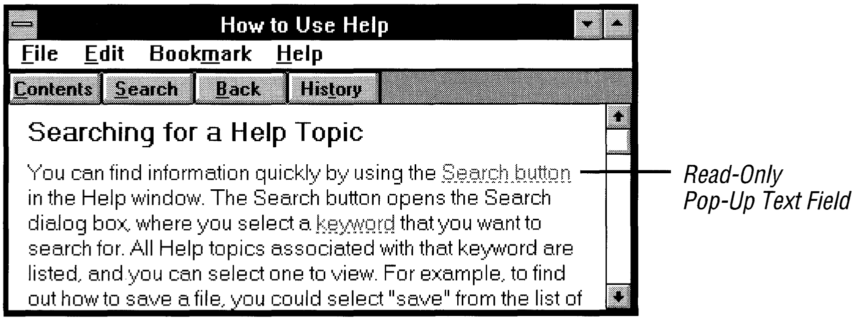
The user can type a new value into the text box, click the UP ARROW key to increase the value, or click the DOWN ARROW key to decrease the value. In effect, the arrows function like scroll arrows for a hidden list that is sorted in descending order (in contrast to actual list controls, in which entries normally should be sorted in ascending order).

Spin boxes may be used to display values that consist of several subcomponents (for example, time, which consists of hours, minutes, and seconds). In such cases, the text box is divided into several subfields and the subfields are separated by suitable separators (for example, in the U.S., ":" for time and "/" for date). The arrows affect the selected subfield; if no subfield is selected, the arrows affect the subfield representing the smallest unit of measurement.

A value typed into the text field of a spin box should be validated either immediately or as soon as the user navigates away from the spin box. For example, if the user types a letter into a spin box that is only meant to accept numeric values, the application may beep (or alert the user in some other appropriate way) and either remove or simply never display the letter. Optionally, the value can be validated when the user submits the dialog. At that time, if the value is invalid, the application should display an appropriate error message in a message dialog that contains a single OK button. Choosing OK to acknowledge the message should close the message dialog, but leave the original dialog open so that the user can change the invalid value. See section 6.10 for additional guidelines for validation of input.

6.5 Read-Only Pop-Up Text Fields[11]

Space limitations often restrict the amount of interface text that can be displayed. For example, long path names may not fit in a file dialog. Similarly, in Help text, it is usually impossible to include in-line definitions for every term. In such situations, applications can use the read-only pop-up text field illustrated in Figures 6.14 and 6.15.

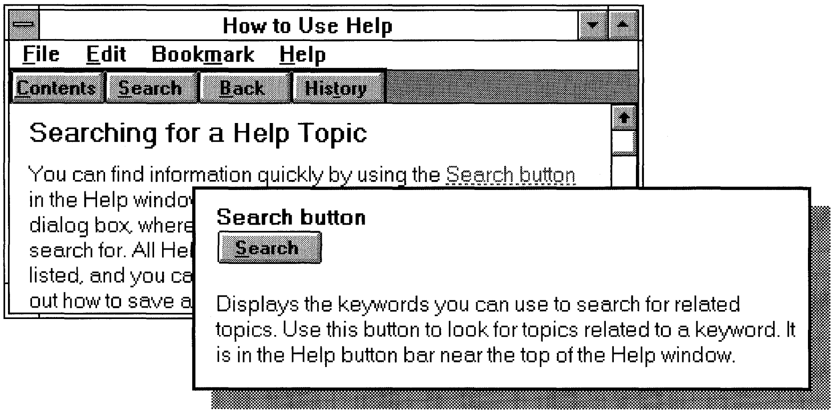


Searching for a Help Topic

You can find information quickly by using the Search button in the Help window. The Search button opens the Search dialog box, where you select a keyword that you want to search for. All Help topics associated with that keyword are listed, and you can select one to view. For example, to find out how to save a file, you could select "save" from the list of

Figure 6.14 Closed Read-Only Pop-Up Text Field

[11]The read-only pop-up text field is not a predefined Windows program control.



Searching for a Help Topic

You can find information quickly by using the Search button in the Help windo
dialog box, where
search for. All He
listed, and you ca
out how to save a

Search button

Displays the keywords you can use to search for related topics. Use this button to look for topics related to a keyword. It is in the Help button bar near the top of the Help window.

Figure 6.15 Open Read-Only Pop-Up Text Field

As shown in Figure 6.15, a dotted underline beneath the text contained in the text field indicates that the user can click on the text to obtain a pop-up with additional information. The pointer changes to a hand with an extended finger (as in Windows Help) when over the field, to provide an additional indication that more information is available. The pop-up opens when the user presses mouse button 1 over the text and remains open when the button is released. Clicking anywhere closes the pop-up.

From the keyboard, the user can navigate to the text field with TAB. If the field is labeled (as it should be, unless it appears in a continuous stream of text), the mnemonic in the field label can also be used to navigate to the field. To open and close the pop-up, the user can press the Select key (SPACEBAR) or any of the keys used for opening and closing drop-down controls (see section 6.3.1.3). When the pop-up opens, its top left corner appears at the same position as the top left corner of the original text. If the contents of the pop-up can change between invocations, the pop-up should be dynamically sized so that it is large enough to hold its current contents.

6.6 Sliders

Sliders are used to display and adjust values on continuous dimensions such as pitch, loudness, and brightness. A slider consists of a bar containing notches or measurement markings, plus an indicator perpendicular to the bar, as shown in Figure 6.16. The indicator shows the present value and can be dragged along the bar with the mouse to set a new value.

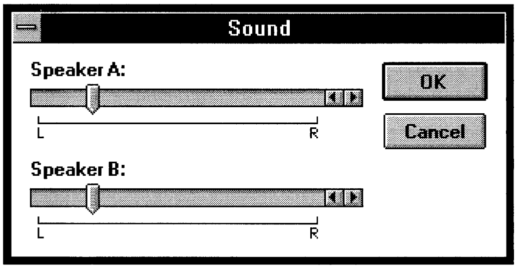


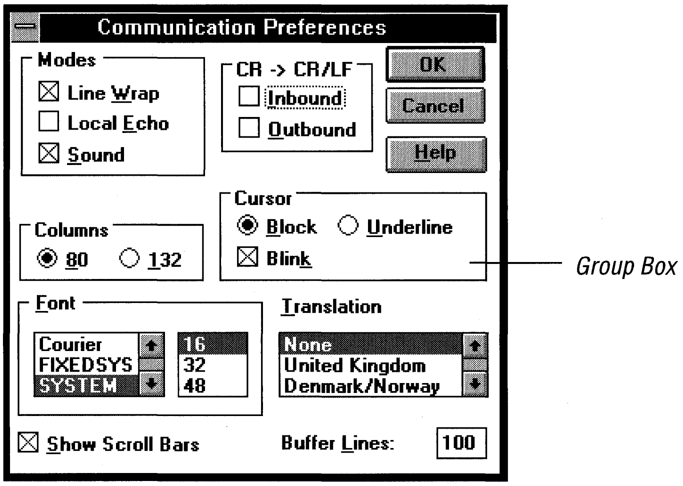
Figure 6.16 Slider

6.7 Static Text Fields

Static text fields are used to present read-only textual information (for example, the current directory location). These fields are static in the sense that the user cannot change the text in them; the application, however, can alter the text to reflect the current state of the application. Static text fields are often used to label controls that are not automatically labeled by the system (see section 6.9). When the user accesses a static text field using a keyboard mnemonic, the focus is passed to the next control in the TAB order. Accordingly, when a static text field is used as a label, it should immediately precede the labeled control in the TAB order.

6.8 Group Boxes

Group boxes, though technically considered controls, do not process any mouse or keyboard input; they may be used to provide visual grouping of related controls. Group boxes consist of a rectangular one-pixel frame with a label at the upper left, as shown in Figure 6.17.



Modes
[✓ Line Wrap]
[ ] Local Echo
[✓ Sound]

CR -> CR/LF
[ ] Inbound
[ ] Outbound

Columns
[● 80] ○ 132

Cursor
[● Block] ○ Underline
[✓ Blink]

Font
Courier / FIXEDSYS / [SYSTEM]
[16] / 32 / 48

Translation
[None] / United Kingdom / Denmark/Norway

[✓ Show Scroll Bars]

Buffer Lines: 100

Figure 6.17 Group Boxes

6.9 Control Labels

Buttons, check boxes, and group boxes are automatically supplied with labels by the system; other controls can be labeled with static text fields (see section 6.7). Labels identify the function of the control and provide direct keyboard access to the control. The following guidelines are suggested for control labels:

- Capitalize the first and last words of labels. In addition, capitalize the initial letters of all other words in labels, except for articles (for example, *a*, *an*, and *the*), coordinate conjunctions (for example, *and*, *or*, *nor*, and *for*), prepositions (for example, *by*, *through*, and *with*), and the *to* in infinitives.
- Provide a unique mnemonic access character for labels of controls to which the user needs direct keyboard access. If possible, use the first character of the label as the access character. In the following cases, use another letter from the label:

  a. Another letter offers a stronger mnemonic link (for example, the letter X in Exit).

  b. The label contains multiple words, one of which is more significant than the first word (for example, "Process" in Set Process).

  c. The first character has already been used as a mnemonic for another control.

  Use consonants in preference to vowels because consonants are usually more distinctive and more easily remembered. Do not assign mnemonics to the OK command button because it can be accessed through the ENTER key.
- Dim the labels of unavailable or inapplicable controls.

- Use a bold font so that dimmed labels are not illegible.
- Position control labels according to the rules in Table 6.1.

Table 6.1 Position of Control Labels

Control	Label Position
Command button	Inside button.
Check box or option button	To right of box or button.
Text box, spin box, list, combo box, slider, read-only pop-up text field	Above or to left of control, followed by a colon, and left-aligned with the section of the dialog in which it appears.
Group box	On top of (and replacing) part of top frame line, starting just after upper left corner.

6.10 Validation of Input

Applications can validate input (or other dialog settings) immediately, after navigating away from a field, or when the dialog is submitted. Generally, the first two techniques provide better feedback because the user remains in the context where the information is supplied. However, these may not be appropriate when data fields cannot be processed individually.

Valid input can also be controlled by the type of control used to receive the input. For example, option buttons, check boxes, and drop-down lists limit the type of input that can be selected. Other types of controls (for example, those with text boxes) generally provide more flexibility for user input.

Using Dialog Boxes

Chapter 7

Some application commands require additional information from the user before they can be completed. For example, if the user selects File Open, the application needs the name of the file and where it is stored. If the user does not or cannot supply the information when issuing the command, the application can request the information by displaying a special window called a dialog box. A dialog box contains controls that collect the user's information and choices. These controls show the attributes of the selected data when the dialog box first opens. Dialogs do not generally contain menu bars, window scroll bars, split bars, resizing buttons, or status bars.

This chapter discusses types of dialog boxes and the use of command buttons in dialog boxes. For information on navigation in dialogs, see Chapter 3, section 3.3. Dialog box controls are discussed in Chapter 6 and several standard dialogs are described in Chapter 8.

7.1 Types of Dialog Boxes

Dialog boxes can be classified according to various characteristics:

- They may be movable or fixed in position.
- They may have a single size or two alternate sizes.
- They may be modal (that is, may require the user to respond before continuing), semimodal, or modeless.
- They may present simple messages with limited response options, or more complex transactional choices accompanied by a variety of controls.

7.1.1 Movable vs. Fixed Dialogs

The appearance of a dialog box depends on whether it is movable. A movable dialog box has a title bar containing a Control menu (which includes only the Move and Close commands) and a title. A dialog box that is not movable has no title bar. In general, an application should use only movable dialog boxes; the user can reposition these to view obscured data and maintain a sense of context because the name of the command or application is displayed in the title bar of the dialog. The title of dialog boxes that present messages should simply reflect the application name (or the source of the error, if the message represents a system or network error; see section 7.1.4 for details). The title of dialog boxes that represent completions of menu commands should reflect the name of the command that led to the dialog, without an ellipsis. The menu name should not be included in the dialog title, unless the same command exists on two different menus, or unless the command name alone is uninformative without the menu name.

7.1.2 Unfolding Dialogs

Whether they are movable or not, all dialog boxes have a non-sizable frame. Applications may, however, have dialogs of two sizes: a small size containing the basic controls and a larger size that includes advanced options. The user can expand the dialog from the small size to the large size by pressing an unfold button (a command button with chevrons, for example, "Options >>") included in the small form of the dialog. The dialog box may expand to the right, downward, or in both directions. The expanded form of the dialog contains both basic and advanced options.

After the dialog is expanded, applications may optionally leave the button active with the chevrons facing the opposite direction (for example, "Options <<") to allow the user to return the dialog to its original size. If the application does not allow the dialog to be returned to its original size, the button label should be dimmed. The next time the dialog box is displayed, it appears in its original (unexpanded) form; however, applications may provide options for using the expanded form instead.

7.1.3 Modal vs. Modeless Dialogs

Dialog boxes can also be classified according to whether they require the user to respond before continuing to work in the current application and in other applications. The four types of dialog boxes in this classification are: application modal, system modal, application modeless, and application semimodal.

Table 7.1 describes the characteristics of these dialog box types.

Table 7.1 Modal, Modeless, and Semimodal Dialog Boxes

Type of Dialog	Characteristics	Uses
Application modal	User must respond to dialog before continuing work in current application, but can switch to and work in other applications without responding.	Obtain information required before current application can continue. Should be used sparingly; modeless dialogs should be used instead whenever possible because the user can leave them open (for easier access) while continuing to work in the application.
System modal	User must respond before doing anything else in any running applications.	Obtain information required before any applications can continue. Should be used only for severe system problems, for example, impending fatal system error or unrecoverable error with active storage device.
Application modeless	User can continue work in current application without responding to dialog; dialog remains on display.	Display information or offer options that don't require immediate attention; obtain information for commands that need not be completed before proceeding (for example, Find). Useful for frequently used commands; user can leave dialog open rather than reselecting command from menu. When a movable palette or toolbox is implemented as a modeless dialog, it should always stay in a layer above the window to which it applies. In MDI applications, it should stay in a layer above all document windows.
Application semimodal	User can perform a limited number of operations outside the dialog as a means of responding to the dialog.	Offer alternative ways of responding. For example, a semimodal dialog in a spreadsheet might allow the user to specify ranges by clicking and dragging outside the dialog in the worksheet, as well as by using controls within the dialog.

Although semimodal dialog boxes are not part of the recommended minimum requirements of interface style, special situations within some applications may make such dialogs appropriate. Semimodal dialogs are similar to modal dialogs, in that the user must complete the dialog before continuing to work on a document in other ways.[1] Semimodal dialogs, however, permit a limited set of actions outside the dialog as a means of completing the dialog when the focus is on certain controls within the semimodal dialog. In particular, semimodal dialogs are appropriate in situations where it is convenient to accept mouse input or keyboard shortcuts from outside a dialog as a means of setting controls or defining attributes within the dialog. The user may only perform actions that set controls in the dialog but is not confined to using only the dialog box controls. For example, if column width can be set in a dialog text box, a semimodal dialog could also permit the user to drag columns to a new location with the mouse, with the text box echoing the measurements as the user drags the column border.

[1] Visually, semimodal dialogs should use the same window style as modal dialogs.

7.1.4 Message Dialogs

Modal dialog boxes that are used to display error messages and other important information are called message dialogs. Because message dialogs are modal dialogs, they require a response by the user before work with the application can proceed. Message dialogs for critical messages, which inform the user of serious system-related problems, are system modal dialogs. All other types of messages appear in application modal message dialogs. Message text should follow the guidelines in Chapter 3, section 3.6.1.2. An application should post messages only when it is the active application; when inactive, it should flash or beep for attention, as described in Chapter 3, sections 3.6.1.1.4 and 3.6.2.

Every message dialog has a title bar. The title is particularly important in a multitasking environment because it identifies the source of the message. The title of message dialogs in applications should consist simply of the name of the application. The title of message dialogs associated with system or network processes should contain the name of the system or network product, plus the word "Message" (see Figure 7.3). The reason for this longer title is that users are often unaware of the system and network processes that are occurring behind the scenes. The longer title "*<System or Network Product Name>* Message" helps clarify the purpose of the message dialog, which may have appeared unexpectedly while the user was working in an application. Message dialog titles should never include the word "Error," which has a negative effect on users.

Each message dialog also includes a graphical symbol that indicates what kind of message is being presented. The three types of messages are:

- Information messages
- Warnings
- Critical messages

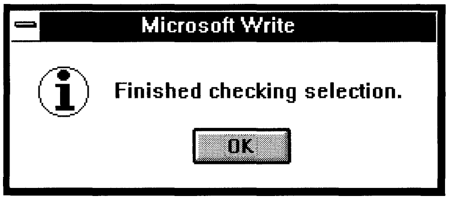
Table 7.2 describes each message type and shows the associated symbol.

Table 7.2 Message Types and Associated Symbols

Symbol	Message Type	Description
ⓘ	Information	Provides information about results of commands. Offers no user choices; user acknowledges message by clicking OK button.
(!)	Warning	Alerts user to an error condition or situation that requires user decision and input before proceeding, such as an impending action with potentially destructive, irreversible consequences. The message can be a question (for example, "Save changes to REPORT.WRI?").*
STOP	Critical	Informs user of a serious system-related or application-related problem that must be corrected before work can continue with the application.

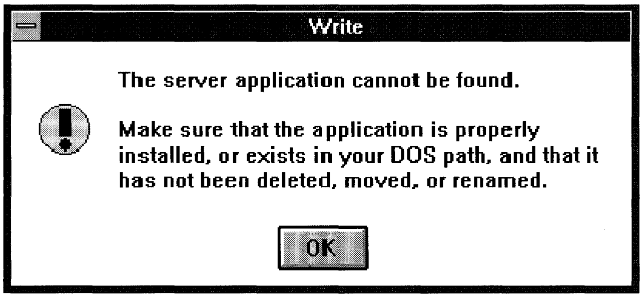
* Some applications may find the question mark symbol more appropriate if the message is a question. However, note that the question mark is also used as the help symbol and may therefore confuse users.

Figures 7.1, 7.2, and 7.3 present examples of complete message dialogs.


Microsoft Write

Finished checking selection.

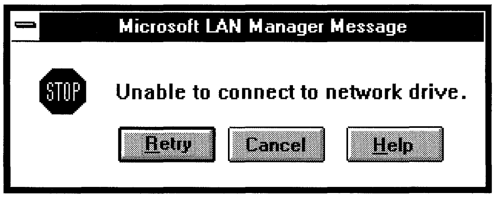
Figure 7.1 Information Message


Write

The server application cannot be found.

Make sure that the application is properly installed, or exists in your DOS path, and that it has not been deleted, moved, or renamed.

Figure 7.2 Warning Message



Unable to connect to network drive.

Figure 7.3 Critical Message

Message dialogs should contain only command button controls. If the application needs additional information, such as a file name, then a regular dialog box should be used instead of a message dialog. The standard command button combinations for message dialogs are discussed in section 7.3.6.

7.2 Dialog Placement

Dialog boxes are usually centered vertically and horizontally within the application window. If an application finds a more appropriate location for the dialog box, it may choose to place it elsewhere. On large screens, this rule could cause small dialogs to be positioned quite far from the menu bar at the top of the window, which is where users must focus their attention in order to obtain most dialogs.

7.3 Using Command Buttons in Dialogs

A command button is a rectangular shape containing a label that indicates what the button does (for example, Assign or Cancel), as discussed in Chapter 6, section 6.1.1. After the user presses and releases (that is, clicks) the command button, the action begins. A dialog box may contain several types of command buttons:

- Command buttons that initiate actions.
- GoTo[2] buttons, which close the current dialog box and open a related one.
- GoSub buttons, which open a related dialog box on top of the current one without closing the current dialog box.
- Unfold buttons, which expand the dialog to include additional options.

[2] GoTo, GoSub, and Unfold are used descriptively; they are not the actual button labels. Applications should choose appropriate labels for these buttons.

