some cases, you can include these properties by implication. For example, requesting the properties of a text selection can also provide access to the properties of the paragraph in which the text selection is included.

Another way to provide access to an object's properties is to create a representation of the object. For example, the properties of a page could be accessed through a graphic or other representation of the page in a special area (for example, the status bar) of the window.

Yet another technique to consider is to include specific property entries on the menu of a related object. For example, the pop-up menu of a text selection could include a menu entry for a paragraph. Or consider using the cascading submenu of the Properties command for individual menu entries, but only if the properties are not easily made accessible otherwise. Adding entries for individual properties can easily end up cluttering a menu.

The Properties command is not the exclusive means of providing access to the properties of an object. For example, folder views display certain properties of objects stored in the file system. In addition, you can use toolbar controls to display properties of selected objects.

Pen-Specific Editing Techniques

A pen is more than just a pointing device. When a standard pen device is installed, the system provides special interfaces and editing techniques.

Editing in Pen-Enabled Controls

If a pen is installed, the system automatically provides a special interface, called the *writing tool*, to make text editing as easy as possible, enhance recognition accuracy, and streamline correction of errors. The writing tool interface, as shown in Figure 5.7, adds a button to your standard text controls. Because this effectively reduces the visible area of the text box, take this into consideration when designing their size.

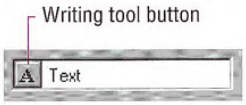


Figure 5.7 A standard text box with writing tool button

Figure 5.8 shows how you can also add writing tool support for any special needs of your software, such as a multiline text box.

Figure 5.8 Adding the writing tool button

When the text box control has the focus, a selection handle appears, as shown in Figure 5.9. The user can drag this handle to make a selection.

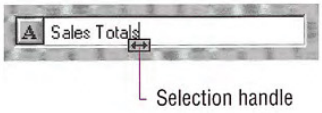


Figure 5.9 Text box displaying a pen selection handle

Tapping the writing tool button with a pen (or clicking it with a mouse) presents a special text editing window, as shown in Figure 5.10. Within this window, the user can write text that is recognized automatically.

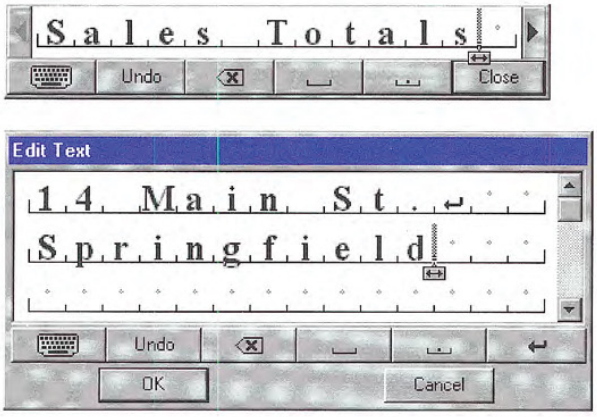


Figure 5.10 Single and multiline writing tool windows

In the writing tool editing window, each character is displayed within a special cell. If the user selects text in the original text field, the writing tool window reflects that selection. The user can reset the selection to an insertion point by tapping between characters. This also displays a selection handle that can be dragged to select multiple characters, as shown in Figure 5.11.

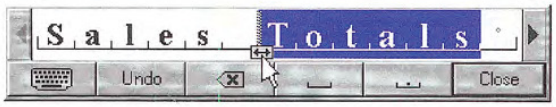


Figure 5.11 Selecting text with the selection handle

The user can select a single character in its cell by tapping, or double-tapping to select a word. When the user taps a single character, an action handle displays a list of alternative characters, as shown in Figure 5.12.

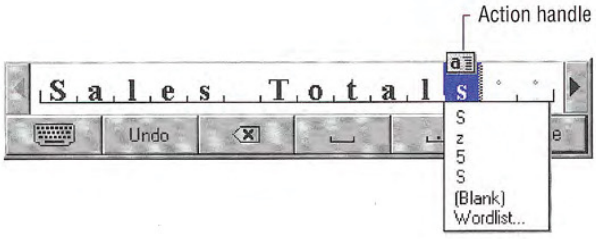


Figure 5.12 An action handle with a list of alternatives

Choosing an alternative replaces the selected character and removes the list. Writing over a character or tapping elsewhere also removes the list. The new character replaces the existing one and resets the selection to an insertion point placed to the left of the new character.

The list also includes an item labeled Wordlist. When the user selects this choice, the word that contains the character becomes selected and a list of alternatives is displayed, as shown in Figure 5.13. This list also appears when the user selects a complete word by double-tapping. Choosing an alternative replaces the selected word.

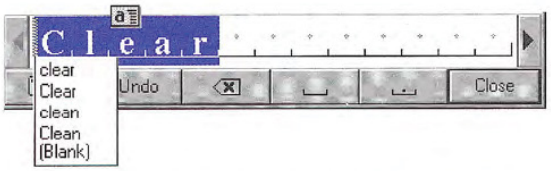


Figure 5.13 Tapping displays a list of alternatives

Whenever a selection exists in the window, an action handle appears; the user can use it to perform other operations on the selected items. For example, using the action handle moves or copies the selection by dragging, or the pop-up menu for the selection can be accessed by tapping on the handle, as shown in Figure 5.14.