7.3.1 Recommended Buttons

Every dialog box contains at least one button that closes the dialog. Message dialogs that require only an acknowledgment from the user (rather than a choice) contain a single button labeled OK. All other dialog boxes contain at least two buttons: One closes the dialog and initiates an action; the other closes the dialog without initiating any action. The button that initiates the action is usually labeled either OK or <*Action-Name*>. The button that closes the dialog box without initiating an action is usually labeled Cancel. Some dialogs (called multiple-action dialogs) include additional buttons that allow the user to initiate actions without closing the dialog. In these dialogs, if the actions performed by the additional buttons irreversibly change the user's data, the label of the Cancel button should change to Close as soon as the first such action is carried out. Whether the button is labeled Cancel or Close, the keyboard user can press this button by pressing ESC. (The OK and Cancel buttons do not have mnemonics.) If the label is Close, C is underlined and serves as a mnemonic access character. This additional means of access helps users who know that ESC presses the Cancel button, but who may not realize that ESC also presses the Close button after Cancel changes to Close.

7.3.2 Default Buttons

One command button in a dialog box may be designated the default button, which will be pushed when the user presses ENTER. The default button should be distinguished from the others with a heavy border. In multiple-action dialogs, it may be appropriate for different buttons to be the default button at different times, depending on the current state of the data and the user's interaction with the dialog. In general, the button that indicates the most likely next action at any given time should be designated the default. However, a button that initiates an action with far-reaching and potentially destructive consequences should never be the default. For example, in a text search and substitution dialog, the Replace All button should never be the default.

If the user navigates to a command button that is not usually the default, the new button is temporarily designated the default (that is, the heavy border moves from the original default to the new button, and ENTER now pushes the new button rather than the old one). If the focus moves away from the temporary default button to a control that is not a command button, the button that was originally the default becomes the default again. For additional information on keyboard navigation to controls, see Chapter 3, section 3.3.2.

7.3.3 Dynamic Button Labels

The labels of individual buttons can change in two ways to reflect the possibilities currently available to the user:

- If the action represented by the button is not currently available, the label should be dimmed.
- If the nature of the action represented by the button changes depending on circumstances, the button label can be modified to reflect that change. For example, as mentioned previously, the label of the Cancel button should change to Close after any actions that cannot be cancelled have been carried out. When the dialog closes, however, the button label should be reset to Cancel for the next time the dialog appears.

7.3.4 Navigation to Related Dialog Boxes

As an application becomes more complex and provides additional features, the need to display options in a consistent and efficient manner becomes increasingly important. For example, if the application allows the user to set character properties from several dialogs, there is no need to complicate each dialog by repeating the controls for setting those properties. Instead, each dialog should provide a means to navigate to a single, uniform character properties dialog.

To provide for consistent presentation of controls and to keep individual dialogs simple, applications may use GoTo and GoSub command buttons to let the user navigate to related dialog boxes. A GoTo button typically closes the original dialog before opening the new dialog, whereas a GoSub button leaves the original dialog open and returns to it when the new dialog is closed. When a GoTo or GoSub button leads to a dialog that is also associated with a menu item, the dialog should have the same title as when it is invoked by selecting the menu item; typically this title will be the name of the menu item. The labels for GoTo and GoSub buttons should include a trailing ellipsis ("..."), which indicates that the commands represented by the buttons require additional information (the information collected in the new dialog) before they can be completed.[3]

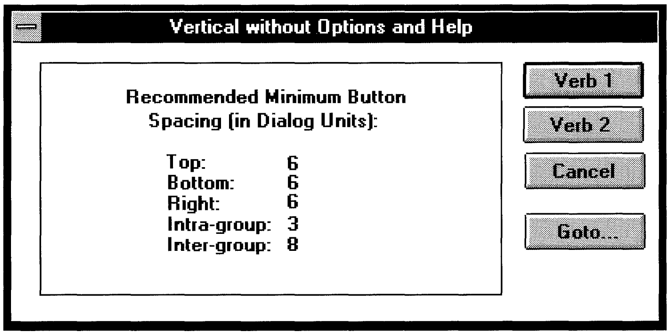
If the user invokes a GoSub dialog and then returns to the original dialog, the Cancel button in the original dialog should cancel all changes that were made in both dialogs if the transaction has been committed and cannot be undone at that point.

[3] The ellipsis implies that the command leads to a dialog, but the reverse is not always true; commands that lead to message dialogs should not be followed by an ellipsis.

7.3.5 Arrangement of Buttons

Before users can decide which buttons to press in a dialog, they must first take in the information presented by the dialog. This information is scanned by eye movements that typically proceed as in normal reading; that is, from left to right and from top to bottom. Accordingly, the most appropriate places for buttons in dialog boxes are at the right or at the bottom, where the buttons will be seen after the user has already scanned the relevant information. Whenever possible, buttons should be arranged in one of two ways:

A. Stacked along the right border of the dialog, starting in the top right corner (see Figures 7.4, 7.5, and 7.6). In this case, buttons should generally be the same width—as wide as necessary to accommodate the longest button text. Group the command buttons according to whether they initiate an action (that is, Add, Remove, Find) or not (that is, GoSub, GoTo, or unfold buttons). If there is an OK button, OK and Cancel should be grouped together, separated from the other action verbs (see Figure 7.5). If there is no OK button, Cancel should be grouped with other action buttons (see Figure 7.6). Leave at least 3 dialog units (DUs) between the bottom of one button and the top of the next within the same group.[4] Between groups, do not use horizontal lines, which simply clutter the dialog; instead, insert at least an extra 5 DUs of white space, for a minimum total of 8 DUs between groups. Leave at least 6 DUs between the edge of the dialog and the edges of the buttons (that is, the top of the first button, the bottom of the last button, and the right edges of all the buttons).

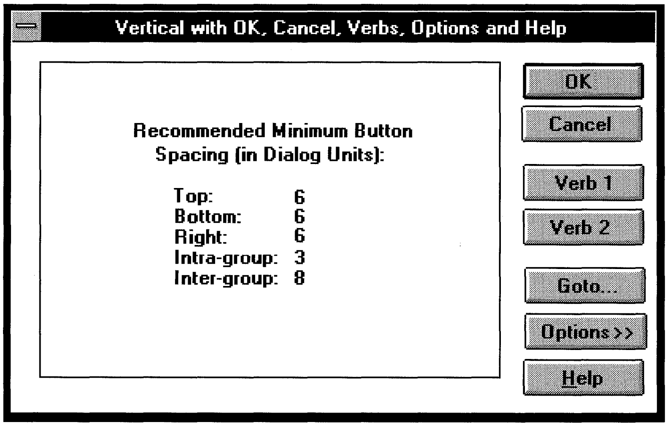


Recommended Minimum Button Spacing (in Dialog Units):

Top:	6
Bottom:	6
Right:	6
Intra-group:	3
Inter-group:	8

Figure 7.4 Vertical Button Layout Without Options and Help

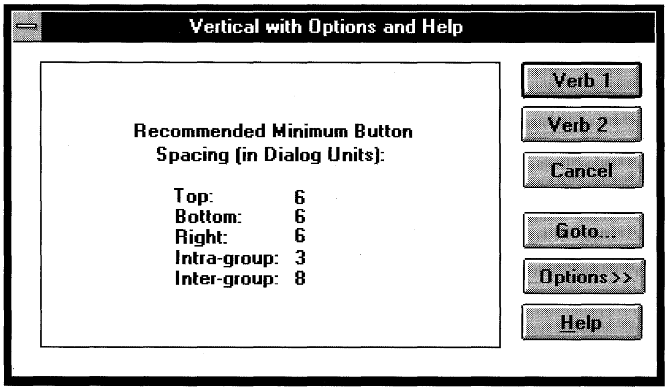
[4] The measurements mentioned in this section are based on the use of 8-point MS Sans Serif Bold in the Windows Dialog Editor.



Recommended Minimum Button Spacing (in Dialog Units):

Top:	6
Bottom:	6
Right:	6
Intra-group:	3
Inter-group:	8

Figure 7.5 Vertical Button Layout with OK, Cancel, Verbs, Options, and Help

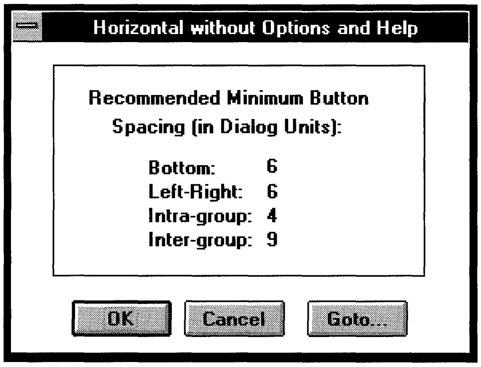


Recommended Minimum Button Spacing (in Dialog Units):

Top:	6
Bottom:	6
Right:	6
Intra-group:	3
Inter-group:	8

Figure 7.6 Vertical Button Layout with Options and Help

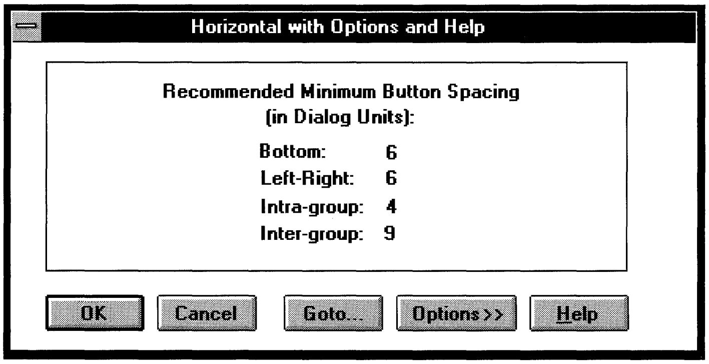
B. Line buttons up across the bottom of the dialog (see Figures 7.7, 7.8, and 7.9). Group the command buttons according to whether they initiate an action (that is, Add, Remove, Find) or not (that is, GoSub, GoTo, or unfold buttons). If there is an OK button, OK and Cancel should be grouped together, separated from the other action verbs (see Figure 7.8). If there is no OK button, Cancel should be grouped with other action buttons. Space the buttons evenly within each group, leaving at least 4 DUs between the right edge of one button and the left edge of the next. Between groups, do not use horizontal lines, which simply clutter the dialog; instead, insert at least an extra 5 DUs of white space, for a minimum total of 9 DUs between groups. Leave at least 6 DUs between the edge of the dialog and the edges of the buttons (that is, the left edge of the first button, the right edge of the last button, and the bottom of all the buttons). Normally the buttons should all be the same width, but individual buttons may be made wider to accommodate exceptionally long text (see Figure 7.9). It is not necessary to align buttons with other dialog controls, because the buttons should form their own visual group rather than being grouped with the other controls.



Recommended Minimum Button Spacing (in Dialog Units):

Bottom:	6
Left-Right:	6
Intra-group:	4
Inter-group:	9

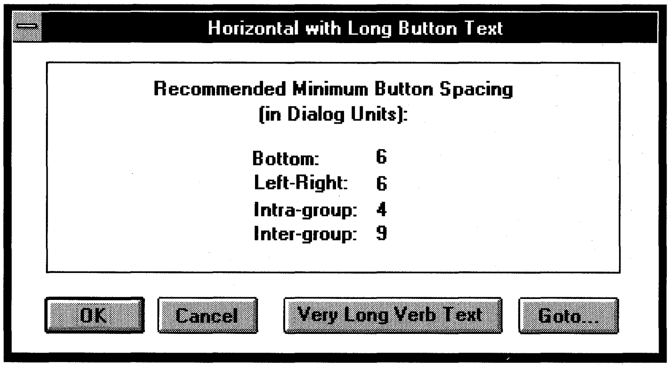
Figure 7.7 Horizontal Button Layout Without Options and Help



Recommended Minimum Button Spacing (in Dialog Units):

Bottom:	6
Left-Right:	6
Intra-group:	4
Inter-group:	9

Figure 7.8 Horizontal Button Layout with Options and Help



Recommended Minimum Button Spacing
(in Dialog Units):

Bottom:	6
Left-Right:	6
Intra-group:	4
Inter-group:	9

Figure 7.9 Horizontal Button Layout with Long Button Text

Additional guidelines:

- The most important button—typically, the default command—should be placed at the top (if arrangement A is used) or at the left (if arrangement B is used), followed by other command buttons that initiate actions, followed by remaining command buttons (if present) in this order: GoTo or GoSub button, unfold button, and Help button (see Figures 7.6 and 7.8).
- The Help button is placed after all other buttons so that it will be located near the bottom right of the dialog, where it will be visible to users after they scan the dialog.
- If an OK button[5] is present, place the Cancel button immediately after it; otherwise, place the Cancel button after all other command buttons, but before GoTo or GoSub, unfold, and Help buttons (see Figures 7.6 and 7.8). If OK is not the default button, it should still be placed first. (This will keep its location consistent with the large number of dialogs where it is first because it is the default.)
- If there is not enough room to fit all buttons in a single location, use arrangement A for the most important command buttons and arrangement B for all other command buttons.

[5] If there is a single, obvious default action in the dialog box, use OK rather than a verb. For example, to access the File Open dialog, the user selects Open from the File menu, and Open appears in the title bar of the dialog. In this case, it is clear that the default action is to open something (a file or a directory), so there is no need to repeat "Open" as the default button name; "OK" is sufficient. In contrast, if the nature of the interaction in the dialog makes the intended action ambiguous (for example, if there are several possible actions, none of which is clearly the default), consider replacing "OK" with a verb.

- Except in message dialogs, arrangement A (vertical stacking at the right) is more common than arrangement B (horizontal alignment at the bottom). Accordingly, if either method would work equally well in terms of dialog layout, arrangement A should be given priority because it will be more familiar to users. Message dialogs are an exception to this rule; buttons in message dialogs should be placed at the bottom (arrangement B), where they will not interfere with the left-to-right flow of message text.
- The recommended minimum size for OK and Cancel buttons is 40 x 14 DUs. The recommended minimum space between the left or right edge of a button and the nearest edge of the text within the button is approximately the width of a lowercase "n".
- Other arrangements may be used if there is a compelling reason, such as a natural mapping relationship. For example, it would make sense to place buttons labeled North, South, East, and West in a compass-like layout.

7.3.6 Command Buttons in Message Dialogs

Command buttons are the only controls used in message dialogs; they represent the responses or choices offered to the user. The safest or most typical response should be designated the default command button.

If a message requires no user choice and only needs acknowledgment, the message dialog should contain only an OK button and (optionally) a Help button. (If this type of message appears as the result of a dialog submission, acknowledging the message should dismiss the message but leave the dialog open, so that the user can correct the error.) If the message requires the user to make a choice, the dialog should contain a button for each option. The clearest way to present the choices is to ask the user a question and provide a button for each response. When possible, questions should be phrased to permit Yes or No answers, which can be represented by Yes and No command buttons. The use of OK and Cancel buttons in place of Yes and No is permissible. However, Yes and No are preferable, because OK and Cancel can be ambiguous for questions that imply cancellation, such as "Interrupt file transfer?" or "Delete reservation?" If the message cannot be phrased unambiguously for Yes or No responses, the command buttons can instead be labeled with the names of the relevant actions (for example, "Save" and "Delete").

Some message dialogs offer the user the following three choices:

- Performing a preliminary action (for example, saving data) before carrying out the process that led to the message dialog (for example, closing a document).
- Not performing the preliminary action before carrying out the process.
- Canceling the process altogether.

The appropriate buttons for this type of message dialog are Yes, No, and Cancel. Yes means "Perform the preliminary action and then carry out the process"; No means "Don't perform the preliminary action, but do carry out the process"; and Cancel means "Don't perform the preliminary action, and don't carry out the process either."

Help buttons in message dialogs are optional but highly recommended, especially for warning and critical message dialogs.[6] They allow the user to obtain further information or suggestions about the problem described by the message.

7.4 Fonts in Dialogs

As mentioned in Chapter 6, section 6.9, a bold font is suggested for control labels in dialogs, so that the labels remain legible when dimmed to reflect unavailability. The font used for control labels in the dialog box is also typically used for other text, but different fonts may be used if there is a good reason to do so. For example, if a dialog displays a sample of text from another document, the text could be displayed in its original fonts. If reproducing the original fonts is too difficult, the text could be displayed in a single font that is different from the font used in the rest of the dialog. The use of a different font emphasizes that the displayed text represents data from another document, rather than standard elements within the dialog.

7.5 Samples in Dialogs

Dialog boxes are frequently used to change visual properties or attributes. If the changes selected in the dialog are not reflected immediately in the document, it is helpful to provide a sample inside the dialog that shows the effect of the changes. For example, the standard character properties dialog includes a text sample that reflects changes to the font, type style, point size, and color (see Chapter 8, Figure 8.10).

[6] The user can also press the F1 key to obtain help.

Common Dialog Boxes

Chapter 8

This chapter describes common dialog boxes for the following operations:

- Opening and saving files (sections 8.1.1 and 8.1.2)
- Creating new documents (section 8.1.3)
- Printing files (section 8.2)
- Searching and replacing text strings (section 8.3)
- Setting character properties (section 8.4)
- Setting page margins (section 8.5)
- Displaying information about an application (section 8.6)

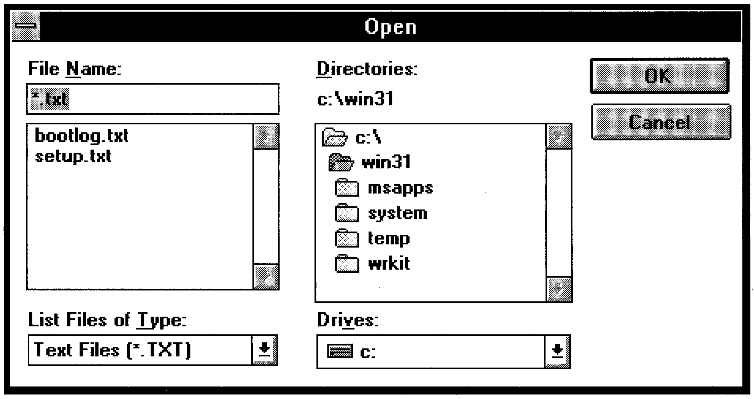
The dialog boxes discussed in this chapter provide a starting point for applications; they can be customized depending on an application's specific needs.

8.1 File Operations

The next three sections describe the new standard for File Open, File Save As, and File New dialogs. Applications that use other dialogs that involve file browsing (for example, Insert File) should model them after the File Open dialog.

8.1.1 File Open Dialog

Figure 8.1 shows the File Open dialog.



File Name:
*.txt
bootlog.txt
setup.txt

Directories:
c:\win31
c:\
win31
msapps
system
temp
wrkit

List Files of Type:
Text Files (*.TXT)

Drives:
c:

Figure 8.1 File Open Dialog

8.1.1.1 Directories Control

The Directories control is a list box that displays:

- First, the parent directories, preceded by 🗁
- Next, the current directory, preceded by 🗁
- Finally, the child directories of the current directory, preceded by 🗀

The directory icons and names are indented according to their depth in the directory tree. To move to a directory, the user double-clicks the directory name or icon, or selects it and then chooses OK (or presses ENTER). If the indentation causes directory names to be clipped at the right, a horizontal scroll bar appears at the bottom of the list.

If the current directory contains so many child directories that there is not enough room in the list box to display all the parents, the immediate parent is displayed at the top of the list, followed by the current directory and its child directories. Displaying the immediate parent helps the user maintain a sense of orientation in the directory tree, and allows the user to move up at least one level in the directory tree without having to scroll the list.

When the user switches to a new drive, the directory list box should always show the root directory at the top, even in the rare case that the active path directory is so far down in the directory tree that it is not visible. Displaying the root directory at the top helps users orient themselves to the directory structure of the newly selected drive.

Note that the Directories list box contains no ".." entry because the parent directories are shown. Note also that the root directory is indicated by a drive letter followed by a backslash (for example, "c:\"), not by the backslash alone.

After the user switches to a new directory, the keyboard focus may optionally move to the File Name text box so that the user can type a new file name immediately. This automatic shift of focus away from the Directories control makes repeated directory navigation slightly more difficult for keyboard users. However, most users will change directories only once (if at all) during each dialog invocation and will want to type a file name immediately after changing directories.

8.1.1.1.1 Directory Tracking Text The non-editable text item above the directory list box should show not only the current directory name (for example, "xyz") but rather a longer string (for example, "c:\..\xyz") that indicates the current position in the directory tree. To truncate this string when pathnames are long, use the following rule: If *N* is the maximum number of wide characters (for example, W is a wide character) that can fit in the available space after an initial "c:\..", the string should show as many entire nodes as will fit into *N* characters. It should not show parts of nodes unless the number of entire nodes that will fit is less than one; that is, unless even the last node will not fit. In that case, the beginning of the last node should be truncated.

Table 8.1 lists truncation rules. Note that increasing *N* from 10 to 11 changes the tracking text in all cases; increasing *N* from 11 to 12 changes the first two cases; and increasing *N* from 12 to 13 only affects the "GammaAndDelta" case.

Table 8.1 Truncation of Directory Tracking Text

N	Path	Tracking Text
10	c:\Alpha\Beta\GammaAndDelta	c:\..maAndDelta
10	c:\Alpha\Beta\Gamma\Delta	c:\..\Delta
10	c:\Alpha\Beta\Gamm\Delta	c:\..Gamm\Delta
11	c:\Alpha\Beta\GammaAndDelta	c:\..mmaAndDelta
11	c:\Alpha\Beta\Gamma\Delta	c:\..Gamma\Delta
11	c:\Alpha\Beta\Gamm\Delta	c:\..\Gamm\Delta
12	c:\Alpha\Beta\GammaAndDelta	c:\..ammaAndDelta
12	c:\Alpha\Beta\Gamma\Delta	c:\..\Gamma\Delta
12	c:\Alpha\Beta\Gamm\Delta	c:\..\Gamm\Delta
13	c:\Alpha\Beta\GammaAndDelta	c:\..GammaAndDelta
13	c:\Alpha\Beta\Gamma\Delta	c:\..\Gamma\Delta
13	c:\Alpha\Beta\Gamm\Delta	c:\..\Gamm\Delta

Showing a small number of whole nodes rather than parts of a larger number of nodes is recommended for two reasons:

- First, whole nodes (for example, "Report" as opposed to "ort") are recognized and understood more easily.
- Second, if parts of two nodes are displayed, the ellipsis between them is somewhat ambiguous. Unless the user knows the truncation rules, it is not immediately apparent that the last ellipsis represents truncation of only one node. For example, "c:\...Beta\...amma" could represent "c:\a\b\Beta\Gamma", "c:\a\b\Beta\Foo\Gamma", "c:\a\b\Beta\Foo\Bar\Gamma", and so on.

The recommended minimum length for *N* is 12. Because *N* does not include the initial drive letter, colon, backslash, and ellipsis, 12 is enough for an 8-character directory name, followed by a period and a 3-character extension. If possible, *N* should be larger than 12 to allow more of the path to be displayed.[1]

8.1.1.2 Drives Control

The drives are displayed in a drop-down list box labeled Drives. This list displays the drive letter followed by the volume name (or server+share names), not the drive letter alone. The suggested format for volume names is "c: [*volume*]" (with two spaces after the colon, and the *volume* name in square brackets). For server+share names, the format is similar, except that the brackets are omitted to save space (for example, "f: \\server\share"). Each drive name should be preceded by an appropriate drive icon (floppy, hard disk, or network), as in the File Manager. If the drives list is open and the selection is changed with arrow keys, the files list is not updated until the drives list is collapsed, because on-the-fly updates would take too long. The drives list can be collapsed with ALT+DOWN ARROW, ALT+UP ARROW, TAB (which also navigates to the next control), or ENTER (which does not close the dialog so the user can see the effect of changing drives).[2]

8.1.1.3 File Name Control

This control consists of a text box labeled File Name and a list box immediately below it. The list box shows the existing files in the current directory. It should be tall enough to contain at least eight items. The list box should track typing in the text box, and selecting from the list should replace the contents of the text box.

[1] *N* should be easily modifiable by product localizers so that it can be changed for different languages on systems supporting long file names.

[2] As explained in Chapter 6, section 6.3.1.3, drop-down lists no longer allow scrolling and autoselection in their closed state. This prevents delays associated with processing or updating information.

8.1.1.3.1 Entering a String in the File Name Text Box In the text box, the user can type a file name, a filter, a drive, a directory, a complete path in the form "drive:directory\..\filename", or a universal naming convention (UNC) pathname.

When the user types a string in the text box and presses ENTER, the following algorithm is used to process the string:

- If the string is a filter, filter the list accordingly.
- If the string is not a filter, try to open the storage location (drive, directory, or file) represented by the string.
  - If successful, update the controls (if the string represents a drive or directory) or close the dialog and display the file (if the string represents a file). The updating of controls proceeds as if the new drive or directory had been chosen in the drive or directory controls.
  - If the open fails, divide the string into two parts (internally, not in the text box): the part before the last backslash (\), which represents a drive or a path, and the part after the last backslash. If the string contains no backslashes but does contain a colon (:), divide the string at the colon, and leave the colon attached to the first part of the string. (For example, "c:\foo" is divided into "c:" and "foo"; while "c:\foo\bar.txt" is divided into "c:\foo" and "bar.txt"; and "c:foo" would be divided into "c:" and "foo".) Try to change to the drive or directory represented by the first part of the string. In other words, eliminate the part of the original string that might represent a file name, and try to change to the drive or directory where the user thought the file was located.

8.1.1.3.2 Effect of Navigation on Contents of Text Box Suppose that the user types a string in the text box, but then decides to navigate with the drive or directory controls (without first pressing ENTER to confirm the string). Any initial part of the string that could represent a drive or path is deleted, leaving only the part that could be a file name. In practice, this means that characters from the text box are deleted up to and including the last colon or backslash. The current filter is always shown in the type control, so there is no need to reproduce it in the text box and destroy a file name typed by the user.

8.1.1.4 Type Control

This control is a drop-down list labeled List Files of Type displayed below the list of files. Selecting a type modifies the contents of the files list box so that only files of the selected type are displayed. As an optional extension, applications can precede each type description in the list with a distinctive icon. The same icons can be used in the files list to distinguish between files of different types.

The purpose of the type list is to allow users to view all files that match a set of criteria, such as parent application (for example, Write), document category (for example, spreadsheet or template), or file format (for example, TIFF). (The set of criteria may be empty, in which case all files match it.) The type list adds the filter name in parentheses after the class description, to acknowledge that the resulting file list is based on name matches only, not on the more abstract class description. For example, the type list may contain items such as TIFF Files (*.TIF), Write Files (*.WRI), and so on. An application can include any filter that is appropriate.

There are two additional optional extensions:

- User-supplied types can be added to the type list. If the user types a filter in the text box that is not already included in the type list, the new filter becomes the first item in the type list, but is not given a label. If the user types in another new filter, this filter replaces the previous one; thus, the type list never includes more than one user-defined filter. The user-defined filter is preserved in the list in subsequent invocations of the dialog (but not in subsequent invocations of the application); but the dialog always opens with the type set to the default filter for the application.
- The type can optionally be preserved as the default during the current session. For example, if the user changes the extension to *.TXT, this remains as the filter until the user changes it or reruns the application. This allows the user to browse for files of a particular type without having to reset the type filter between invocations of the File Open command.

8.1.1.5 Layout

Most layout guidelines are shown in Figure 8.1. The following points deserve explicit mention:

- Align control labels with the left edges of the controls, not with the text inside the controls.
- Align the left edge of the List Files of Type control with the left edge of the File Name text box and list.
- Align the left edge of the Drives control with the left edge of the Directories list.
- The layout of the command buttons follows the recommendations in Chapter 7, section 7.3.5.
- Add application-specific controls either at the bottom of the dialog or under the Cancel command button.

8.1.2 File Save As Dialog

The File Save As dialog (see Figure 8.2) is similar in appearance to the File Open dialog, except for the following differences:

- The dialog contains the File Name text box and the list box underneath, but the list box items are dimmed and nonselectable, although still scrollable.
- The type control is labeled Save File as Type instead of List Files of Type. Selecting a type specifies the format of the file to be saved. It also filters the list of files, but does not affect the files directly. The type control contains only format descriptions, such as Normal, Text Only, Windows Write, and so on. Changing the type does not affect the contents of the File Name text box. Applications should provide appropriate extensions for each format and supply the extension if the user does not specify one in the File Name box. The format indicated by the type control overrides the extension specified by the user in the File Name box.

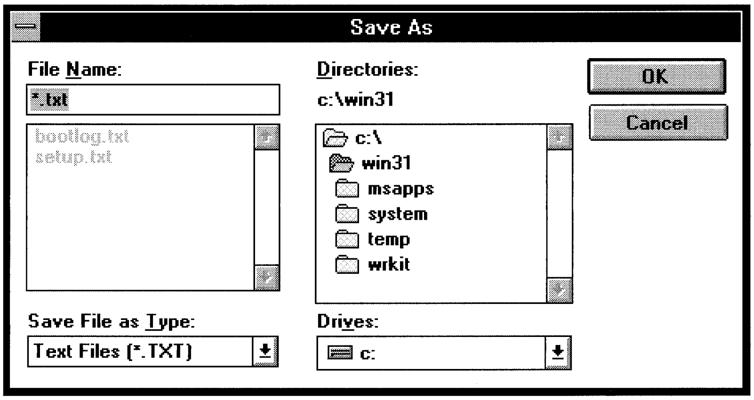


Figure 8.2 File Save As Dialog

8.1.3 File New Dialog

Applications that allow the user to create more than one type of document may provide a File New dialog similar to that shown in Figure 8.3.

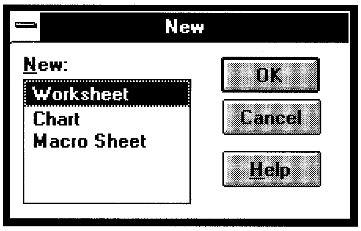


Figure 8.3 File New Dialog

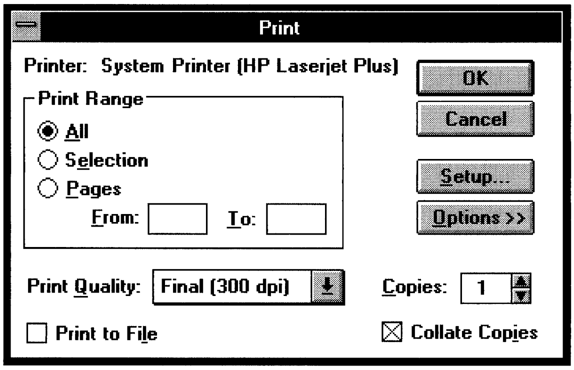
The New list contains the predefined document types for the application, as well as any document types that the user has created. When the dialog opens, the most common document type should be selected as the default. When the user presses the OK button, the dialog is closed, and a document based on the selected type is created and displayed in a window. The File New dialog should not slow down the process of document creation by requiring a user-supplied file name, because the user may create a temporary document without intending to save it as a file.

8.2 Printing

As discussed in Chapter 5, section 5.4.2.1, the Print and Print Setup commands are common items in the File menu. These commands lead to the common dialogs described in sections 8.2.1 and 8.2.2. The Print dialog allows the user to set properties for a particular print job (for example, page range and number of copies). The Print Setup dialog allows the user to set printing properties that will be stored with the current document (for example, paper type and orientation). Both dialogs are modal.

8.2.1 Print Dialog

The recommended format for the Print dialog is shown in Figure 8.4. The printing properties set in the Print dialog last only for the duration of a particular print job; they are not stored with the document.



Printer: System Printer (HP Laserjet Plus)

Print Range
● All
○ Selection
○ Pages
From: To:

Print Quality: Final (300 dpi)

Copies: 1

☐ Print to File

☒ Collate Copies

Figure 8.4 Print Dialog

Most features of the Print dialog are self-explanatory, but a few deserve comment:

- In the Print Range group box, the Pages option button allows the user to print specific pages (or units) by typing the appropriate numbers in the From and To text boxes. You can enter ranges of pages as well; and leaving a box blank results in the printing of all remaining pages (or all preceding pages). Examples are shown in Table 8.2:

Table 8.2 Printing Ranges

From:	To:	Print Result:*
1	1	Page 1
3	-	Page 3 to end of document
2	3	Pages 2 and 3
-	5	Beginning of document to page 5
3	1	Pages 3 to 1 in reverse order

* In pages or in other appropriate document units.

- The Print Quality drop-down list box allows the user to choose from the list of printing resolutions provided by the printer device driver.
- If the Print to File check box is checked, a new dialog appears after the user presses OK and asks for the name of the file to which the print output should be sent. A file created in this way will print properly only on the printer for which it was generated; it is a device-dependent file, not a metafile.

- The Collate Copies check box is turned on by default. When printing multiple copies of multipage documents on page-oriented printers, users can turn off this check box to speed up printing.[3] On printers that do not support the printing of uncollated copies, the Collate Copies check box should always be checked, and the check box and its label should both be dimmed.
- The Setup button allows the user to reach the Print Setup dialog without going back to the File menu.
- The Options button unfolds the dialog to include application-specific options, such as controls for reversing print order, printing hidden text, and so on, as described in Chapter 7, section 7.1.2. This button may be omitted if the application includes no printing options. Applications that include only one or two printing options may also omit this button and simply add the options at the bottom of the dialog.

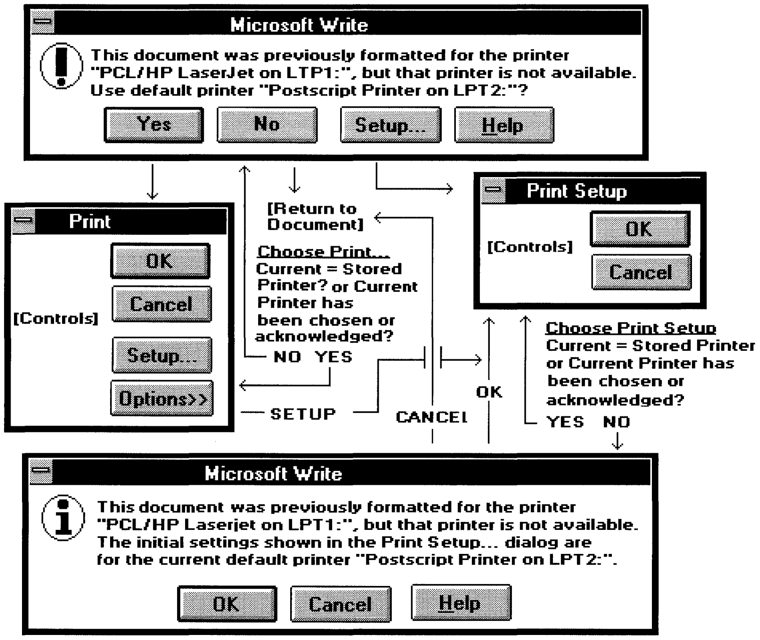
When the user chooses Print from the File menu, the following two conditions are tested before the Print dialog opens: Is the printer that was stored with the document currently available? If not, has the current printer already been explicitly chosen or acknowledged by the user? If neither of these conditions is met, a warning message (see Figure 8.5) is displayed with the text:

```
This document was previously formatted for the printer "<stored
printer name>", but that printer is not available. Use system printer
"<default printer name>"?
```

The message dialog contains three buttons: Yes, No, and Setup. Pressing any of these buttons closes the message dialog. Yes opens the Print dialog, No returns to the document, and Setup opens the Print Setup dialog so that the user can choose a different printer. The same tests are used if the user chooses Print Setup from the File menu, but the resulting message dialog contains no Setup button, and the message text is:

```
This document was previously formatted for the printer "<stored
printer name>", but that printer is not available. The initial
settings shown in the Print Setup... dialog are for the current
system printer "<default printer name>".
```

[3] On page-oriented printers, printing all the copies of the first page before going on to the second page is faster than printing the entire document once before going on to the second copy of the document.


This document was previously formatted for the printer "PCL/HP LaserJet on LTP1:", but that printer is not available. Use default printer "Postscript Printer on LPT2:"?

This document was previously formatted for the printer "PCL/HP Laserjet on LPT1:", but that printer is not available. The initial settings shown in the Print Setup... dialog are for the current default printer "Postscript Printer on LPT2:".

Figure 8.5 Message Dialogs for Printing

A message is also provided if all the following conditions are met when the user chooses Print or Print Setup:

- The document was previously stored with the default printer as the printer.
- The default printer was subsequently changed from the Control Panel.
- The new default printer does not support one of the settings (for example, landscape) that was previously stored with the document.

This "default printer changed" message does not change to reflect the condition; there is just one general-purpose message, with slight variations for Print versus Print Setup. For the Print dialog, the message is:

```
This document was previously formatted for a different default
printer, "<stored default printer name>". Use current default printer
"<current default printer name>"?
```

This message appears in a message dialog that contains Yes, No, and Setup buttons, like the "Non-default printer not available" message described previously (see also Figure 8.5). If the user presses Yes, the default settings of the new default printer are used wherever the ones stored with the document aren't available.

For the Print Setup dialog, the "Default printer changed" message is:

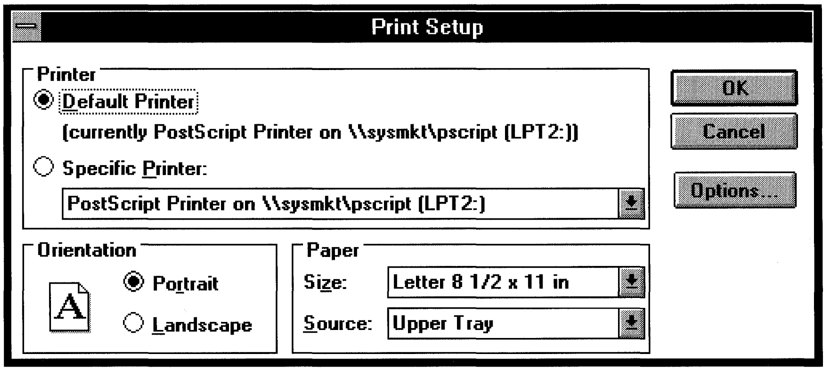
```
This document was previously formatted for a different default
printer, "<stored default printer name>". The initial settings shown
in the Print Setup... dialog are for the current system printer
"<current default printer name>".
```

This message is displayed in a message dialog that contains OK and Cancel buttons, like the Print Setup version of the "Nondefault printer not available" message described previously (see also Figure 8.5).

The user is not warned if the current printer matches the stored printer, but the current orientation or paper type do not match the stored settings. This situation is rare but could occur if a driver-specific dialog was used to modify the list of available paper types for a particular printer. In this case, the application should simply provide the best possible match to the stored values.

8.2.2 Print Setup Dialog

The recommended format for the Print Setup dialog is shown in Figure 8.6.[4] The printing properties set in this dialog are stored with the current document; they do not affect systemwide defaults, which can be changed only from the system Printer Setup dialog. Applications may expand the Print Setup dialog downward to include controls for document-specific properties such as margin settings.


Print Setup

Printer
● Default Printer
(currently PostScript Printer on \\sysmkt\pscript (LPT2:))
○ Specific Printer:
PostScript Printer on \\sysmkt\pscript (LPT2:)

Orientation
● Portrait
○ Landscape

Paper
Size: Letter 8 1/2 x 11 in
Source: Upper Tray

Figure 8.6 Print Setup Dialog

When the user chooses Print Setup from the File menu, the application should test the same conditions that are tested when Print is chosen (see Figure 8.5). The Print Setup dialog operates as follows:

[4] Note that the name is Print Setup, not Printer Setup. The latter name is now reserved for the system dialog that changes systemwide printing defaults.