For more information about pop-up menus, see Chapter 7, "Menus, Controls, and Toolbars."

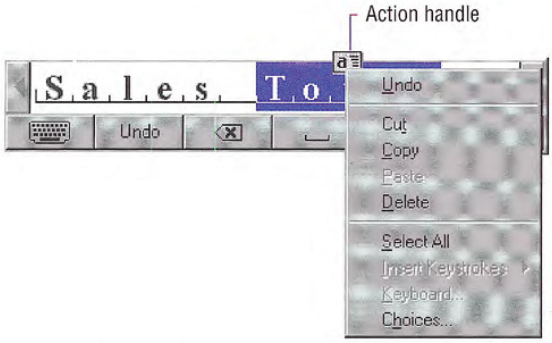


Figure 5.14 Tapping on the handle displays a pop-up menu

The buttons on the writing tool window provide for scrolling the text as well as common functions such as Undo, Backspace, Insert Space, Insert Period, and Close (for closing the text window). A multiline writing tool window includes Insert New Line.

The writing tool window also provides a button for access to an onscreen keyboard as an alternative to entering characters with the pen, as shown in Figure 5.15. The user taps the button with the corresponding keyboard glyph on it and the writing tool onscreen keyboard pop-up window replaces the normal writing tool window.

Figure 5.15 The writing tool onscreen keyboard window

The writing tool "remembers" its previous use — for text input or as an onscreen keyboard — and opens in the appropriate editing window when subsequently used. In addition, note that when the user displays a writing tool window, it gets the input focus, so avoid using the loss of input focus to a field as an indication that the user is finished with that field or that all text editing occurs directly within a text box.

Pen Editing Gestures

The pen, when used as a pointing device, supports editing techniques defined for the mouse. When used as a writing device, the pen supports gestures for editing. Gestures (except for Undo) operate positionally, acting upon the objects on which they are drawn. If the user draws a gesture on an unselected object, it applies to that object, even if a selection exists elsewhere within the same selection area. Any pending selections become unselected. If a user draws a gesture over both selected and unselected objects, however, it applies only to the selected ones. If a gesture is drawn over only one element of the selection, it applies to the entire selection. If the gesture is drawn in empty space (on the background), it applies to any existing selection within that selection scope. If no selection exists, the gesture has no effect.

For most gestures, the hot spot of the gesture determines specifically which object the gesture applies to. If the hot spot occurs on any part of a selection, it applies to the whole selection.

Table 5.2 lists the common pen editing gestures. For these gestures, the hot spot of the gesture is the area inside the circle stroke of the gesture.

Table 5.2 Pen Editing Gestures

Gesture	Name	Operation
	circled-check	Edit (displays the writing tool editing window) for text; Properties for all other objects.
	circled-c	Copy
	circled-d	Delete (or Clear)
	circled-m	Menu
	circled-n	New line
	circled-p	Paste
	circled-s	Insert space
	circled-t	Insert tab
	circled-u	Undo
	circled-x	Cut
	circled-^	Insert text

These gestures may be localized in certain international versions. In Japanese versions, the circled-k gesture is used to convert Kana to Kanji.

Transfer Operations

Transfer operations are operations that involve (or can be derived from) moving, copying, and linking objects from one location to another. For example, printing an object is a form of a transfer operation because it can be defined as copying an object to a printer.

Three components make up a transfer operation: the object to be transferred, the destination of the transfer, and the operation to be performed. You can define these components either explicitly or implicitly, depending on which interaction technique you use.

The operation defined by a transfer is determined by the destination. Because a transfer may have several possible interpretations, you can define a default operation and other optimal operations, based on information provided by the source of the transfer and the compatibility and capabilities of the destination. For example, attempting to transfer an object to a container can result in one of the following alternatives:

- Rejecting the object.
- Accepting the object.
- Accepting a subset or transformed form of the object (for example, extract its content or properties but discard its present containment, or convert the object into a new type).

Most transfers are based on one of the following three fundamental operations.

Operation	Description
Move	Relocates or repositions the selected object. Because it does not change the basic identity of an object, a move operation is not the same as copying an object and deleting the original.
Copy	Makes a duplicate of an object. The resulting object is independent of its original. Duplication does not always produce an identical clone. Some of the properties of a duplicated object may be different from the original. For example, copying an object may result in a different name or creation date. Similarly, if some component of the object restricts copying, then only the unrestricted elements may be copied.
Link	Creates a connection between two objects. The result is usually an object in the destination that provides access to the original.

There are two different methods for supporting the basic transfer interface: the command method and the direct manipulation method.

Command Method

The command method for transferring objects uses the Cut, Copy, and Paste commands. Place these commands on the Edit drop-down menu and on the pop-up menu for a selected object. You can also include toolbar buttons to support these commands.

To transfer an object, the user:

1. Makes a selection.
2. Chooses either Cut or Copy.
3. Navigates to the destination (and sets the insertion location, if appropriate).
4. Chooses a Paste operation.