- If the Default Printer option button is selected, the document is printed on the printer established as the default by the system Printer Setup dialog. As a result, if the user changes the system default printer, all documents that were stored with the default printer setting will automatically print on the new printer. For some documents, however, users may want to choose a particular target printer and store that choice with the document so that the document will always print on that printer, even if the system default printer is changed. This capability is provided by the Specific Printer option button and its associated drop-down list. The list entries are in the form "PCL / HP Laserjet on LPT1," "PostScript Printer on LPT2," and so on.
- The Orientation controls allow the user to choose portrait or landscape printing; the icon changes to reflect the choice. In portrait mode, the lines of text or the tops of the graphics are parallel to the short edge of the page; in landscape mode they are parallel to the long edge. If one of the modes is not available for the current combination of printer and paper type, the label and option button for that mode are dimmed, and the option button for the available mode is selected.
- The Paper Size drop-down list allows the user to choose from the list of paper sizes provided by the printer device driver. This list specifies the paper sizes that can be loaded into the printer, not the sizes that are actually loaded at the time of printing. The list may also include custom forms added in the printer properties dialog provided by the printer device driver. If the current printer does not support the paper size that was stored with the document, the default size for the current printer is used.
- The Paper Source drop-down list specifies the source of the paper (for example, Upper Tray, Lower Tray, or Manual Feed).
- The Options button leads to a dialog that allows the user to select additional driver-supplied printing options (for example, duplex printing) that will be stored with the document. This button is dimmed if no such options are provided by the current printer driver.

8.3 Text Search and Substitution

Most of the recommendations and design decisions for the common Find and Replace dialogs are shown in Figures 8.7 and 8.8; the key points are summarized below. The recommendations have the following order of priority: (1) Command names; (2) Names and behaviors of controls within dialogs; (3) Type of dialog (modal versus modeless) and menu location of commands.[5]

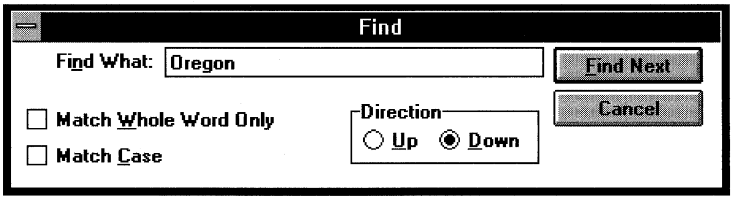


Figure 8.7 Find Dialog

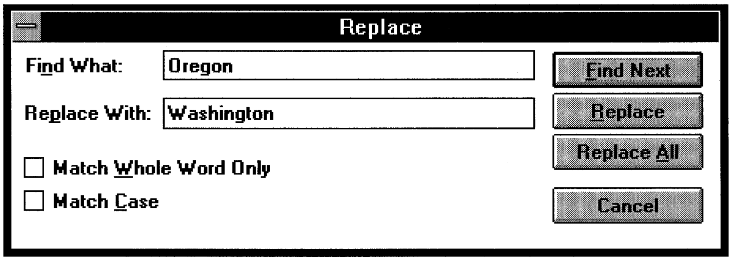


Figure 8.8 Replace Dialog

8.3.1 Command Names and Menu Location

The text search command should be named Find (rather than Search) in all applications. "Find" is a more common, familiar word. The substitution command should be named Replace (rather than Change, which is too vague).

The Find and Replace commands should both be on the Edit menu. Replace is clearly an editing operation; Find is less so, but is placed on the same menu as Replace because it is a logical component of that command. Placing both commands on the Edit menu, which is present in all applications, will make the commands easier to find for users of many different applications.

[5] As shown in the illustrations, the Find and Replace dialogs are very similar. Applications may consolidate the two dialogs and provide access to these functions through a single command called Find/Replace on the Edit menu. If a single dialog is used, its design should be consistent with the guidelines for the Find and Replace dialogs.

8.3.2 Dialog Type and Operation of Commands

If possible, the dialogs invoked by the Find and Replace commands should be modeless. If making the dialogs modeless is too difficult, they should at least have a title bar and be movable, to let the user expose text hidden by the dialogs.

8.3.2.1 Operation of the Find Command

Searches in the Find dialog may proceed forward or backward, according to the option chosen with the option buttons at the bottom of the dialog (see Figure 8.7). Searches begin at the current insertion point or at the beginning of the current selection. The top command button in the Find dialog is always labeled Find Next. This button causes the search to proceed in the specified direction until the next instance, if any, of the search text is found. Each time an instance is found, the document is scrolled behind the dialog box to show the text and some context before and after the text.

When a forward search reaches the end of the document or a backward search reaches the beginning, a message is displayed. For forward searches, the following text is suggested: "No matches found. Continue search from beginning of document?" (where "document" may be replaced by "spreadsheet," "presentation," and so on as appropriate). For backward searches, the text reads, "Continue search from end of document?" In both cases, the message dialog contains two buttons below the text: the default button Yes on the left, and No on the right. Pressing either button closes the message but leaves the Find dialog open. Pressing Yes continues the search from the appropriate boundary of the document. If the other boundary of the document is reached again, another message is shown, with the text "No matches found. <*Boundary*> of document reached" (where <*Boundary*> is "Beginning" or "End" as appropriate). This message contains a single button labeled OK beneath the text. Pressing OK closes the message but leaves the Find dialog open.

8.3.2.2 Operation of the Replace Command

The Replace command invokes a single four-button dialog that stays open until the user has finished replacing text. The single-dialog model allows easy adjustment of replacement text in the middle of a series of replacements, without requiring the user to start over again.

The Replace dialog uses the document-boundary messages described for the Find dialog. If the user changes the search string in the middle of a series of replacements so that it no longer matches the current selection, the search is considered to have restarted from the beginning of the selection. This new starting location is used to determine document-boundary messages.

The four command buttons for the Replace dialog are arranged as shown in Figure 8.8.

- The top button is always the default button and is always labeled Find Next. This button finds the next instance, if any, of the search text without replacing the current instance.
- The second button is labeled Replace. This button replaces the current instance of the search text and finds the next instance. The button is active when a selection matches the search string; otherwise it is dimmed.
- The third button is labeled Replace All. If there is a selection when the dialog opens or is reactivated, the button replaces all instances of the search text in the selection and retains the selection. If there is no selection when the dialog opens or is reactivated, this button replaces all instances of the search text from the current location to the end of the document. The document-boundary message discussed previously allows the user to continue the replacement operation from the beginning of the document.
- The bottom button, Cancel, closes the dialog. The label of this button should change to Close after the first replacement if replacements cannot be undone.

If the user uses the Find or Replace command to search for text, changes the cursor location, and then continues the search, the next search starts at the new location. If the user switches to another document window, the search proceeds from the current cursor location of that window.

8.3.3 Labels

The Search text should be labeled Find What. Replacement text should be labeled Replace With.

The check box for limiting matching to whole words should be labeled Match Whole Word Only. The check box for turning on case sensitivity should be labeled Match Case.

8.3.4 Other Controls

8.3.4.1 Direction Controls

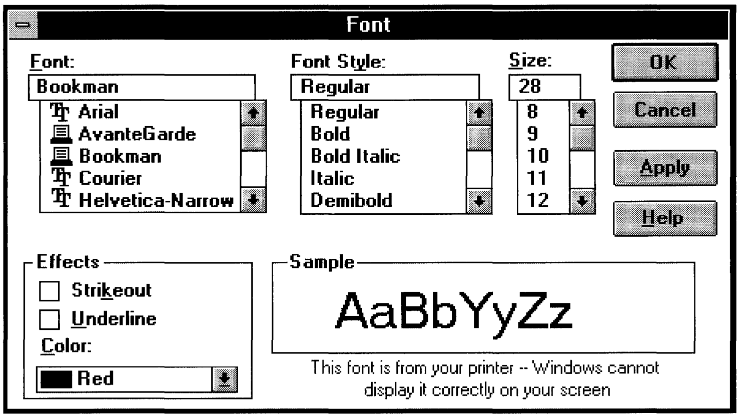
All applications should include options for forward and backward searches in the Find dialog. The simplest case will have only two such options (labeled either Up and Down, or Forward and Backward), represented by option buttons in a group box labeled Direction. Applications that include variations on these options (for example, Forward from Start, Forward from Here) may use a drop-down list in place of option buttons.

8.3.4.2 Application-Specific Options

An Options button can be placed beneath the Cancel button in the Find dialog and to the left of the Cancel button in the Replace dialog. This button unfolds the dialog to include application-specific options (see Chapter 7, section 7.1.2). If these options are used frequently, the application can provide easier access to them by omitting the Options button and simply adding the options to the bottom of the dialog.

8.4 Character Properties

Figure 8.9 shows the recommended format for the dialog used for changing character properties such as font, size, type style, and color.



Font: Bookman
- Arial
- AvanteGarde
- Bookman
- Courier
- Helvetica-Narrow

Font Style: Regular
- Regular
- Bold
- Bold Italic
- Italic
- Demibold

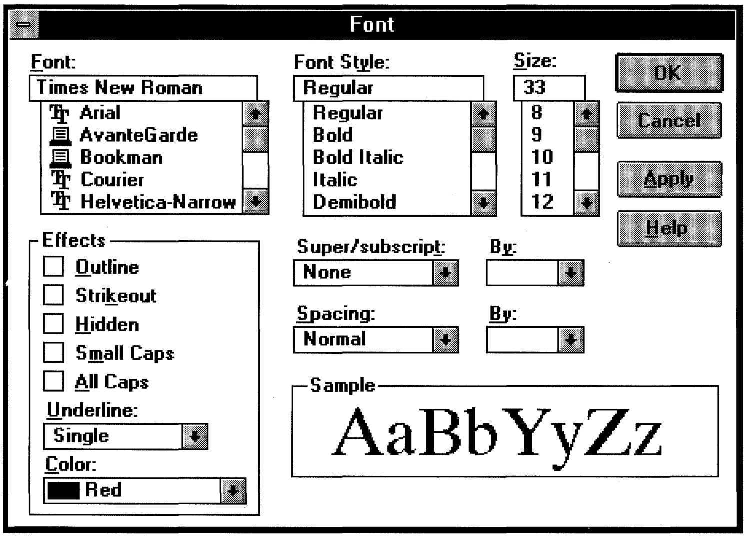
Size: 28
- 8
- 9
- 10
- 11
- 12

Strikeout ☐
Underline ☐
Color: Red

This font is from your printer -- Windows cannot display it correctly on your screen

Figure 8.9 Basic Version of Character Properties Dialog

Applications that require advanced features can add them at the bottom of the dialog. Figure 8.10 shows an example of how such features can be added.


Font

Font: Times New Roman

Font Style: Regular

Size: 33

Super/subscript: None

By:

Spacing: Normal

By:

Underline: Single

Color: ■ Red

Figure 8.10 Example of Enhanced Version of Character Properties Dialog

Adding features may involve adding new controls, altering old ones, or repositioning otherwise unchanged controls to make the best use of space, but the overall structure of the dialog should be preserved as much as possible. In Figure 8.10, for example, the Underline check box has been changed to a drop-down list to allow more underlining options, and the Sample box has been moved to make room for the Super/Subscript and Spacing controls, but the appearance and position of the buttons and the Font, Size, and Color controls remain essentially unchanged from the basic dialog.

Applications that offer advanced features should use an expanded dialog rather than an unfolding dialog, for two reasons:

- First, for many users, the only reason to invoke the dialog at all is to access the advanced features because the basic ones are usually available in other ways (for example, from menus, from a ribbon, or through keyboard shortcuts). Accordingly, the advanced features should be immediately available without the intermediate step of pressing an Options button.
- Second, it is difficult to construct a space-efficient basic dialog that neatly unfolds into an advanced version through simple addition of elements. As Figures 8.9 and 8.10 show, the basic dialog has to be somewhat reorganized to accommodate new controls. Having this reorganization suddenly occur after the user presses the Options button would be visually disruptive.

8.4.1 Character Dialog Box Controls

8.4.1.1 Choice of Controls

The font and size controls are combo boxes rather than lists for two reasons: speed, because users can quickly type in names rather than seeking them in the lists; and flexibility, since users who don't currently have a particular font or size available can type its name, even though it doesn't appear in the list; the name will be stored for later use.

8.4.1.2 Font Combo Box

This combo box lists font family names. If the user selects a printer font, the following message is displayed under the sample box: "This font is from your printer—Windows cannot display it correctly on your screen." If the user selects a screen font, the following message is displayed: "This is a Windows screen font—it may not print correctly on your printer."

8.4.1.3 Font Style Combo Box

When the user selects a font style (that is, a particular weight, slant, or combination of weight and slant), the system attempts to provide a built-in typeface corresponding to the selected style. For example, if the current font is Helvetica and the user checks Bold and Italic, the system uses the built-in Helvetica Bold Italic typeface. If a built-in font is not available, the system synthesizes it and displays the following message under the sample box: "This font style is simulated by Windows—it may not print correctly on your printer."

8.4.1.4 Size Combo Box

This combo box contains a list of point sizes. Users may type any size, but the list box should provide the following sizes for TrueType™ and vector fonts: 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 26, 28, 36, 48, 72. (For fonts that are not infinitely scalable, only actually available sizes should be listed.) Applications may alter this list to suit the needs of their users; for example, desktop presentation programs may want to include additional large sizes.

8.4.1.5 Effects Check Boxes

These check boxes allow the user to choose effects such as strikeout and underline, which do not involve weight and slant, and typically do not correspond to built-in typefaces.

8.4.1.6 Color List

This drop-down list may contain color patches followed by color names. Using a list rather than a palette is recommended, because a palette would give too much space and emphasis to a relatively infrequently used part of the dialog.

The list will contain the eight most commonly used standard system colors (black, white, red, green, blue, yellow, cyan, and magenta) and additional colors if the application provides them. On monochrome systems, the color patches will be mapped to patterns. The choices available in the list should be independent of what is currently available on the printer and on the display. Optionally, applications may include an Auto list entry, which means "Use black on the printer and the system default text color on the screen."

Applications that provide a large set of colors may replace the color drop-down list by a patch of the current color next to a button labeled Color, which leads to a color selection dialog. The form of the dialog is not addressed in this guide.

8.4.1.7 Sample Box

The sample box contains the string "AaBbYyZz" displayed in the selected typeface with all the selected attributes. "AaBbYyZz" includes one character with an ascender and one with a descender, and will present fewer internationalization problems than a sample word. The sample box should be at least large enough to accommodate a 24-point font.

8.4.1.8 Command Buttons

The dialog contains two required command buttons (OK and Cancel) and two optional buttons (Apply and Help).

- The OK button accepts all changes and closes the dialog.
- The Apply button applies all changes but leaves the dialog open. Although this button is optional, we suggest including it whenever possible. The Apply button is useful whether the dialog is modal or modeless. In modal dialogs, it allows the user to see the effect of changes one step at a time; this makes it easier to reverse a decision by resetting the property that was just changed. (The sample box also shows the effect of changes but isn't sufficient by itself; the user must press Apply to see the changes. Also, the sample box shows only a few characters.) In modeless dialogs, Apply is useful for the same reasons. In addition, it allows the user to apply changes to one piece of text and then select a different piece of text and apply different changes, without having to close and reopen the dialog.

- The Cancel button changes to Close once any changes have been applied. This button closes the dialog, ignoring any changes made since the last Apply command but accepting all previous changes.
- The optional Help button allows the user to obtain further information about the current font family or about the current combination of font family, size, and attributes. For example, the Help information might provide advice on the most appropriate uses for particular fonts. The exact content of Help information is determined by the application.

8.4.1.9 Effect of Undo

If the dialog is modeless, it would be difficult for Edit Undo to restore all character properties to their state before the dialog opened; too much could have happened in the meantime. Instead, Undo should have one of the following effects (for consistency, the same recommendations apply if the dialog is modal):

A. Reverse all formatting changes made to the last selection.

B. Reverse all formatting changes committed by the last Apply or OK. (If the user commits some changes with Apply and then presses OK without making further changes, Undo should cancel the changes made by Apply rather than doing nothing.)

Option A is preferable, but B is acceptable if implementing A is too difficult. Here is an example of how the two options would work:

1. User makes selection, opens dialog, checks Bold, presses Apply.
2. Bold formatting is added to the selection in the document window.
3. User checks Italic and Underline, presses Apply.
4. Selection is italicized and underlined.
5. User activates document window, either by clicking on it or by pressing OK or Close to close the dialog.
6. User chooses Edit Undo. (Undo is dimmed if document window isn't active.)
7. In case A, the italics, underlining, and bold formatting are removed from the selection; in case B, the italics and underlining are removed, but the bold formatting is left intact.

8.5 Page Setup Dialogs

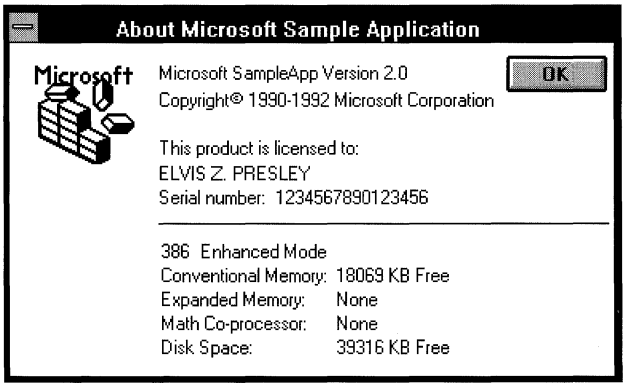
There is no common dialog for setting page margins, but there is a standard layout for text boxes used to set top, bottom, left, and right page margins. Consider the following two layouts:

Option A:	Top	Left
	Bottom	Right
Option B:	Left	Right
	Top	Bottom

Option A is the suggested layout because it places the two members of a natural pair (two opposites) closer to each other. The arrangement also makes it easier to compare the values of pair members. This is useful because the user often wants to set paired margins equal (for example, top and bottom margins of 1 inch, left and right margins of 1.25 inches).

8.6 About *<Application-Name>* Dialog

Figure 8.11 shows the common About *<Application-Name>* dialog. This dialog should be accessed by an About *<Application-Name>* item on the Help menu. Some elements of the dialog are recommended; others are optional.



Microsoft SampleApp Version 2.0
Copyright© 1990-1992 Microsoft Corporation

This product is licensed to:
ELVIS Z. PRESLEY
Serial number: 1234567890123456

386 Enhanced Mode
Conventional Memory: 18069 KB Free
Expanded Memory: None
Math Co-processor: None
Disk Space: 39316 KB Free

Figure 8.11 About *<Application-Name>* Dialog

8.6.1 Recommended Elements

The dialog has a title bar that includes a title in the form About <*Application-Name*>, where <*Application-Name*> is the official name of the application. The first line of the dialog repeats the official name of the application. The second line displays the version number. The next line contains a copyright statement in the form "Copyright (c) 19xx-19xx <*Corporation-Name*>." An icon (or other graphic) associated with the application appears to the left of these lines; this is typically the icon used to identify the application on the desktop, but other icons may also be used. An OK button appears in the top-right corner of the dialog.

8.6.2 Optional Information

The licensing information above the horizontal black line in the dialog is optional. The system information below the line is optional but highly recommended. If present, it should be formatted as shown in Figure 8.11. Note that the labels are left-aligned, following standard dialog box style. If this information is not present, the black line should be omitted. Applications may also add graphics, animated icons, and logos, as appropriate, to the dialog box.

Object Linking and Embedding

Chapter 9

This chapter introduces object linking and embedding (OLE), which is the process of creating compound documents that contain embedded and linked objects. OLE can best be understood through the concept of the compound document.

9.1 Compound Documents

Over the last decade, productivity applications have become sophisticated managers of specific types of information; for example, spreadsheets, databases, charts, richly formatted text, and so on. This specialization has resulted in an inability to create documents that integrate different types of information; in general, it is difficult to include charts in a spreadsheet, or tables and figures in a text document.

This frustrates users who want to create documents that integrate several types of information through one graphical interface, without switching between applications and without using cumbersome methods to assemble and to maintain integrated information over time. Users want a consistent way to manipulate a given type of information without dealing with different interfaces. For example, it is difficult enough to master one drawing tool, let alone different tools for different applications; users want one drawing capability that they can access from all applications.

The answer lies in the concept of compound documents. A compound document is a container document that includes components from various source applications. The compound document provides the framework for housing different components and for invoking their respective applications.

In a compound document, users can point to any location and insert any kind of information—for example, text, a table, a picture, or a chart. If the user wants to insert a chart, the container application invokes a charting application. If the user wants to insert a table, the container application invokes a spreadsheet application, and so on.

In an ideal implementation of compound documents, the user is unaware that different source applications are being invoked. The process of browsing, selecting, and editing information is seamless; users can manipulate various types of information within the body of a single document without the inconvenience of switching from one application to another. They can create a single document that either links (references) or embeds different packets of information (objects); hence the term "object linking and embedding."

OLE is the process of creating compound documents that contain embedded and linked objects. These objects can be of the same type or of different types. For example, the bulk of a Microsoft Word for Windows document (client.doc) shown in Figure 9.1 is probably text, but it also contains a link to a Paintbrush™ object illustrating the company's logo and embeds a Microsoft Graph object summarizing the company's monthly expenses over the last year. Objects and their applications come in all shapes and sizes—voice objects, audiovisual objects, equation editors, graphic designers, and so on. The power of OLE, however, is limited only by the user's imagination (and perhaps by hardware).

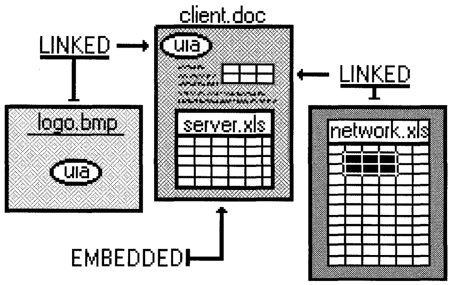


Figure 9.1 Linked and Embedded Objects in Word Document (client.doc)

9.2 OLE Concepts

- **Object.** Objects are information entities (for example, text, graphics, sound, video, and so on) that are the components of a compound document. An object is like an opaque package that contains the source application's data (or a linked reference to that data) and the name of the source application. OLE objects are "opaque" because the container application never looks inside the object and therefore doesn't need to understand its contents or its format. Instead, the container application simply calls application programming interfaces (APIs) in the OLE dynamic link library whenever it needs to display an object or wants to invoke the object's source application program.
- **Class.** The object class describes the type of information contained within an object and is assigned by the server application. Examples of Microsoft object classes include Drawing (Microsoft Draw), Picture (Microsoft Paintbrush), Worksheet (Microsoft Excel), Document (Microsoft Word), and Chart (Microsoft Graph).

- **Client.** The client is the container application that produces the container document. The use of the term "client" in OLE is similar to its use in network terminology. That is, the application that receives and stores the object is the client. For example, if a Microsoft Excel chart is embedded within a Word document, Word is the client application, the Word document is the container document, and the Microsoft Excel chart is the embedded object.
- **Server.** The server is the source application that produces the embedded or linked object. The use of the term "server" in OLE is similar to its use in network terminology. That is, the application that remotely creates, edits, and displays (or plays) the object is the server. For example, if a Paintbrush picture is embedded within a Word document, Paintbrush is the server application and the Paintbrush picture is the embedded object. In object linking, the server produces a source document from which the container document can extract information (or an image thereof).
- **Package.** A package is a special type of OLE object that contains an OLE object, a file, or a command line. It is represented by an arbitrary and selectable icon or other graphic. Double-clicking a packaged object activates the object inside the package. Packages present compact tokens of large files or OLE objects. They also provide some of the functionality associated with hyperlinks.
- **Embedding.** Embedding is the process of inserting a new or an existing object into a container document (see Figure 9.2). All information normally stored in a file created by a server is instead embedded into the body of a file created by the client. This allows all components of a compound document to be stored in a single file. The user can spend more time composing and updating a single document and less time dealing with the bookkeeping of multiple source documents. For example, embedding a chart within a Word document is easier than remembering the location of the chart on disk.
- **Linking.** Linking achieves the effect of copying information from one document into another without actually making a physical copy (whereas in embedding the actual information is stored within the document). When the user links information from a source document into a container document, the information appears inside the container as if it had been physically copied there. In fact, the container simply contains a link, that is, a reference to where the information exists in the source document and where the source document can be found. The container document uses the link to get this information when needed—for example, when updating the display (sometimes a bitmap or metafile will be placed into the container as a facade). Links provide an effective way for documents on a local drive or documents distributed over machines on a network to share information. For example, Figure 9.3 shows how a Microsoft Excel worksheet that contains a summary of monthly financial transactions can be used for linking. The user can link the summary lines (omitting the raw data) into an end-of-month document for the manager and link all the data over a network into a database at corporate headquarters. Updating the monthly transactions automatically updates the end-of-month document and the corporate database.

Compound documents can contain any number of embedded and linked objects.

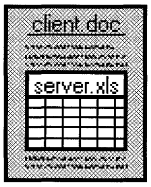


Figure 9.2 Embedding (Microsoft Excel Worksheet Embedded Within Word Document)

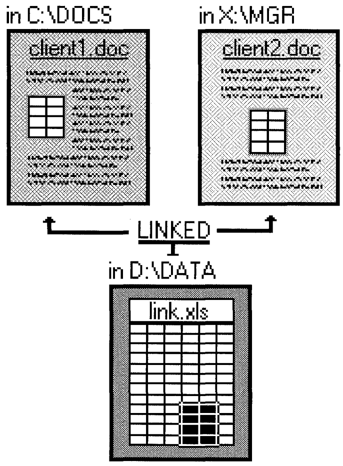


Figure 9.3 Linking (Separate Word Documents Link Fragment of Microsoft Excel Worksheet)

9.3 OLE Interface

The OLE interface provides easy methods for inserting, editing, viewing, and activating linked and embedded objects, and modifying the properties of such objects.

9.3.1 Clients and Servers

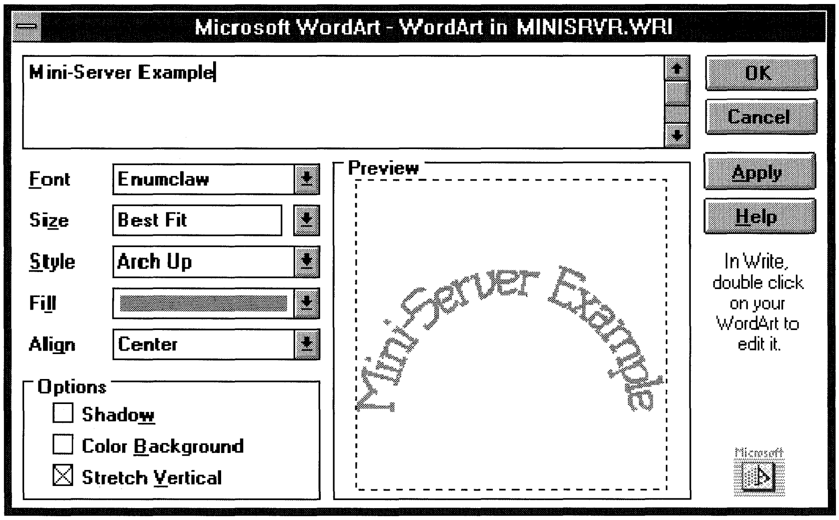
The OLE process requires a dialog between client and server. Any application is capable of being a client, a server, or both. In addition to the changes in functionality, there are several key differences in the interface of clients and servers.

- Clients have four commands added to the Edit menu below the Paste command: Paste Special (and/or Paste Link[1]), Links, Object, and Insert Object, (see Figure 9.12). If there is an Insert menu, the Insert Object command may be placed there as Object instead.
- There are two types of servers: full servers and mini-servers. The title bar for both server types should read: "<*Server*> - <*Descriptive-Class-Name*> in <*Container-Document*>" (see Figure 9.4). An object class must always be presented to the user (in window titles, list panes, dialogs, and so on) in the human-readable format provided by the registration database. The client application must supply an appropriate <*Container-Document*> string to the server. Because link servers open the source document, their title bars display the standard information: "<*Server*> - <*Source-Document*>".

9.3.1.1 Mini-Servers

A mini-server looks like a dialog box and requires three command buttons: OK, Cancel, and Help. (The example mini-server in Figure 9.4 includes an optional Apply button.)

[1] Paste Link is optional if you have the preferred Paste Special command discussed in section 9.3.2.4.


Microsoft WordArt - WordArt in MINISRVR.WRI

Mini-Server Example

Font: Enumclaw
Size: Best Fit
Style: Arch Up
Fill:
Align: Center

Options
- [ ] Shadow
- [ ] Color Background
- [x] Stretch Vertical

In Write, double click on your WordArt to edit it.

Figure 9.4 Mini-Server Interface

9.3.1.2 Full Servers

A full server is a stand-alone application with a full set of menus. In a full server (or in a server window in an MDI application), the File menu should be modified as follows (see Figure 9.5):

- The Save command should be replaced by Update <*Container-Document*>.
- The Close command (MDI applications) should be replaced by Close & Return to <*Container-Document*>.
- The Save As command should change to Save Copy As.
- The Exit command should be replaced by Exit & Return to <*Container-Document*>. The client application must supply the appropriate <*Container-Document*> string to the server.

Additional note: SDI applications should not implement New and Open when running as servers. It is recommended that these commands be replaced by the Import command, which would allow the loading of an existing file (object) without severing the connection with the client. (See section 9.3.5.1.)

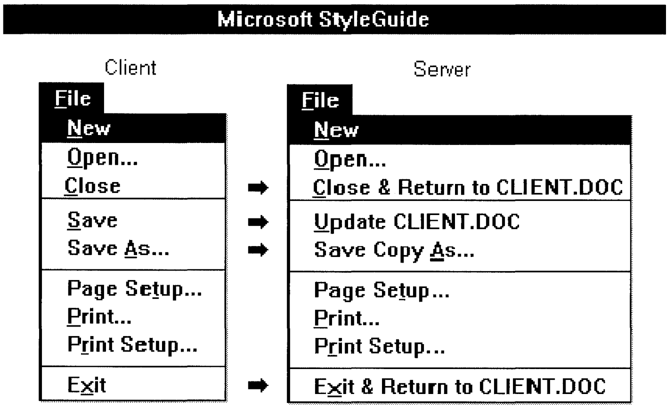
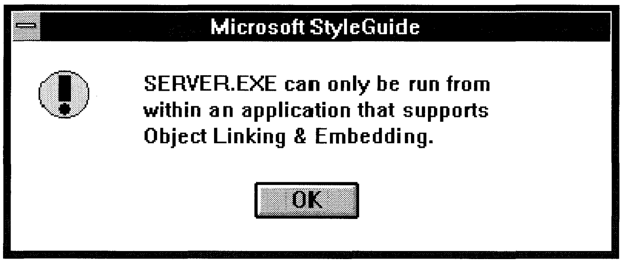


Figure 9.5 Full Server Interface Changes

If the user attempts to launch a server (for example, a mini-server) that cannot be run as a stand-alone application, the error message shown in Figure 9.6 should be issued.


Microsoft StyleGuide

SERVER.EXE can only be run from within an application that supports Object Linking & Embedding.

OK

Figure 9.6 Warning Message When a Server Cannot Be Run Stand-Alone

9.3.2 Inserting Objects

The basic user interface for inserting linked and embedded objects relies on the familiar Cut, Copy, and Paste commands on the Edit menu, plus one additional command—Paste Special (and/or Paste Link)—for inserting linked objects and for providing additional control over data formats. To accelerate object embedding, applications should implement the Insert Object[2] command. The use of Copy and Paste is not limited to selected document fragments. Entire files may be linked or embedded into documents and displayed as double-clickable icons (that is, packages) with Copy/Paste or by using drag/drop from the File Manager.

[2] Paste Special and Insert Object are optional only if a cogent argument can be made for their omission. Paste Link is optional if the preferred Paste Special is used.

9.3.2.1 Inserting Embedded Objects with the Insert Object Command

To accelerate the procedure for embedding new objects, applications should implement an Insert Object command. In applications that include an Insert menu, the command should appear on that menu and should be called Object. In applications that do not include an Insert menu, the command should appear on the Edit menu and should be called Insert Object. The command leads to the dialog shown in Figure 9.7.

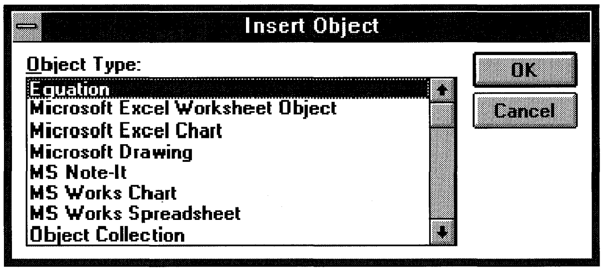
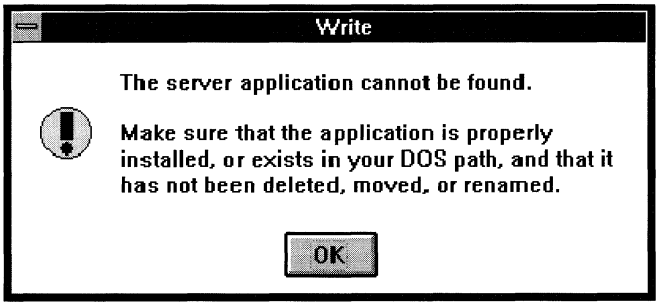


Figure 9.7 Insert Object Dialog Box

The Object Type list box in the Insert Object dialog contains the descriptive object class names drawn from the registration database maintained by Windows. The client application must sort and display the list in alphabetical order. To embed an object, the user selects its name from the list and presses OK. This results in the following:

- A rectangle (default size is determined by the client application) immediately appears at the insertion point as an interim placeholder until the new object image is available. The rectangle is masked with the open visualization (see Figure 9.13) to indicate that its server is currently open.
- The server application is launched for the selected object type. The server displays a blank or default window in which the user can create or edit the object.

If the client application fails to locate the requested server when the user selects the entry from the Insert Object dialog or double-clicks on an object, the following error message should be displayed.

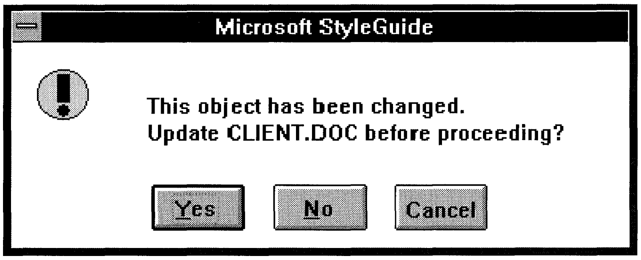


The server application cannot be found.

Make sure that the application is properly installed, or exists in your DOS path, and that it has not been deleted, moved, or renamed.

Figure 9.8 Warning Message When Server Application Cannot Be Found

While editing, the user may select the Update <*Container-Document*> command from the server's File menu at any time to place the current rendition of the embedded object in the client at the current cursor location. (This will replace the placeholder if it is the first update.) After editing is complete, the user selects the Exit & Return to <*Container-Document*> command from the File menu of the server. This command closes the server and returns focus to the container document. If the user does not choose the Update command for a modified embedded object before exiting, the prompt shown in Figure 9.9 is displayed.



This object has been changed.
Update CLIENT.DOC before proceeding?

Figure 9.9 Warning Message When Exiting Server with a Modified Embedded Object

9.3.2.2 Inserting Embedded Objects with Cut, Copy, and Paste

Although the Insert Object command for embedding objects is preferred, most users are familiar with the Cut, Copy, and Paste commands and will want to use these commands. The basic procedure for inserting embedded objects with Cut, Copy, and Paste is simple. For example, inserting a drawing into a text document requires the following steps:

- In the server application for the drawing, select all or part of the drawing and choose the Copy (or Cut) command from the Edit menu.

- Switch to the text application.
- In the text application, position the cursor as desired and choose the Paste command from the Edit menu.

This procedure inserts the drawing at the current cursor location in the text document.

In general, for applications that support OLE, the Paste command will embed the object that is on the clipboard. Under some circumstances, however, the application may choose not to embed the object. In particular, if the object is represented on the clipboard not only by the embedded object format but also by an alternate format that completely represents the original data and that the application knows how to edit, the application should insert this editable data rather than embedding the object. Most applications, however, will not be able to provide full editing capabilities for alternate formats, so they will simply embed the object and allow the source application to later function as a server.

Thus, in most applications that support OLE, the Paste command will embed the object in the current document when the clipboard contains an object from another application. Note that in applications that do not support OLE, the Paste command simply inserts a static copy of the data without providing easy access to the tools required to edit the data.

9.3.2.3 Inserting Linked Objects with Copy and Paste Link

Inserting linked objects is as easy as inserting embedded objects. To continue the previous example, suppose that instead of embedding the drawing, the user wanted to insert a link to the drawing. The following steps would be required:

- In the drawing application, select all or part of the drawing and choose the Copy command from the Edit menu. (Do not use Cut because it eliminates the source.)
- Switch to the text application.
- In the text application, position the cursor as desired and choose the Paste Link (or Paste Special) command from the Edit menu.

Note that the only difference between linking and embedding an object is the selection of Paste Link instead of Paste in the final step.

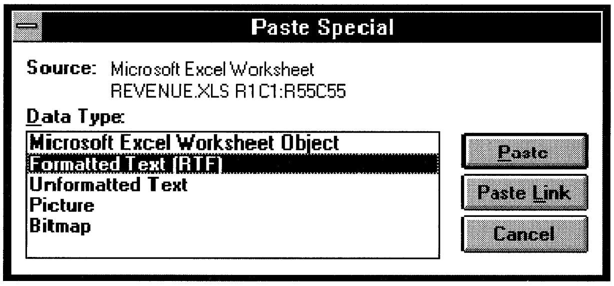
This procedure inserts a linked object to the drawing at the current cursor location in the text document. The drawing is displayed in the text document but stored in the original drawing file. The link is an "automatic" link; in other words, whenever the drawing file changes, the drawing in the text file is updated automatically. Automatic and manual links are described more fully in section 9.4.

9.3.2.4 Selectively Inserting Linked and Embedded Objects with the Paste Special Command

Some applications will need only the simple methods discussed in the preceding sections for inserting linked and embedded objects. Most applications, however, may want to provide additional control with the Paste Special command.

Some applications can interpret and edit data in a variety of formats—for example, formatted text (that is, rich text format or RTF), ASCII text, bitmaps, and object-oriented picture formats. These applications may provide a Paste Special command on the Edit menu to provide greater control over the format to be pasted.

By default, the standard Paste and Paste Link commands look on the clipboard for a format that completely represents the original data and that the client application can edit with its own tools. If such a format is found, the original data is translated into that format and inserted into the client; if not, an image representing the original data is usually inserted. In some cases, however, the user may want to override the default format. For example, special tools or editing operations (for example, rotation, translation, applying format) might be available only for a nondefault format (for example, copying the format, but not the contents, of a paragraph to another paragraph). Alternatively, the user might want to force the data to be inserted as an embedded object so that the tools of the original creator application can easily be invoked to edit the object. These format choices are supported by the Paste Special dialog shown in Figure 9.10.[3]


Paste Special

Source: Microsoft Excel Worksheet
REVENUE.XLS R1C1:R55C55

Data Type:

Microsoft Excel Worksheet Object
Formatted Text (RTF)
Unformatted Text
Picture
Bitmap

Figure 9.10 Paste Special Dialog Box

[3] Although not shown, a Help button may be included in the Paste Special dialog box if elaboration on data types or paste behavior is useful.

The Data Type list in the Paste Special dialog shows the formats available on the clipboard that the client can process. The names of these formats should clearly suggest the resulting capability of the pasted information. The list also includes one additional entry, <*Descriptive-Class-Name*> Object (for example, Microsoft Excel Worksheet Object). When the dialog opens, the format that would have been used if the Paste command had been selected from the Edit menu is highlighted. The dialog contains two command buttons, Paste and Paste Link, which operate as follows:

- The Paste button translates the data into the selected format and inserts it into the document without establishing a link to the original source of the data. (If the user does not change the default format that is originally selected in the list, pressing Paste in this dialog is equivalent to choosing Paste from the Edit menu.) If the user selects <*Descriptive-Class-Name*> Object instead of a data format, the data is not translated into a native format, but inserted as an embedded object.
- Like Paste, the Paste Link button translates the data into the selected format and inserts it into the document. (If a particular data type cannot support a link, the Paste Link button is dimmed.) Unlike Paste, however, Paste Link establishes an automatic link to the original source of the data. Because the Paste Special dialog provides a superset of the functionality provided by the Paste Link command, applications that include Paste Special need not also include Paste Link. The Paste Link button should be dimmed only when the clipboard does not contain an object link format. Paste-linking <*Descriptive-Class-Name*> Object behaves like any other link—that is, it inserts a facade into the container document along with a reference to the source file.