Cut removes the selection and transfers it (or a reference to it) to the Clipboard. Copy duplicates the selection (or a reference to it) and transfers it to the Clipboard. Paste completes the transfer operation. For example, when the user chooses Cut and Paste, remove the selection from the source and relocate it to the destination. For Copy and Paste, insert an independent duplicate of the selection and leave the original unaffected. When the user chooses Copy and Paste Link or Paste Shortcut, insert an object at the destination that is linked to the source.

Choose a form of Paste command that indicates how the object will be transferred into the destination. Use the Paste command by itself to indicate that the object will be transferred as native content. You can also use alternative forms of the Paste command for other possible transfers, using the following general form.

Paste [*type name*] [**as** *type name* | **to** *object name*]

For example, Paste Cells as Word Table, where [*type name*] is Cells and Word Table is the converted type name.


For more information about object names, including their type name, see Chapter 10, "Integrating with the System."

The following summarizes common forms of the Paste command.


For more information about these Paste command forms and the Paste Special dialog box, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects."

Command	Function
Paste	Inserts the object on the Clipboard as native content (data).
Paste [*type name*]	Inserts the object on the Clipboard as an OLE embedded object. The OLE embedded object can be activated directly within the destination.
Paste [*type name*] as Icon	Inserts the object on the Clipboard as an OLE embedded object. The OLE embedded object is displayed as an icon.
Paste Link	Inserts a data link to the object that was copied to the Clipboard. The object's value is integrated or transformed as native content within the destination, but remains linked to the original object so that changes to it are reflected in the destination.
Paste Link to [*object name*]	Inserts an OLE linked object, displayed as a picture of the object copied to the Clipboard. The representation is linked to the object copied to the Clipboard so that any changes to the original source object will be reflected in the destination.
Paste Shortcut	Inserts an OLE linked object, displayed as a shortcut icon, to the object that was copied to the Clipboard. The representation is linked to the object copied to the Clipboard so that any changes to the original source object will be reflected in the destination.
Paste Special	Displays a dialog box that gives the user explicit control over how to insert the object on the Clipboard.

Use the destination's context to determine what form(s) of the Paste operation to include based on what options it can offer to the user, which in turn may depend on the available forms of the object that its source location object provides. It can also be dependent on the nature or purpose of the destination. For example, a printer defines the context of transfers to it.

Typically, you will need only Paste and Paste Special commands. The Paste command can be dynamically modified to reflect the destination's default or preferred form by inserting the transferred object — for example, as native data or as an OLE embedded object. The Paste Special command can be used to handle any special forms of transfer. Although, if the destination's context makes it reasonable to provide fast access to another specialized form of transfer, such as Paste Link, you can also include that command.

Use the destination's context also to determine the appropriate side effects of the Paste operation. You may also need to consider the type of object being inserted by the Paste operation and the relationship of that type to the destination. The following are some common scenarios:

- When the user pastes into a destination that supports a specific insertion location, replace the selection in the destination with the transferred data. For example, in text or list contexts, where the selection represents a specific insertion location, replace the destination's active selection. In text contexts where there is an insertion location, but there is no existing selection, place the insertion point after the inserted object.

- For destinations with nonordered or Z-ordered contexts where there is no explicit insertion point, add the pasted object and make it the active selection. Also use the destination's context to determine where to place the pasted object. Consider any appropriate user contextual information. For example, if the user chooses the Paste command from a pop-up menu, you can use the pointer's location when the mouse button is clicked to place the incoming object. If the user supplies no contextual clues, place the object at the location that best fits the context of the destination — for example, at the next grid position.

- If the new object is automatically connected (linked) to the active selection (for example, table data and a graph), you may insert the new object in addition to the selection and make the inserted object the new selection.

You also use context to determine whether to display an OLE embedded or OLE linked object as content (view or picture of the object's internal data) or as an icon. For example, you can decide what presentation to display based on what Paste operation the user selects; Paste Shortcut implies pasting an OLE link as an icon. Similarly, the Paste Special command includes options that allow the user to specify how the transferred object should be displayed. If there is no user-supplied preference, the destination application defines the default. For documents, you typically display the inserted OLE object as in its content presentation. If icons better fit the context of your application, make the default Paste operation display the transferred OLE object as an icon.

The execution of a Paste command should not affect the content of the Clipboard. This allows data on the Clipboard to be pasted multiple times, although subsequent Paste operations should always result in copies of the original. However, a subsequent Cut or Copy command replaces the last entry on the Clipboard.

Direct Manipulation Method

The command method is useful when a transfer operation requires the user to navigate between source and destination. However, for many transfers, direct manipulation is a natural and quick method. In a direct manipulation transfer, the user selects and drags an object to the desired location, but because this method requires motor skills that may be difficult for some users to master, avoid using it as the exclusive transfer method. The best interfaces support the transfer command method for basic operations and direct manipulation transfer as a shortcut technique.

When a pen is being used as a pointing device, or when it drags an action handle, it follows the same conventions as dragging with mouse button 1. For pens with barrel buttons, use the barrel+drag action as the equivalent of dragging with mouse button 2. There is no keyboard interface for direct manipulation transfers.