9.3.2.5 Copying Linked Objects

When a linked object exists in one document, it can be inserted in other documents with Copy and Paste (not Paste Link). For example, suppose that Document1 contains Link1, which refers to a bitmap in a graphics file (see Figure 9.11). If Link1 is selected, the Copy command copies Link1, not the bitmap that is the source object. The Paste command then inserts a copy of Link1 (which we can call CopyOfLink1) at the current cursor location. Like Link1, CopyOfLink1 refers to the original bitmap. If Paste Link is used instead of Paste, a link to (not a copy of) Link1 is inserted. This new link (LinkToLink1) refers to Link1, which in turn refers to the original bitmap. Link1, CopyOfLink1, and LinkToLink1 all yield exactly the same visual presentation (a representation of the original bitmap) in their client documents, but Link1 and CopyOfLink1 refer directly to the bitmap, whereas LinkToLink1 refers to the bitmap only indirectly, through Link1.

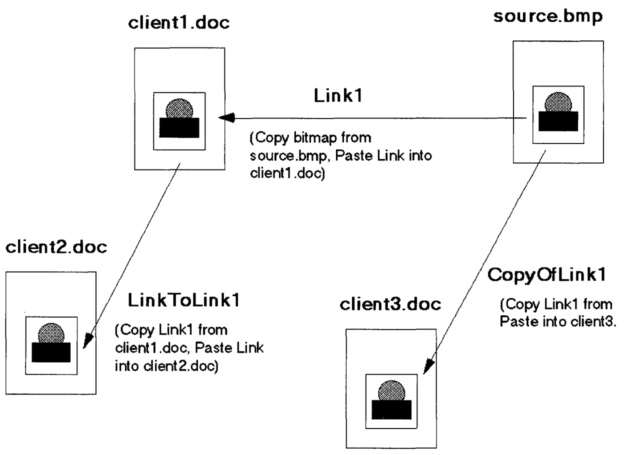

client1.doc
source.bmp
Link1
(Copy bitmap from source.bmp, Paste Link into client1.doc)
client2.doc
LinkToLink1
(Copy Link1 from client1.doc, Paste Link into client2.doc)
client3.doc
CopyOfLink1
(Copy Link1 from Paste into client3.

Figure 9.11 Transferring Links: Paste vs. Paste Link

Just as linked objects can be copied from one location to another with Copy and Paste, they can also be moved with Cut and Paste. In this respect, linked objects behave like all other objects that can be manipulated from the clipboard.

9.3.3 Viewing Objects

Compound documents may contain several OLE objects interspersed with objects that are native to the document. Because OLE objects support operations different from those supported by the native objects, it is convenient to be able to visually distinguish the two. For this purpose, applications should provide visual indications of OLE object boundaries. The boundaries for a particular object should appear whenever the object is selected. In addition, applications should provide a way for all object boundaries to be turned on or off at once, to facilitate easy viewing of all OLE objects in a document. For example, applications can reveal object boundaries and other normally hidden information with a Hidden Structure command on the View menu or a Show All check box in an Options dialog.

Figure 9.12 shows recommended boundaries for linked and embedded objects.

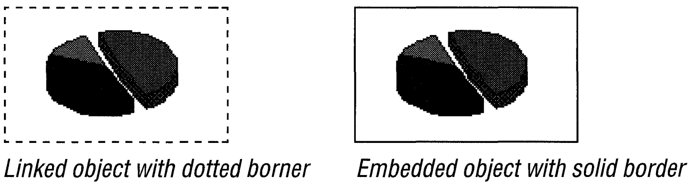

Linked object with dotted borner
Embedded object with solid border

Figure 9.12 Recommended Boundaries for Linked and Embedded Objects

In addition, when an embedded or packaged object is open in the server application, its appearance should be masked in the container document. The masking is also applied to objects whose representation is an icon (like sound). There is no masking for open linked objects.

Figure 9.13 shows recommended visuals for the inactive, selected, and open (embedded or packaged objects only) states of an object.

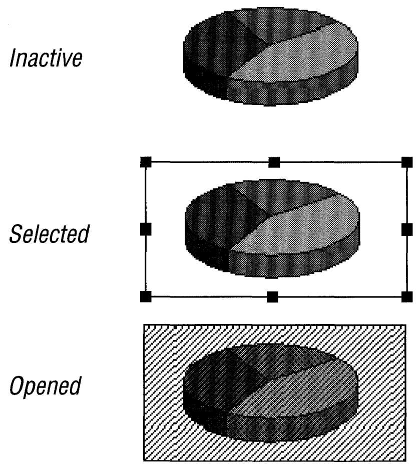


Figure 9.13 Visual Appearance Recommendations

9.3.4 Activating Objects

When OLE objects are inserted into a document, the standard user interface provides two methods for editing the objects: by double-clicking and through the Edit menu *<Descriptive-Class-Name>* Object command.

9.3.4.1 Double-Clicking

Double-clicking on the object boundary (or selecting the object and pressing ENTER[4]) invokes the server application associated with the object. If the object is unitary (that is, if only the whole object can be selected in the container document) the user can double-click anywhere on the object to invoke the server.[5] If the server is an SDI server, a new instance of the server is started, even if one is already running. This tight connection between the new instance and the current object helps promote the impression that the object is an integral part of the compound document in which it is displayed.

There are three exceptions to the rule that a new instance of the server should be started to edit an object. No new instance is necessary in the following cases:

1. The object is already open in an existing instance of the server. In this case, the existing instance should be surfaced.
2. The server application is an MDI application and is the same as the client application (that is, the editor for the object is the client itself). The appropriate behavior depends on whether the link is external or internal.
   a. External links: For example, suppose that one Word document contains a link to another Word document. In this case, double-clicking on the linked object should not start a new instance of Word. Instead, if a document window for the source document is already open within the current instance (case 1 above), that document window should be surfaced. Otherwise, a new document window should be opened for it within the current instance of Word.
   b. Internal links: If the link refers to another part of the same document, double-clicking on the linked object should scroll to show the source of the link.
3. If the server is an MDI application that does not use memory efficiently when multiple instances are running, it should make an intelligent decision about what to do. For example, the application can check available memory to see whether running one more instance is likely to cause problems. If so, the application can issue a warning and/or offer the user a choice between starting a new instance or loading the file into an existing instance.

[4] Selection+ENTER need not be implemented in a single or multiple-edit line because, by default, this should replace the selection with a carriage return and a line feed. In this case, the user can activate the object by using the *<Descriptive-Class-Name>* Object command discussed in section 9.3.4.2.

[5] The double-click or (selection+ENTER) starts the server and invokes the primary verb for that object. Sometimes the primary verb is not Edit. Instead, it is an Activate-type verb such as Run (for a script), Play (for a voice note), and so on. The distinction between primary and secondary verbs is described further in the discussion of the *<Descriptive-Class-Name>* Object command in section 9.3.4.2.

9.3.4.2 The *<Descriptive-Class-Name>* Object Command

If an object supports only one verb, it appears as *<Verb-0>* *<Descriptive-Class-Name>* Object on the Edit menu. If an object supports multiple verbs, they appear within a cascading menu; selecting *<Descriptive-Class-Name>* Object from the Edit menu displays a submenu for *<Verb-0>*, *<Verb-1>*, and so on. Scripts, videos, and voice notes are examples of objects that may support multiple verbs such as play, edit, and rewind.[6] The commands should change dynamically in the menus to reflect object class-specific verb names retrieved from the registration database. The menus must access the verbs from the database; no fixed commands should be used. These verbs should be registered as mixed-case strings and follow the general style of menu commands described in Chapter 5. Each registered verb should also have an assigned mnemonic (underlined letter) for keyboard access. Also note that each verb should be a single word to ensure that the status bar messages and menus in the client application (described later) will read correctly.

If a selection contains multiple objects of the same class or of different classes, the object verbs should not be available to the user. The object verbs should appear only when there is exactly one object in the current selection; no attempt should be made to join verbs of multiple objects in a selection.[7]

The most frequent operations for an object should be registered as its primary verb (*<Verb-0>*). Mouse users can invoke the primary verb through double-clicking; keyboard users can invoke the primary verb by selecting the object and pressing ENTER.[8]

Figure 9.14 shows an example of a *<Descriptive-Class-Name>* Object submenu that leads to dynamic, object-specific menu items. The primary verb is accessed by the first (topmost) item, followed by subsequent verbs in order.

[6] When such an object plays, if possible, it should provide a way for the user to interrupt playing and start editing. For example, a voice note can display a control panel with buttons for stop, rewind, and so on.

[7] Verbs are not displayed even if the selection contains objects of the same class, because applying a verb concurrently to the whole set would have ambiguous results.

[8] Selection+ENTER need not be implemented in a single or multiple-edit line because, by default, this should replace the selection with a carriage return and a line feed. In this case, the user can activate the object by using the *<Class>* Object command.

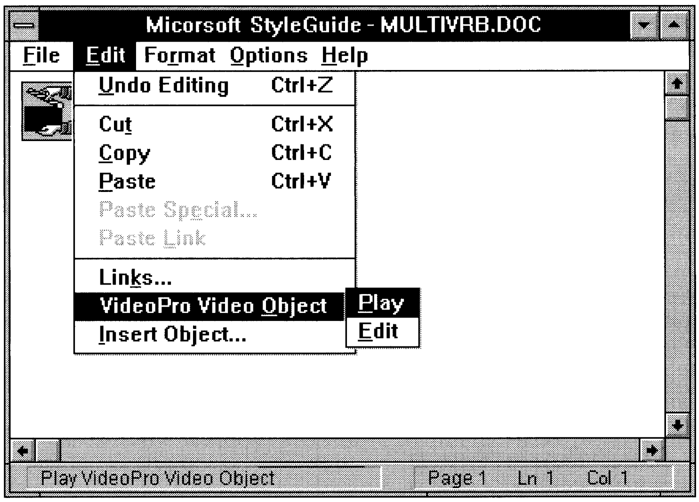


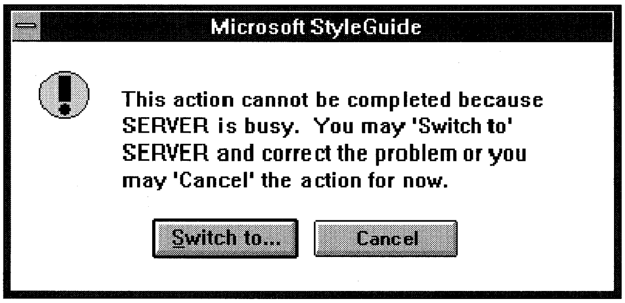
Figure 9.14 ***<Descriptive-Class-Name>*** **Object Command with Object-Specific Verbs**

For packages (which are always embedded, never linked), the primary verb is the primary verb of the object inside the package; the primary menu item name is Activate Contents. This means that the mouse user can invoke the primary verb of the object inside the package by double-clicking on the package icon.

The secondary verb for a package is Edit Package. Selecting this verb invokes the Packager. Some objects (for example, text objects) have no secondary verb. For such objects, the *<Descriptive-Class-Name>* Object command does not lead to a cascading menu; the command simply executes the primary verb. The verb is prepended to the menu item: *<Verb> <Descriptive-Class-Name>* Object. For example, if the object is a Word document, the menu item is Edit Word Document Object.

9.3.4.3 Busy and Unavailable Servers

A server may be busy or unavailable for several reasons. For example, it may be busy printing, it may be waiting for user input to a modeless error message, or it may be hung or accidentally deleted. If the server is not available, the warning message in Figure 9.15 should be displayed. The recommended time between the first request and displaying the dialog is 2-3 seconds.



This action cannot be completed because SERVER is busy. You may 'Switch to' SERVER and correct the problem or you may 'Cancel' the action for now.

Figure 9.15 Server Busy Warning Message

Three different cases can cause the warning message in Figure 9.15 to be displayed: Busy, Blocked, and OLE_BUSY. Table 9.1 describes the behavior of the buttons in each case.

Table 9.1 Behavior Caused by Different States

State	Caused By	Button Pressed	
		Switch To...	**Cancel**
Busy	Client receives OLE_QUERY_RETRY callback notification	Will invoke Task Manager	The OLE operation is discontinued. The dialog is dismissed.
Blocked	Client timed out before getting OLE_RELEASE	Will invoke Task Manager	The OLE operation will continue. The dialog is dismissed.
OLE_BUSY	OLE libraries returned OLE_BUSY from an API call	Will invoke Task Manager	The previous OLE operation will continue. The dialog is dismissed.

9.3.5 Editing Objects

When the server window opens, the object is loaded into the window. For embedded objects, the window is initially sized to show only the portion of the object that was displayed in the client.[9] The user can resize the window to display additional portions of the embedded object. In the case of linked objects, the entire linked file is loaded, and the linked portion is selected. If possible, the server should not come up maximized and should obscure as little of the object in the container document as possible.

[9] Some embedded objects such as spreadsheets may include portions that are not displayed. These hidden portions are included only if the displayed portion draws data from them.

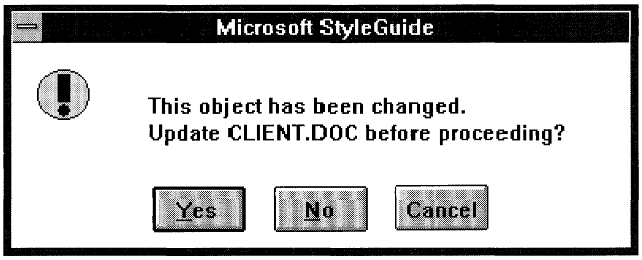
The user can modify the object with the editing tools provided by the server. The process for updating the object in the container file varies depending on whether the server supports the single document interface (SDI) or the multiple document interface (MDI).

9.3.5.1 Updating Objects from SDI Servers

9.3.5.1.1 The Update Command When an SDI application functions as a server for an embedded object, the Save command on the File menu changes to Update, as illustrated in Figure 9.5. (This change occurs only for embedded objects, not for linked objects.) The Update command updates the object in the container document, but (like Save) does not close the server; the server is left open to allow the user to make further changes after seeing the effects (for example, repagination) of the update in the client. If the user tries to exit the server without updating the object, the warning message shown in Figure 9.8 is displayed.

9.3.5.1.2 The Save Copy As, File New, and Open Commands When editing an embedded object, the user can choose the Save Copy As command from the File menu to save a copy of the embedded object in a separate file. Save Copy As does not sever the connection with the client. The inclusion of the New or Open commands in the File menu is not recommended for SDI applications. These commands should be replaced by the Import command, which allows the user to load an existing file (object) into the server application without severing the connection between the server and the client application.

If an SDI application implements the File New and File Open commands, which sever the connection to the client, the warning shown in Figure 9.16 should be displayed.


Microsoft StyleGuide

This object has been changed.
Update CLIENT.DOC before proceeding?

Figure 9.16 Warning Message when Terminating OLE Connection with Modified Object

9.3.5.1.3 Closing (Exiting) a Server or a Client When the server closes, the focus returns to the client. If the user closes a client while servers for that client are still open, the usual client application save confirmation appears. (The container document is considered modified as soon as any object is opened.) If the user chooses to save the container document, all objects are updated without prompting before the file is actually saved. Declining to save the file will likewise discard any modifications made to the objects.

9.3.5.2 Updating Objects from MDI Servers

The procedure for editing and updating objects from MDI servers is similar to the procedure for SDI servers, with the following exceptions:

- If the focus changes from the embedded object window to a "normal" document window (that is, a window containing an existing file or a new document), the File menu reflects that of a client (see Figure 9.5). If the embedded object window regains focus, File menu reflects that of a server again.
- When the user chooses the File New or File Open command, the window containing the embedded object remains open. Therefore, it is not necessary to display any warning messages about updating the object.
- When the user chooses the File Save As command, the client application is informed and the link follows the newly saved file.

9.3.5.3 Operations in Clients Containing Open OLE Objects

9.3.5.3.1 Save Command When the user saves the container document with the File Save command, all open objects are automatically updated in the container document before the document is saved. The server application remains open.

9.3.5.3.2 Delete Command Deleting a selection destroys objects (whether or not they are open) just as it destroys native data. Servers of deleted objects close silently. The client should request an update from the server before deletion so that the user can get the latest updates in the case of an undo.

9.3.5.3.3 Close (Exit) Command The case of closing a client containing an open object was discussed in section 9.3.5.1.3.

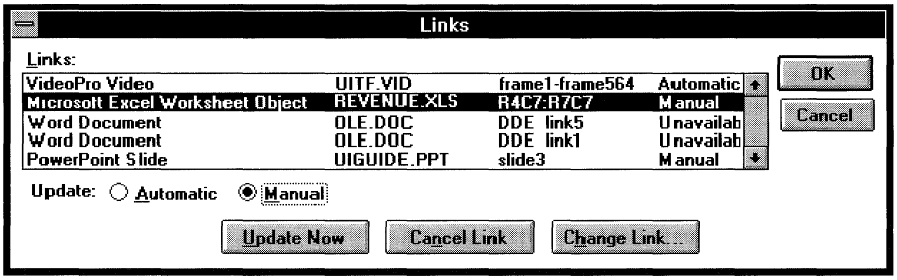
9.3.5.3.4 Cut and Copy Commands When the user chooses the Cut or Copy command, open objects will silently update and the updated versions will be carried to the clipboard. Copying open objects leave their servers open and connected to the original object; cutting open objects close their servers.

9.3.5.3.5 Paste Command (Dropping on, Inserting over, Typing over, etc.)

Pasting over objects (whether or not they are open) replaces them just as it replaces native data. Pasting over open objects also closes their servers. Before closing the server, the client should request an update to prevent data loss in the case that the user wants to undo the paste operation.

9.4 Links and Link Dialogs

Links are "displayed" references to data stored in external documents (or sometimes to data stored within the same document), as illustrated in Figure 9.1. Because linking relies on a dialog with source documents, an interface for maintaining and updating such links is necessary. Figure 9.17 shows the Links dialog, which allows users to change the type of updating (automatic or manual) for links, update linked objects, cancel links, and repair broken links.



Links:

VideoPro Video	UITF.VID	frame1-frame564	Automatic
Microsoft Excel Worksheet Object	REVENUE.XLS	R4C7:R7C7	Manual
Word Document	OLE.DOC	DDE link5	Unavailab
Word Document	OLE.DOC	DDE link1	Unavailab
PowerPoint Slide	UIGUIDE.PPT	slide3	Manual

Update: Automatic [✓ Manual]

Figure 9.17 Links Dialog

When the Links dialog first opens, the Links list shows each link contained in the document. Links contained in the current selection in the document are initially selected in the list. (Embedded objects are not shown in the list.) The list is an extended-selection list; the user can select one or several links by using SHIFT+click for range selection and CTRL+click for disjoint selection, as described in Chapter 3, section 3.1.2.1.

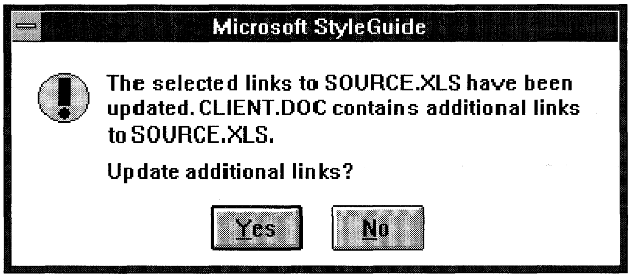
9.4.1 Update Option Buttons

Below the Links list, the Links dialog contains two Update option buttons: Automatic and Manual. When the dialog opens, these buttons reflect whether the currently selected links are automatically updated whenever the linked file changes, or whether they must be manually updated. If the selected links have different update rules, neither button is selected. In this case, choosing one of the buttons changes all the selected links to have the corresponding update behavior.