You can support direct manipulation transfers to any visible object. The object (for example, a window or icon) need not be currently active. For example, the user can drop an object in an inactive window. The drop action activates the window. If an inactive object cannot accept a direct manipulation transfer, it (or its container) should provide feedback to the user.

How the transferred object is integrated and displayed in the drop destination is determined by the destination's context. A dropped object can be incorporated either as native data, an OLE object, a partial form of the object — such as its properties — or a transformed object. You determine whether to add to or replace an existing selection based on the context of the operation, using such factors as the formats available for the object, the destination's purpose, and any user-supplied information such as the specific location that the user drops or commands (or modes) that the user has selected. For example, an application can supply a particular type of tool for copying the properties of objects.

Default Drag and Drop

Default drag and drop transfers an object using mouse button 1. How the operation is interpreted is determined by what the destination defines as the appropriate default operation. As with the command method, the destination determines this based on information about the object (and the formats available for the object) and the context of the destination itself. Avoid defining a destructive operation as the default. When that is unavoidable, display a message box to confirm the intentions of the user.

Using this transfer technique, the user can directly transfer objects between documents defined by your application as well as to system resources, such as folders and printers. Support drag and drop following the same conventions the system supports: the user presses button 1 down on an object, moves the mouse while holding the button down, and then releases the button at the destination. For the pen, the destination is determined by the location where the user lifts the pen tip from the input surface.

The most common default transfer operation is Move, but the destination (dropped on object) can reinterpret the operation to be whatever is most appropriate. Therefore, you can define a default drag and drop operation to be another general transfer operation such as Copy or Link, a destination specific command such as Print or Send To, or even a specialized form of transfer such as Copy Properties.

Nondefault Drag and Drop

Nondefault drag and drop transfers an object using mouse button 2. In this case, rather than executing a default operation, the destination displays a pop-up menu when the user releases the mouse button, as shown in Figure 5.16. The pop-up menu contains the appropriate transfer completion commands.

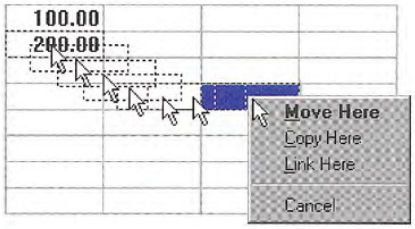


Figure 5.16 A nondefault drag and drop operation

The destination always determines which transfer completion commands to include on the resulting pop-up menu, usually factoring in information about the object supplied by the source location.

The form for nondefault drag and drop transfer completion verbs follows similar conventions as the Paste command. Use the common transfer completion verbs, Move Here, Copy Here, and Link Here, when the object being transferred is native data of the destination. When it is not, include the type name. You can also display alternative completion verbs that communicate the context of the destina-

tion; for example, a printer displays a Print Here command. For commands that support only a partial aspect or a transformation of an object, use more descriptive indicators — for example, Copy Properties Here, or Transpose Here.

Use the following general form for nondefault drag and drop transfer commands.

[*Command Name*] [*type name* | *object name*] **Here** [**as** *type name*]

The following summarizes common forms for nondefault transfer completion commands.

Command	Function
Move Here	Moves the selected object to the destination as native content (data).
Copy Here	Creates a copy of the selected object in the destination as native content.
Link Here	Creates a data link between the selected object and the destination. The original object's value is integrated or transformed as native data within the destination, but remains linked to the original object so that changes to it are reflected in the destination.
Move [*type name*] Here Copy [*type name*] Here	Moves or copies the selected object as an OLE embedded object. The OLE embedded object is displayed in its content presentation and can be activated directly within the destination.
Link [*type name*] Here	Creates an OLE linked object displayed as a picture of the selected object. The representation is linked to the selected object so that any changes to the original object will be reflected in the destination.

(Continued)

Command	Function
Move [*type name*] Here as Icon Copy [*type name*] Here as Icon	Moves or copies the selected object as an OLE embedded object and displays it as an icon.
Create Shortcut Here	Creates an OLE linked object to the selected object; displayed as a shortcut icon. The representation is linked to the selected object so that any changes to the original object will be reflected in the destination.

Define and appropriately display one of the commands in the pop-up menu to be the default drag and drop command. This is the command that corresponds to the effect of dragging and dropping with mouse button 1.

For more information about how to display default menu commands, see Chapter 13, "Visual Design."

Canceling a Drag and Drop Transfer

When a user drags and drops an object back on itself, interpret the action as cancellation of a direct manipulation transfer. Similarly, cancel the transfer if the user presses the ESC key during a drag transfer. In addition, include a Cancel command in the pop-up menu of a nondefault drag and drop action. When the user chooses this command, cancel the operation.

Differentiating Transfer and Selection When Dragging

Because dragging performs both selection and transfer operations, provide a convention that allows the user to differentiate between these operations. The convention you use depends on what is most appropriate in the current context of the object, or you can provide specialized handles for selection or transfer. The most common technique uses the location of the pointer at the beginning of the drag operation. If the pointer is within an existing selection, interpret the drag to be a transfer operation. If the drag begins outside of an existing selection, on the background's white space, interpret the drag as a selection operation.

Scrolling When Transferring by Dragging

When the user drags and drops an object from one scrollable area (such as a window, pane, or list box) to another, some tasks may require transferring the object outside the boundary of the area. Other tasks may involve dragging the object to a location not currently in view. In this latter case, it is convenient to automatically scroll the area (also known as *automatic scrolling* or autoscroll) when the user drags the object to the edge of that scrollable area. You can accommodate both these behaviors by using the velocity of the dragging action. For example, if the user is dragging the object slowly at the edge of the scrollable area, you scroll the area; if the object is being dragged quickly, do not scroll.

To support this technique during a drag operation, you sample the pointer's position at the beginning of the drag each time the mouse moves, and on an application-set timer (every 100 milliseconds recommended). If you use OLE drag and drop support, you need not set a timer. Store each value in an array large enough to hold at least three samples, replacing existing samples with later ones. Then calculate the pointer's velocity based on at least the last two locations of the pointer.

To calculate the velocity, sum the distance between the points in each adjacent sample and divide the total by the sum of the time elapsed between samples. Distance is the absolute value of the difference between the x and y locations, or (abs(x1 - x2) + abs(y1 - y2)). Multiply this by 1024 and divide it by the elapsed time to produce the velocity. The 1024 multiplier prevents the loss of accuracy caused by integer division.

Distance as implemented in this algorithm is not true Cartesian distance. This implementation uses an approximation for purposes of efficiency, rather than using the square root of the sum of the squares, (sqrt((x1 - x2)^2 + (y1 - y2)^2)), which is more computationally expensive.

You also predefine a hot zone along the edges of the scrollable area and a scroll time-out value. Use twice the width of a vertical scroll bar or height of a horizontal scroll bar to determine the width of the hot zone.

During the drag operation, scroll the area if the following conditions are met: the user moves the pointer within the hot zone, the current velocity is below a certain threshold velocity, and the scrollable area is able to scroll in the direction associated with the hot zone it is in. The recommended threshold velocity is 20 pixels per second. These conventions are illustrated in Figure 5.17.

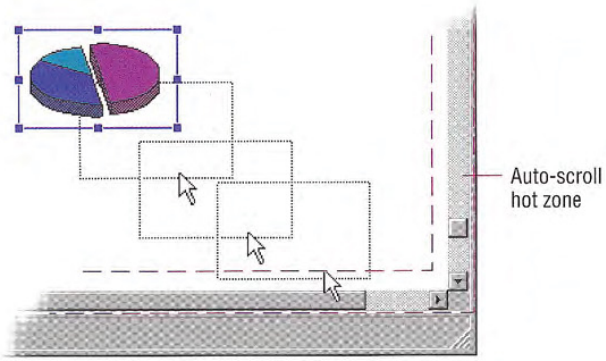


Figure 5.17 Automatic scrolling based on velocity of dragging

The amount you scroll depends on the type of information and reasonable scrolling distance. For example, for text, you typically scroll vertically one line at a time. Consider using the same scrolling granularity that is provided for the scroll bar arrows.

For more information about scrolling, see Chapter 6, "Windows."

To support continuous scrolling, determine what the scroll frequency you want to support — for example, four lines per second. After using a velocity check to initiate auto-scrolling, set a timer — for example, 100 milliseconds. When the timer expires, determine how long it has been since the last time you scrolled. If the elapsed time is greater than your scrolling frequency, scroll another unit. If not, reset your timer and check again when the timer completes.

Transfer Feedback

Because transferring objects is one of the most common user tasks, providing appropriate feedback is an important design factor. Inconsistent or insufficient feedback can result in user confusion.

For more information about designing transfer feedback, see Chapter 13, "Visual Design."

Command Method Transfers

For a command method transfer, remove the selected object visually when the user chooses the Cut command. If there are special circumstances that make removing the object's appearance impractical, you can instead display the selected object with a special appearance to inform the user that the Cut command was completed, but that the

object's transfer is pending. For example, the system displays icons in a checkerboard dither to indicate this state. You will also need to restore the visual state of the object if the user chooses Cut or Copy for another object before choosing a Paste command, effectively canceling the pending Cut command. The user will expect Cut to remove a selected object, so carefully consider the impact of inconsistency if you choose this alternate feedback.

The Copy command requires no special feedback. A Paste operation also requires no further feedback than that already provided by the insertion of the transferred object. However, if you did not remove the display of the object and used an alternate representation when the user chose the Cut command, you must remove it now.

Direct Manipulation Transfers

During a direct manipulation transfer operation, provide visual feedback for the object, the pointer, and the destination. Specifically:

- Display the object with selected appearance while the view it appears in has the focus. To indicate that the object is in a transfer state, you can optionally display the object with some additional appearance characteristics. For example, for a move operation, you can use the checkerboard dithered appearance used by the system to indicate when an icon is Cut. Change this visual state based on the default completion operation supported by the destination the pointer is currently over. Retain the representation of the object at the original location until the user completes the transfer operation. This not only provides a visual cue to the nature of the transfer operation, it provides a convenient visual reference point.

- Display a representation of the object that moves with the pointer. Use a presentation that provides the user with information about how the information will appear in the destination and that does not obscure the context of the insertion location. For example, when transferring an object into a text context, it is important that the insertion point not be obscured during the drag operation. A translucent or outline representation, as shown in Figure 5.18,

works well because it allows the underlying insertion location to be seen while also providing information about the size, position, and nature of the object being dragged.