9.4.2 Link Command Buttons

At the bottom of the Links dialog, three push buttons allow the user to update, cancel, and change the selected links.

- Update Now updates all links selected in the Links list. In other words, the presentations of the linked objects in the client are updated to reflect the current data in the linked files. Suppose that the selected links are all associated with a file called SOURCE.DOC. It is possible that the client contains other, currently unselected links associated with the same file. In this case, after the selected links are updated, the message dialog shown in Figure 9.18 is displayed. If the links selected originally are associated with several files and if the document contains other unselected links to those files, the message dialog is displayed once for each file.



The selected links to SOURCE.XLS have been updated. CLIENT.DOC contains additional links to SOURCE.XLS.

Update additional links?

Figure 9.18 Message for Updating Additional Links to the Same File

- Cancel Link permanently breaks the link between the client and the server. The linked object in the container document is changed to a picture that can no longer be updated or edited with the standard OLE techniques. The picture can still be edited with the older Cut, Copy, and Paste techniques, but it is unlikely that the picture will retain all the data present in the original linked object. When the user presses the Cancel Link button, the entry for the link disappears from the Links list in the Links dialog. Offering the option of changing an embedded object into a picture should be part of an application's own controls (like a menu or a button).
- The Change Link button is dimmed if the selection in the Links list includes multiple links that are not all linked to the same file. Otherwise, the button is active and leads to a dialog exactly like the File Open dialog, except that the title is Change Link (see Figure 9.19). The standard File Open dialog is described in Chapter 8, section 8.1.1.

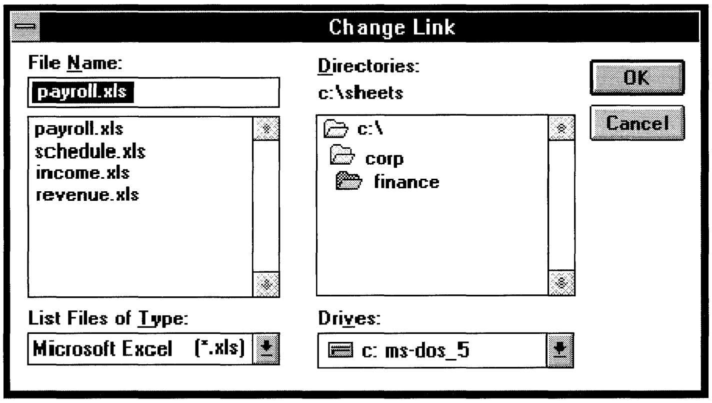
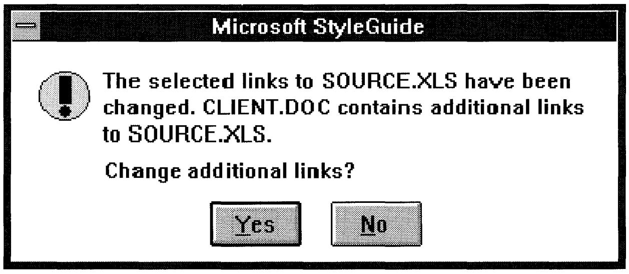


Figure 9.19 Change Link Dialog Box

The Change Link dialog allows the user to change the file to which a link refers. For example, if a linked file is renamed or moved to a new location, this dialog lets the user reconnect the link in the container document, using the new name or the new location of the linked file.

When the user chooses a file and presses OK in the Change Link dialog, the links that were selected in the Links list are disconnected from their previous file and connected to the newly chosen file. It is possible that the previous file was also associated with other, currently unselected links in the container document. In this case, after the selected links are changed, the message dialog shown in Figure 9.20 is displayed. This message is analogous to the one shown in Figure 9.17.



The selected links to SOURCE.XLS have been changed. CLIENT.DOC contains additional links to SOURCE.XLS.

Change additional links?

Figure 9.20 Message for Changing Additional Links to the Same File

9.4.3 Dialog Control Buttons

- The OK button confirms the changes that the user made in the dialog and closes the dialog.
- The Cancel button discards all the changes that the user made and closes the dialog.

9.4.4 Link Status Entries

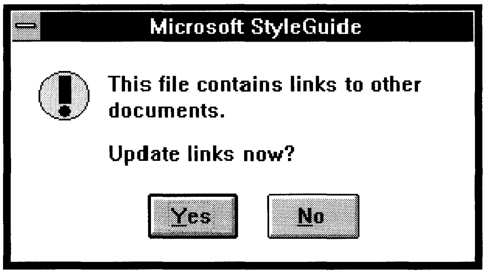
The link status entries in the Links list box consist of four parts: the human-readable form of the class name, the source file for the link, the item name for the link, and the status of the link.

- The human-readable form of the class name is the string that is registered in the registration database for that object class.
- The source file for the link contains the full pathname for the link. If the full path is too long to be displayed, the entry should be truncated.
- The item name contains the server-specific item name for the object.
- The status of the link can be Automatic, Manual, or Unavailable:
  - An *automatic* link is updated automatically when it is changed in the server or when the linked object is loaded and the server is open with that object. It is also updated when the linked object is loaded and the user responds "Yes" to the Link Update Message Dialog (Figure 9.21).
  - A *manual* link is a link that the user must explicitly update through the Links dialog. The update takes place when the link is selected and the user presses the Update Link button in the Links dialog.
  - The link receives an *unavailable* status when the attempt to update the link (upon loading the file or requesting OleUpdate) fails.

9.4.5 Other Dialogs for Link Updating

When the user opens a file containing links (manual or automatic[10]), a message dialog is displayed to ask the user whether to update the links (see Figure 9.21). If the user presses the Yes button, the application updates all of the links.

[10]Automatic links are not automatically updated unless the server is open and the source of the link is loaded, because this could make simple viewing of the file cumbersome.

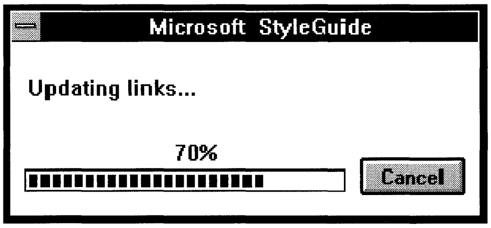

Microsoft StyleGuide

This file contains links to other documents.

Update links now?

Figure 9.21 Link Update Message Dialog

The progress indicator shown in Figure 9.22 may be displayed while the links are being updated. The Cancel button interrupts the update process and cancels all updating that has already been carried out.

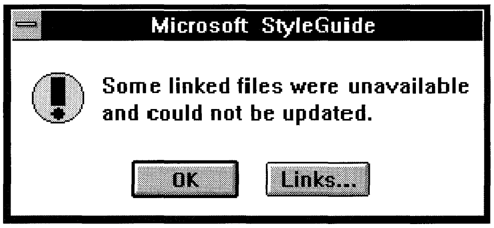

Microsoft StyleGuide

Updating links...

70%

Figure 9.22 Progress Indicator for Link Updating

If some of the linked files are unavailable, the warning dialog shown in Figure 9.23 is displayed. This dialog contains two buttons, OK and Links. The OK button closes the dialog without updating the links. The Links button displays the Links dialog (see Figure 9.17) with all the links listed. Unavailable linked files are marked with the word "Unavailable" in the third column of the list. The user can attempt to locate the unavailable files by using the Change Link dialog (see Figure 9.19), which is available from the Change Link command button in the Links dialog.


Microsoft StyleGuide

Some linked files were unavailable and could not be updated.

Figure 9.23 Warning Message for Unavailable Links

9.5 Status Line Message Recommendations

If a client application uses the status line to elaborate on menu commands, the messages below can be used for OLE commands.

Table 9.2 Status Line Messages

Menu Commands	Status Line Message
File Menu	
Update	Update changes in <*Container-Document*>
Save Copy As	Save a copy of <*Descriptive-Class-Name*> in a separate file
Exit & Return to <*Container-Document*>	Exit <*Server* > and return to <*Container-Document*>
Edit Menu	
Paste	Inserts clipboard contents as <*Default-Data-Type*>[11]
Paste Special	Inserts clipboard contents as a linked object, embedded object, or other format
Paste Link	Inserts a link to <*Descriptive-Class-Name*> Object from <*Source-Document*>
Insert Object	Inserts a new embedded object
<*Verb*>[12]<*Descriptive-Class-Name*> Object	None
<*Descriptive-Class-Name*> Object	Apply the following commands to <*Descriptive-Class-Name*> Object
<*Descriptive-Class-Name*> Object <*Verb*>	None
Links	Allows links to be viewed, updated, opened, or canceled
Options (Preferences) Menu	
Show Objects	Displays the borders around objects (toggle)
Mouse Interface	
When an object is selected	Double-click to <*Primary-Verb*> <*Descriptive-Class-Name*> Object

[11] <*Default-Data-Type*> is identical to the initially highlighted value in the Paste Special Data Type list. This status line message indicates the data format used to paste clipboard contents.

[12] If no verb in the registration database is specified, "Activate" should be used as the default.

The Pen Interface

Chapter 10

Pen-based computers let users provide input by tapping or writing on the surface of the computer screen with a special pen. The pen provides a natural and intuitive way of interacting with the computer. The number of pen-capable computers and applications is expected to grow rapidly in the next few years. Any application that makes good use of menus and graphical controls has a head start on a good pen interface.

10.1 Pen Input

The pen can be used for both pointing and writing, depending on where it is placed.

10.1.1 Pointing

When the pen is moved over menus or controls, it becomes a pointing device and lets the user select menu commands, choose buttons, or perform other mouse-like operations. Tapping[1] the pen once on the screen is equivalent to clicking mouse button 1 once. A double-tap is equivalent to a double-click. If the user holds down the barrel button of the pen while tapping, the tap is equivalent to a click with mouse button 2.

10.1.2 Writing

When the pen is over an edit control or a text area, it becomes a writing tool and the pointer changes into a pen shape. When the tip of the pen touches the screen, the pen starts "inking"—that is, tracing lines on the screen. The user can draw shapes, characters, and other patterns; these can remain on the screen exactly as drawn or can be recognized, interpreted, and redisplayed.

10.1.3 Dragging

The pen retains the power of the mouse even in contexts where it normally functions as a writing or a drawing tool. If the user presses the pen tip down on a text area and holds it steady for a certain period before moving it, the subsequent pen movements are interpreted as mouse movements. Thus, the pen can be used for drag selection of text, outline selection of graphical objects, object movement, and other mouselike operations.

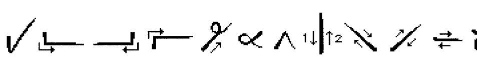
[1] To tap, the user presses the pen tip on the screen and releases it without moving the pen.

Table 10.1 summarizes the principal pen techniques described above and gives a few examples of their use.

Table 10.1 Pen Techniques

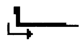
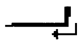
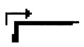
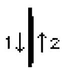
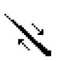
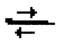
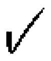
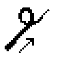
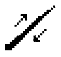
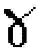
Technique	Examples of Use
Tap	Select object or menu command; set insertion point in text; push command button.
Double-tap	Open object; select word.
Drag	Move object (for example, to move a window, drag its title bar); resize object (for example, to resize a window, drag its border; to resize a graphical object, drag its resize handles).
Press/hold/drag	Select text from pen-down location to pen-up location; perform drag operations (for example, object movement or marquee selection) in contexts where the pen normally functions as a writing or drawing tool.
Write/draw	Enter text or graphics; execute gestural commands (see Table 10.2).

10.1.4 Gestures



When the pen is used for writing, certain ink patterns are interpreted as "gestures" —special symbols that issue a command, such as deleting text, or produce a non-printing text character, such as a carriage return or a TAB. For example, the "^" shape is equivalent to the Paste command. After a gesture is interpreted, its ink is removed from the display. Table 10.2 lists the 12 standard pen gestures.

Table 10.2 Pen Gestures

Name	Glyph	Hot Spot	Acts Where?	Granularity	Effect	Equivalent**	Comments
Space		First point	Positional	Insertion point	Insert space where drawn.	Space character	Don't replace selection. (For non-positional operation, which replaces selection, use on-screen key-board, menu, or circled letters.)
New Line					Insert new line where drawn.	New line character	
Tab					Insert tab where drawn.	Tab character	
Paste		Top			Paste where drawn.	SHIFT+INS***	
Extend Selection		Center			Extend selection from anchor point to gesture.	SHIFT+click	Start downward to avoid con-fusion with *l*.
Backspace		Lowest point		Character	Delete character under gesture.	BACKSPACE	Start in either direction.
Delete Words		Left, right		Word	Delete words under gesture.	Double-click +DEL****	
Edit Text		Inside center of lower "v"	Act on selection if one exists; otherwise positional* (always positional in boxed edit controls).	Selection or word	Put selection (if any) or word into Edit Text dialog.	—	
Cut		First point			Cut selection if any, otherwise cut word under gesture.*	SHIFT+DEL***	
Copy		Center of bounding box			Copy selection if any, otherwise copy word under gesture.*	CTRL+INS***	Feedback: copy pointer flashes.
Delete		Lowest point		Selection or character	Delete selection if any, otherwise delete character under gesture.*	DEL	Start in either direction.
Undo		None	Non-positional	Operation	Undo last operation.	ALT+ BACKSPACE***	

* Cut, Copy, and Delete never act positionally in Windows applications that were not designed for the pen. If there is no selection, they act at the insertion point. For Cut and Copy, this usually means that no operation is performed.

** This column specifies keyboard and mouse equivalents used in Windows applications that were not designed specifically for the pen.

*** This column is not intended to represent the recommended shortcuts for Cut/Copy/Paste, but to show the mapping that Windows for Pen Computing uses for compatibility with Windows version 3.0 applications.

**** Note that double-click+DEL deletes only one word, whereas the Delete Words gesture is capable of deleting multiple words.

10.1.4.1 Positionality of Gestures

Most gestures act positionally. They contain a "hot spot" that can be used to determine where the gesture should act. For example, the hot spot of the Paste gesture is at the top of the "^" shape. When the user draws the Paste gesture, the pasted data is inserted at the location of the hot spot.

Undo is the only gesture that never acts positionally. Regardless of where the user draws it, this gesture cancels the last operation. Positional, object-specific Undo functionality is a possible extension for future interfaces.

10.1.4.2 Basic and Advanced Gestures

To reduce the amount that new users must learn, the gestures are divided into two sets: basic and advanced.

The basic gestures are Edit Text, Backspace, Space, New Line, Cut, and Undo. Cut is included instead of Delete for two reasons:

- In handwriting edit controls and in pen-centric applications, Cut can be used to delete one word at a time.
- Delete is only necessary when the user doesn't want to destroy the contents of the clipboard. This is an advanced situation that beginning users don't need to know about.

Undo is included in the basic set, despite its availability on the Edit menu of many applications. Rapid gestural access to Undo functionality makes the interface seem more forgiving and approachable.

The advanced gestures are Copy, Paste, Delete, Delete Words, Extend Selection, and Tab. Copy and Paste are included in the advanced set instead of the basic set for two reasons:

- Copy and Paste commands are available on the Edit menu of most applications.
- Users can remember the basic set more easily if it is limited to fewer commands.

Pen applications should support both the basic and advanced gestures. However, documentation should focus on the basic set, and pen application designers should ensure that their applications can be used productively with the basic set alone.

10.1.4.3 Circled-Letter Gestures

The pen interface also allows users to define gestures consisting of circled letters that can be mapped to specific functions or key equivalents. Four circled-letter gestures are assigned default meanings: C (Copy), P (Paste), U (Undo), and X (Cut). These gestures are non-positional in handwriting edit (hedit) controls and in Windows applications that were not designed specifically for the pen. In boxed edit (bedit) controls, the circled-letter gestures are positional.

10.1.4.4 Advantages of Gestures

Pointing, drawing, and text input are important functions already supported by current mouse-based and keyboard-based applications. A unique virtue of the pen is its ability to specify a selection (the "noun" for an operation) as well as an action (the "verb" for that operation) directly through a gesture. Gestures eliminate the "select object then select operation from a menu" interface enforced by the mouse and by the keyboard. With the pen, users make a single gesture at the object. The application then determines which data to change and which operation to perform.

The rapidity and naturalness of gestural commands are among the key advantages of the pen interface. However, applications should not rely on gestures as the only way to perform commands, because gestures are hidden from the user. As a supplement, applications should also provide menu commands or buttons to carry out the functions performed by the gestures. Applications can put a bitmap of the gesture next to the corresponding menu command. This bitmap helps the users learn gestures; it replaces the keyboard shortcut text that appears on standard Windows menus.

10.2 Designing Pen Interfaces

These guidelines ensure that pen applications take advantage of the strengths of the pen while avoiding its weaknesses.

10.2.1 Simplicity and Directness

10.2.1.1 Keep The Interface Simple

Pen applications are often used by unsophisticated users on machines with small displays and limited storage space, so simplicity is especially important for pen interfaces. Pen applications should forego kitchen-sink interfaces in favor of streamlined simplicity: short menus, small dialog boxes, uncrowded control bars, few overlapping windows, and simple metaphors. Many pen applications can dispense with some of the standard interface elements altogether. For example, the Microsoft Windows for Pen Computing tutorial uses a simple interface with no menus or dialog boxes.

10.2.1.2 Exploit Direct Manipulation

Direct manipulation is particularly useful in pen-based systems for two reasons:

- First, dragging objects with the pen requires less coordination than dragging them with the mouse, because the user does not have to press a button while manipulating the pen.
- Second, manipulating objects with the pen is even more direct than manipulating objects with the mouse. The mouse is located on the user's desk, separated from the pointer on the screen, whereas the pen points directly to the object on the screen. To drag objects with the pen, the user presses the pen tip onto the screen over the object and then moves the pen along the surface of the screen without lifting it.

Pen applications should provide adequate hot zones around small areas that will be targets for pen taps or drag-and-drops. In general, the minimum area of the target plus the hot zone should be at least five pixels. Hot zones are especially important for pen systems because the thickness of the display surface can cause distortion and make precise positioning difficult.

10.2.1.3 Take Advantage of Positionality

One of the great strengths of the pen is its ability to specify a spatial position as it draws a gesture, a character, or a graphical object. Pen applications can take advantage of this feature to process pen input intelligently. For example:

- A gesture typically affects the object underneath it; however, if the user does not draw the gesture directly on any object, the application can apply the gesture to the nearest appropriate object.
- In free-form input, a character can be inserted where it was written; in formatted text, it can be snapped into alignment with the nearest neighboring character.
- In flow charts or organization charts, a square drawn in empty space can be left at the drawing location, whereas a square drawn near an arrow can be moved to abut the arrow (see Figure 10.1).
- In an application that supports both shape and character recognition, an “o” shape can be interpreted as a character or as a circle, depending on whether it was closer to other characters or to other graphical objects.

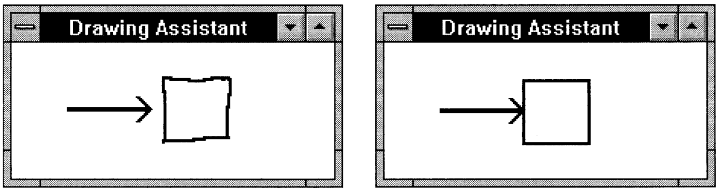


Figure 10.1 Using Proximity to Determine Placement After Recognition

Some pen hardware can detect the proximity of the pen to the display surface. Applications can use this information to provide feedback about the operations that will be available if the user taps or draws on the display. One way to provide this feedback is with pointer changes, as described in the next section.