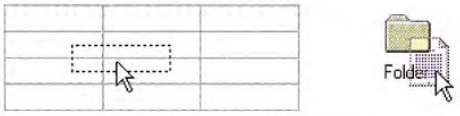


Figure 5.18 Outline and translucent representations for transfer operations

- The object's existing source location provides the transferred object's initial appearance, but any destination can change the appearance. Design the presentation of the object to provide feedback as to how the object will be integrated by that destination. For example, if an object will be embedded as an icon, display the object as an icon. If the object will be incorporated as part of the native content of the destination, then the presentation of the object that the destination displays should reflect that. For example, if a table being dragged into a document will be incorporated as a table, the representation could be an outline or translucent form of the table. On the other hand, if the table will be converted to text, display the table as a representation of text, such as a translucent presentation of the first few words in the table.

- Display the pointer appropriate to the context of the destination, usually used for inserting objects. For example, when dragging an object into a text editing context such that the object will be inserted between characters, display the usual text editing pointer (sometimes called the I-beam pointer).

- Display the interpretation of the transfer operation at the lower right corner of the pointer, as shown in Figure 5.19. No additional glyph is required for a move operation. Use a plus sign (+) when the transfer is a copy operation. Use the shortcut arrow graphic for linking.

Figure 5.19 Pointers - move, copy, and link operations

- Use visual feedback to indicate the receptivity of potential destinations. You can use selection highlighting and optionally animate or display a representation of the transfer object in the destination. Optionally, you can also indicate when a destination cannot accept an object by using the "no drop" pointer when the pointer is over it, as shown in Figure 5.20.

Figure 5.20 A "no drop " pointer

Specialized Transfer Commands

In some contexts, a particular form of a transfer operation may be so common, that introducing an additional specialized command is appropriate. For example, if copying existing objects is a frequent operation, you can include a Duplicate command. Following are some common specialized transfer commands.

Command	Function
Delete	Removes an object from its container. If the object is a file, the object is transferred to the Recycle Bin.
Clear	Removes the content of a container.
Duplicate	Copies the selected object.
Print	Prints the selected object on the default printer.
Send To	Displays a list of possible transfer destinations and transfers the selected object to the user selected destination.

Delete and Clear are often used synonymously. However, they are best differentiated by applying Delete to an object and Clear to the container of an object.

Shortcut Keys for Transfer Operations

Following are the defined shortcut techniques for transfer operations.

Shortcut	Operation
CTRL+X	Performs a Cut command.
CTRL+C	Performs a Copy command.
CTRL+V	Performs a Paste command.
CTRL+drag	Toggles the meaning of the default direct manipulation transfer operation to be a copy operation (provided the destination can support the copy operation). The modifier may be used with either mouse button.
ESC	Cancels a drag and drop transfer operation.

For more information about reserved and recommended shortcut key assignments, see Appendix B, "Keyboard Interface Summary."

Because of the wide use of these command shortcut keys throughout the interface, do not reassign them to other commands.

Creation Operations

Creating new objects is a common user action in the interface. Although applications can provide the context for object creation, avoid considering an application's interface as the exclusive means of creating new objects. Creation is typically based on some predefined object or specification and can be supported in the interface in a number of ways.

Copy Command

Making a copy of an existing object is the fundamental paradigm for creating new objects. Copied objects can be modified and serve as prototypes for the creation of other new objects. The transfer model conventions define the basic interaction techniques for copying objects. Copy and Paste commands and drag and drop manipulation provide this interface.

New Command

The New command facilitates the creation of new objects. New is a command applied to a specific object, automatically creating a new instance of the object's type. The New command differs from the Copy and Paste commands in that it is a single command that generates a new object.

Insert Command

The Insert command works similarly to the New command, except that it is applied to a container to create a new object, usually of a specified type, in that container. In addition to inserting native types of data, use the Insert command to insert objects of different types. By supporting OLE, you can support the creation of a wide range of objects. In addition, objects supported by your application can be inserted into data files created by other OLE applications.

For more information about inserting objects, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects."

Using Controls

You can use controls to support the automatic creation of new objects. For example, in a drawing application, buttons are often used to specify tools or modes for the creation of new objects, such as drawing particular shapes or controls. Buttons can also be used to insert OLE objects.

For more information about using buttons to create new objects, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects."

Using Templates

A *template* is an object that automates the creation of a new object. To distinguish its purpose, display a template icon as a pad with the small icon of the type of the object to be created, as shown in Figure 5.21.

Figure 5.21 A template icon

Define the New command as the default operation for a template object; this starts the creation process, which may either be automatic or request specific input from the user. Place the newly created object in the same location as the container of the template. If circumstances make that impractical, place the object in a common location, such as the desktop, or, during the creation process, include a prompt that allows a user to specify some other destination. In the former situation, display a message box informing the user where the object will appear.

Operations on Linked Objects

A *link* is a connection between two objects that represents or provides access to another object that is in another location in the same container or in a different, separate container. The components of this relationship include the link source (sometimes referred to as the referent) and the link or linked object (sometimes referred to as the reference). A linked object often has operations and properties independent of its source. For example, a linked object's properties can include attributes like update frequency, the path description of its link source, and the appearance of the linked object. The containers in which they reside provide access to and presentation of commands and properties of linked objects.

Links can be presented in various ways in the interface. For example, a *data link* propagates a value between two objects, such as between two cells in a worksheet or a series of data in a table and a chart. *Jumps* (also referred to as hyperlinks) provide navigational access to another object. An *OLE linked object* provides access to any operation available for its link source and also supplies a presentation of the link source. A shortcut icon is a link, displayed as an icon.

For more information about OLE linked objects, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects." For more information about jumps, see Chapter 12, "User Assistance."

When the user transfers a linked object, store both the absolute and relative path to its link source. The absolute path is the precise description of its location, beginning at the root of its hierarchy. The relative path is the description of its location relative to its current container.

The destination of a transfer determines whether to use the absolute or relative path when the user accesses the link source through the linked object. The relative path is the most common default path. However, regardless of which path you use, if it fails, use the alternative path. For example, if the user copies a linked object and its link source to another location, the result is a duplicate of the linked object and the link source. The relative path for the duplicate linked object is the location of the duplicate of the link source. The absolute path for the duplicate linked object is the description of the location of the initial link source. Therefore, when the user accesses the duplicate of the linked object, its inferred connection should be with the duplicate of the link source. If that connection fails — for example, because the user deletes the duplicate of the linked source — use the absolute path, the connection to the original link source.

Optionally, you can make the preferred path for a linked object a field in the property sheet for linked object. This allows the user to choose whether to have a linked object make use of the absolute or relative path to its link source.

When the user applies a link operation to a linked object, link to the linked object rather than its linked source. That is, linking a linked object results in a linked object linked to a linked object. If such an operation is not valid or appropriate - for example, because the linked object provides no meaningful context - then disable any link commands or options when the user selects a linked object.

Activation of a linked object depends on the kind of link. For example, a single click can activates a jump. However, a single click only results in selecting a data link or an OLE linked object. If you use a single click to do anything other than select the linked object,

distinguish the object by either presenting it as a button control, displaying the hand pointer (as shown in Figure 5.22) when the user moves the pointer over the linked object, or both. These techniques provide feedback to the user that the clicking involves more than selection.

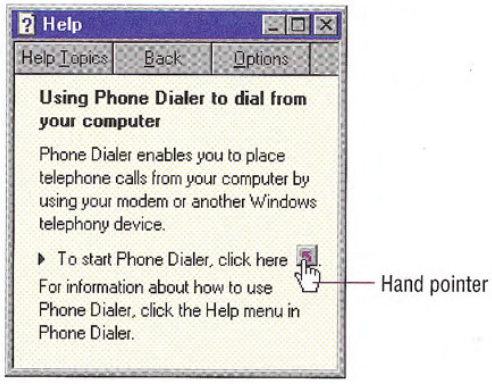
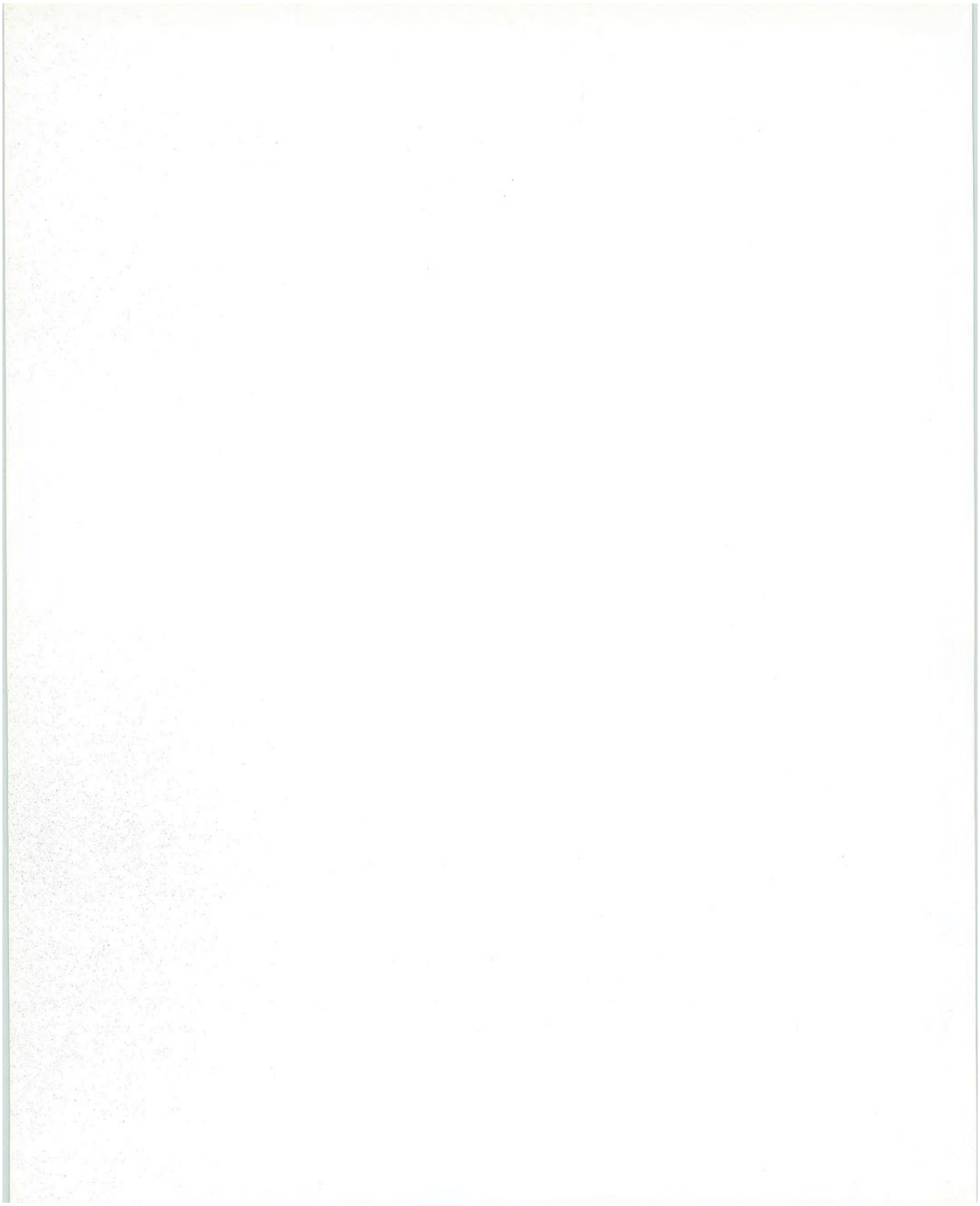