10.2.1.4 Use Pointers to Increase Accuracy and to Provide Feedback

Because the pen (unlike the mouse) points directly at the screen, graphical on-screen pointers may seem superfluous; however, they do have an important role to play. Usability tests show that pointers help pen users select small targets faster. Moreover, changes from one pointer to another provide useful feedback about the actions supported by the object under the pen. For example, when the pen moves over a resizable border, the pointer can change from a pen (indicating that writing is possible) to a resize pointer (indicating that the border can be dragged to resize the object). Pen applications should use this type of feedback whenever possible to help users understand the actions that are currently enabled by the application. For a list of suggested pointers, see Chapter 3, section 3.6.1.1.1.

In principle, pen applications could dispense with pointers altogether and instead provide target feedback by changing the object under the pen. However, this approach has some disadvantages. First, it can easily lead to a larger number of distracting display changes than are required with pointers. For example, when the pen is moved over a tool bar, the pointer approach only requires one small change (from pen to arrow pointer). The object-change approach, however, would probably require either one, much larger, change (such as highlighting the whole tool bar) or many small changes (such as making each button flash as the pen passes over it). Another potential disadvantage of the object-change approach is its lack of real-world intuitiveness; real objects typically don't change when we approach them. The pointer approach is slightly more realistic in that respect; our hands (analogous to pointers) can undergo changes (for example, sensations of heat and cold) when moved near objects. For these reasons, the object-change approach cannot be recommended without further design work and usability testing.

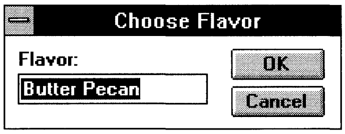
10.2.2 Recognition Issues

10.2.2.1 Minimize the Need For Writing and Recognition

Because handwriting recognition takes extra time and may occasionally result in errors that must be corrected by the user, pen applications should minimize the need for the user to write text that must be recognized. Two general rules for minimizing writing and recognition are (1) avoid text boxes and (2) preserve ink where appropriate.

10.2.2.1.1 Avoid Text Boxes Text boxes can sometimes be replaced by lists, combo boxes, or spin boxes; these controls present entries that the user can select without typing or writing. If your application has some knowledge of possible values for the text box, you can use:

- Standard or drop-down lists, if the entire set of possible input values is known (for example, list of available macros or templates or see Figure 10.2). For additional information on lists, see Chapter 6, section 6.3.
- Standard or drop-down combo boxes, if likely field values are known but others are also possible (for example, font names). For additional information, see Chapter 6, section 6.4.1.
- Spin boxes, if likely values are known but others are possible, if the values are intrinsically ordered, and if the user typically only wants to make small increments or decrements (for example, margin settings or month names). For additional information, see Chapter 6, section 6.4.2.



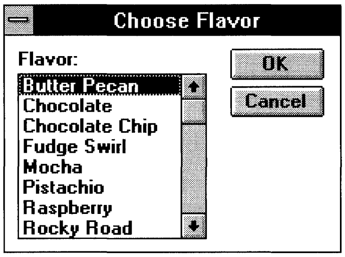


Figure 10.2 Replace a Text Box (Left) with a List (Right) Whenever Possible

10.2.2.1.2 Preserve Unrecognized Ink When Appropriate In situations where writing cannot be avoided, it is sometimes appropriate to avoid recognition by preserving the ink exactly as written. For example, in applications involving personal notes, annotations, or electronic mail, uninterpreted handwriting is a simple and natural means of expression (see Figure 10.3). Handwriting edit controls and boxed edit controls both support the ability to accept and preserve ink input without recognizing it. Inking capabilities can be added to existing window classes without much effort.

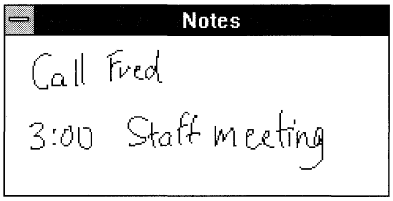


Figure 10.3 Ink Need Not be Recognized to Be Useful

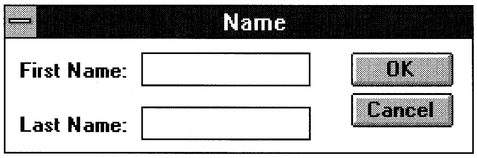
Many of the convenient features of traditional word processors—alignment, cut/paste, bold versus plain styles, and so on—are equally valuable for uninterpreted ink. Ink also opens up many new possibilities such as free-form sketches, annotations, ink erasure, selective recognition for indexing, and more.

10.2.2.2 Aid Recognition By Providing Input Areas for Neat Handwriting

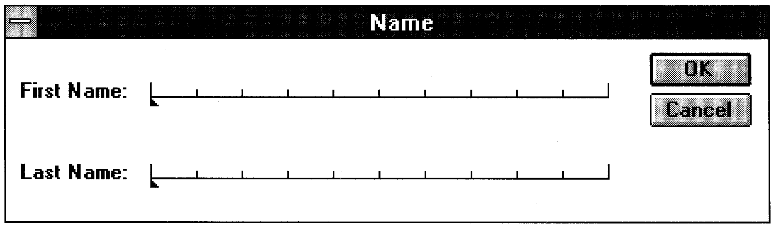
When the need for recognition cannot be avoided, applications can improve the accuracy of recognition by providing input areas that encourage neat, well-segmented handwriting.

10.2.2.2.1 Use Boxed Edit Controls Use boxed edit controls (see Figure 10.4) to get handwritten input from the user whenever possible. Users write more neatly when constrained. Moreover, boxed edit controls provide excellent segmentation and baseline information for the handwriting recognizer. Boxed edit controls function best when the input length and type are known, for example, in a social security number, a phone number, a first name, and so on. Boxed input is less than optimal when the amount of user input cannot be predicted or restrained.

10.2.2.2.2 Provide Large Areas for Handwriting Input Applications can improve recognition in boxed edit controls and in handwriting edit controls by providing plenty of space (see Figure 10.4). In general, larger handwriting is recognized more accurately. People write more neatly and deliberately in large spaces as opposed to small spaces. Ample space also makes selection, correction, and other modifications easier.



Not recommended



Recommended

Figure 10.4 Replacing Small Text Boxes with Spacious Boxed Edit Controls

10.2.2.3 Use Contextual Constraints to Improve Recognition

Applications can improve recognition accuracy by telling the recognizer what type of data to expect. For example, an address book application can constrain fields, such as phone number and zip code, to contain only numbers. Applications can also constrain recognition by supplying lists of acceptable values (for example, currently registered license plates). In addition to providing strictly defined constraints to the recognizer, applications can apply their own, more flexible, heuristic constraints after the recognizer returns a set of possible results. For example, suppose that after the user writes text in a spreadsheet cell, the recognizer returns "25", "2s", "z5", and "sz" as possibilities. Because the spreadsheet cells can contain text as well as numbers, all four choices are potentially valid, but the spreadsheet can select "25" as the most likely choice.

10.2.2.4 Use Recognition for Graphical Input

Recognition is not limited to writing. Special recognizers (such as the shape recognizer that is included with Windows for Pen Computing) can convert drawn graphics to application-specific or context-specific input. Examples include a drawing package that can snap a rough circle or square to a perfect one (see Figure 10.5) and a CAD/CAM application that can recognize the symbols specific to that industry.

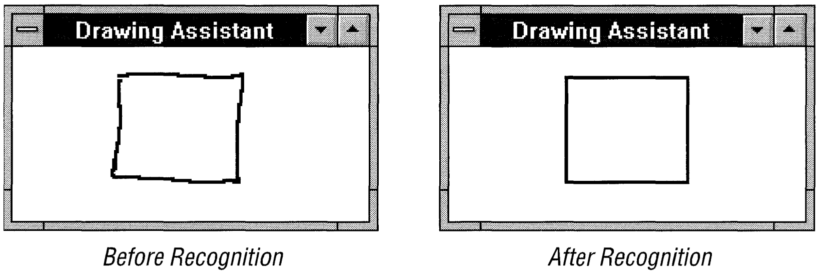


Figure 10.5 Recognizing Graphical Objects

10.2.3 Hardware Constraints

Pen application designers should take the constraints imposed by pen-capable hardware into consideration.

10.2.3.1 Economize on Storage

Many early pen machines have limited RAM and disk space. As a result, simple, streamlined applications are preferable to large, feature-laden applications. Simple applications not only require less storage but also tend to be easier to use, and ease of use is a key factor for the pen market.

10.2.3.2 Provide Configurable Layouts

A pen application interface should be adaptable to the variety of screen sizes, shapes, and orientations that will be available on pen machines from different manufacturers. Most computer screens adopt a horizontal (landscape) orientation. Clipboard-style pen computers can also be used in a vertical (portrait) orientation. This is a result of how we traditionally hold clipboards. The impact on applications is that the display, which is 640x480 when held horizontally, suddenly becomes 480x640 when held vertically. Pen applications should provide layouts that take both orientations into account. Dialog boxes and control bars must also be designed to fit in both orientations.

Pen applications should also take different screen sizes into account. Hardware manufacturers will soon be providing a wide range of display sizes for pen machines, including some displays as small as 320x200. Pen applications can prepare for this market—while also increasing their simplicity and usability—by streamlining their interfaces (for example, by keeping menus short and dialogs small), and by making interface elements scalable or resizable whenever possible.

10.2.3.3 Don't Rely on Color ??

For at least the next year or so, clipboard computers will not provide color displays; instead, most will provide 16 shades of gray. For this reason, pen applications should not rely on color to distinguish interface elements or to provide other essential information.

10.2.3.4 Don't Assume An Auxiliary Keyboard ??

Pen-based clipboard computers will often be used without a keyboard. Accordingly, pen application designers should ensure that the application can be driven entirely from its menus, dialog boxes, and control bars. The most important actions should be directly available through on-screen buttons or easily remembered gestures.

10.2.3.5 Minimize Setup and Startup Time ⌛

Because pen computers are frequently used by people on the go, pen applications should minimize setup and startup time. One helpful technique is to restore the interface settings that were in effect the last time the application was used. For example, applications can automatically restore the previous window location and layout. In some applications, it may also be appropriate to reload the most recently used data and to scroll the window to the most recently viewed portion of the data.

10.2.3.6 Conserve Power

Although power management is not strictly a user interface issue, pen application designers should be aware of the power implications of interface decisions and minimize power consumption whenever possible. For example, features such as automatic background repagination substantially increase power consumption and thus reduce the operating time of a portable pen-based computer. In general, when waiting for user input, pen applications should simply wait in the Windows message loop rather than trying to accomplish numerous background tasks.

Miscellaneous Topics

Chapter 11

11.1 Loading and Initialization

The following list shows the suggested sequence of steps for an application after it is invoked:

1. Display the application window.
2. Display the startup message.

In general, additional initialization tasks may be performed any time after the application window is displayed—either before or after the startup message. These tasks typically have no visible results. If they take a long time to complete, they should be divided into two groups. The first group should be performed before the startup message is displayed; the second group should be performed after the startup message. The appearance of the startup message between the two sets of tasks serves as a progress indicator that makes the lengthy initialization seem shorter.

11.1.1 Memory Check

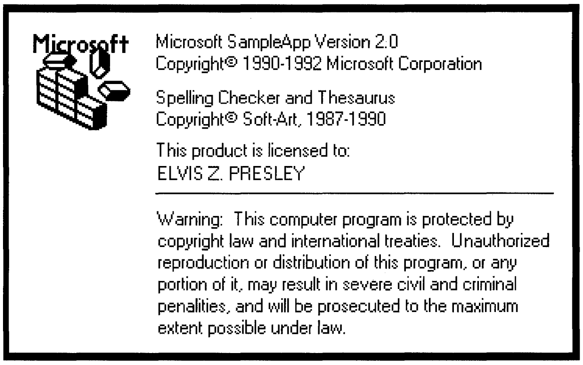
Generally, if insufficient memory is available when the application is invoked, the system posts a message informing the user. However, once the application code is executed, it becomes the application's responsibility to inform the user if insufficient memory is available for performing specific operations.

11.1.2 Display of Application Window

The application should display the application window as soon as possible, rather than leaving the screen blank until the application is fully started and ready for user interaction.

11.1.3 Display of Startup Message

After the application window has been displayed, the application should display a modal dialog window that includes copyright, version, and user identification information. (The dialog window need not include a title bar.) Figure 11.1 shows the standard format for this information. The startup message window may also contain an icon or other graphic that identifies the application; typically this will be the same icon that is used in the About *<Application-Name>* dialog box (see Chapter 8, section 8.6). Once the application is loaded and ready for user input, the startup window should be automatically removed.



Microsoft SampleApp Version 2.0
Copyright© 1990-1992 Microsoft Corporation

Spelling Checker and Thesaurus
Copyright© Soft-Art, 1987-1990

This product is licensed to:
ELVIS Z. PRESLEY

Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalities, and will be prosecuted to the maximum extent possible under law.

Figure 11.1 Startup Screen

11.2 User Levels and Customization

To accommodate user preferences and skill levels, applications may provide means for the user to customize the interface. When the user quits an application, the current customizations should be saved so that they can be set up in the same way the next time the user invokes the application.

Customization methods differ widely in how much flexibility they offer the user. Some only allow the user to choose from a limited set of predefined possibilities, whereas others allow the user to rearrange parts of the interface in a virtually unlimited number of ways, or even to create completely new commands.

11.2.1 Unfolding Dialog Boxes

Large, complex dialog boxes can intimidate new or inexperienced users. To provide simple, easy-to-understand dialogs for these users while retaining advanced functionality for experienced users, applications can implement dialog boxes that have two sizes—small and expanded. Whenever the dialog is invoked, it should initially appear in the small size, which includes the basic controls necessary to provide the most common functions in the dialog. The small size also includes an unfold button (typically labeled Options >>), which the user can press to expand the dialog. The expanded form of the box contains both basic and advanced controls. For more information on unfolding dialogs, see Chapter 7, section 7.1.2.

11.2.2 Customization Dialogs

Applications may also allow users to customize the interface by providing one or more customization dialogs. For example, such dialogs may offer options for displaying or hiding interface elements (such as special characters, gridlines, or horizontal scroll bars) or for changing interface behavior (such as whether typing replaces an existing selection). Customization dialogs may also let the user change the location of menu commands or even add new commands (for example, commands created with macros).

11.2.3 Considerations for Disabled Users

Some customization methods provided by the application (for example, menu reconfiguration) or the system (for example, mouse, keyboard, and volume adjustments) allow disabled users to adjust the interface to suit their needs. Applications can further accommodate disabled users by observing the following guidelines:

- Use multiple perceptual input channels.
  - Avoid using only audio cues, such as beeps, for any situation that absolutely requires attracting the user's attention.
  - Don't rely on color alone to provide essential information.
- Do not require rapid responses.
  - Avoid time-out situations that require a quick response, except in games. If time-outs are used at all, either make the time-out period long (at least one minute) or permit the user to run the application in a "slow" mode.
- Avoid rapid flashing on the screen.
  - Avoid using high rates of flashing for any interface elements or data items. Rapid flashing can cause seizures in some users.

11.3 Help

Applications can facilitate users' tasks by providing small amounts of helpful information automatically while the user is working, either through the message bar (see Chapter 4, section 4.2.6) or through message dialogs (see Chapter 7, section 7.1.4). However, the scarcity of screen space and the need to keep the application window uncluttered limit the amount of helpful text that can be displayed during the user's normal interaction with the application. Applications should therefore provide a way for users to access additional help whenever they need it.

11.3.1 Access to Help

Users may access help from the Help menu, the Help key (F1), or Help mode (SHIFT+F1, followed by a mouse click on the element for which help is desired). For information on the Help menu, see Chapter 5, section 5.4.4.

11.3.1.1 Help Key

When the user presses the Help key (F1), the Help application window appears. Whenever possible, the information initially displayed in the Help window should be context-sensitive; that is, it should reflect the currently active interface element. For example, if a menu command is selected, the Help window should provide information about the command; if a message dialog is being displayed, the Help window should provide additional information about the message. If the application cannot support context-sensitive help, the Help window should initially display a list of possible help topics when F1 is pressed.

11.3.1.2 Help Mode

As an optional extension to Help, applications may implement a Help mode. To enter Help mode, the user presses SHIFT+F1, which changes the mouse pointer to the Help pointer (the standard selection pointer joined to a question mark; see Chapter 3, Table 3.9). To cancel Help mode, the user presses ESC. In Help mode, the user positions the pointer over the interface element for which help is desired. When the user clicks mouse button 1, the Help window appears with information appropriate to that element. As a rule of thumb, applications should always provide the most specific help possible for the context, to the extent that the context can be determined.

Keyboard access to menu items is also available in Help mode. Choosing a menu item with the keyboard in Help mode displays Help information for the menu item instead of initiating the item. If a dialog box is open when the user initiates Help mode, the Help window and appropriate information should appear without further user intervention; that is, the user should not have to close the dialog first.

11.3.1.3 Help in Dialogs and Messages

When using dialog boxes, users can always obtain Help by pressing F1. For more visible access to Help, applications may also provide a Help command button in the dialog box. The "H" in the button label should be underlined to indicate that it is a mnemonic access character. The keyboard user can press the Help button with F1 or ALT+H. Although the ALT+H method is somewhat redundant, it provides two advantages over the F1 method. First, the H mnemonic is more visible. Second, it is easier to remember because H is the first letter in Help and because ALT+H is also used to access the Help menu outside dialog boxes.

Help buttons in message dialogs are optional but highly recommended, especially for warning and critical messages. They provide additional information or suggestions about the problem described by the message.

11.4 International Concerns

To compete successfully in international markets, applications should ensure that their interfaces can be easily adapted to accommodate differences in language, culture, and hardware.

11.4.1 Interface Text

The process of internationalizing an interface starts with translating the interface text. Interface text includes title bar titles, menu names, menu items, control labels, list items, and messages. For easy localization, such text should be stored as resources in the resource file rather than being included in the source code for the application.

Translation of interface text from English to other languages typically increases the length of the text by 30% or more. In some extreme cases, the character count can increase by more than 100%; for example, the word "restore" becomes "zurückspeichern" or "wiederherstellen" in German. Accordingly, if the amount of space for displaying text is strictly limited, as in the status bar, the length of the English interface text should be limited to approximately one-half the available space. (Mode indicators in the status bar, such as NUM, may use all the available space because localized versions try to use the same number of letters as the U.S. version.) In contexts that allow more flexibility, such as dialog boxes, the interface design should allow for text expansion of at least 30%; message text in message dialogs, however, should allow for expansion by about 100%. Applications should never rely on the position of text in a menu, dialog box, or window because translation might require that the text be moved.

11.4.2 Hardware

Outside the U.S., display hardware is not dominated by EGA or VGA standards. Accordingly, dialog boxes and other interface elements should be designed to maintain their aesthetic appeal on various resolutions and screen aspect ratios.

International keyboards may also differ from those in the U.S. installed base. In particular:

- ALT+key combinations should be chosen carefully because some international keyboards use them to enter certain characters.
- Function key accelerators are easier to localize than modifier+letter accelerators.

- Shortcuts that use punctuation marks should be chosen carefully because some punctuation marks (for example, braces and brackets) are frequently not found on international keyboards or are only available in combination with the ALT key.
- All international applications should support multiple code pages and sorting tables to allow for the use of different extended character sets. For sorting and case conversion, applications should use system-supplied rather than application-specific routines whenever possible.

11.4.3 Formats

Different countries often use substantially different formats for dates, time, money, measurements, and telephone numbers. As a result, international applications should allow these formats to be changed easily. The setup program for the application should initialize the formats to the default values obtained from the system Control Panel. The application itself may also allow the formats to be changed whenever necessary during normal use of the application. Such changes may be saved on an application-specific or document-specific basis, but should not affect the system defaults.

Table 11.1 lists the most common format categories.

Table 11.1 Formats for International Applications

Category	Format Considerations
Date	Order, separator, and long/short formats
Time	Separator and cycle (12-hour vs. 24-hour)
Physical quantity	Metric vs. English measurement system
Currency	Symbol and format (for example, trailing vs. preceding symbol)
Separators	List, decimal, and thousandths separators
Telephone numbers	Separators for area codes and exchanges
Paper sizes	U.S. vs. European paper sizes

Index

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-63

Microsoft®

0592 Part No. 2892