Figure 5.22 The hand pointer



Part II

Windows Interface Components

CHAPTER

Windows

Windows provide the fundamental way a user views and interacts with data. Consistency in window design is particularly important because it enables users to easily transfer their learning skills and focus on their tasks rather than learn new conventions. This chapter describes the common window types and presents guidelines for general appearance and operation.

Common Types of Windows

Because windows provide access to different types of information, they are classified according to common usage. Interacting with objects typically involves a *primary window* in which most primary viewing and editing activity takes place. In addition, multiple supplemental *secondary windows* can be included to allow users to specify parameters or options, or to provide more specific details about the objects or actions included in the primary window.

For more information about secondary windows, see Chapter 8, "Secondary Windows."

Primary Window Components

A typical primary window consists of a frame (or border) which defines its extent, and a title bar which identifies what is being viewed in the window. If the viewable content of the window exceeds the current size of the window, scroll bars are used. The window can also include other components like menu bars, toolbars, and status bars.

Figure 6.1 shows the common components of a primary window.

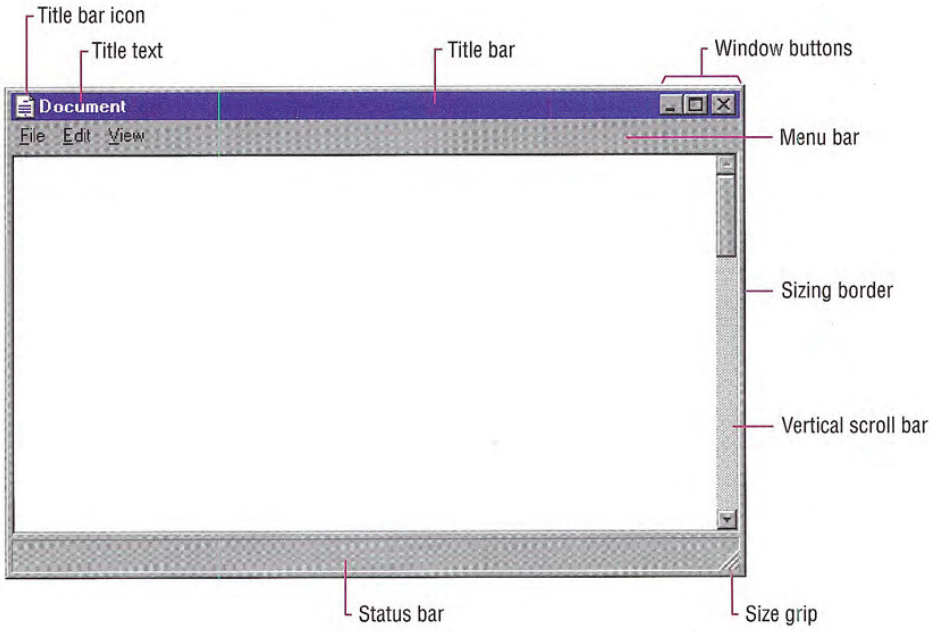


Figure 6.1 A primary window

Window Frames

Every window has a boundary that defines its shape. A sizable window has a distinct border that provides control points (handles) for resizing the window using direct manipulation. If the window cannot be resized, the border coincides with the edge of the window.

Title Bars

At the top edge of the window, inside its border, is the *title bar* (also referred to as the caption or caption bar), which extends across the width of the window. The title bar identifies what the window is viewing. It also serves as a control point for moving the window and an access point for commands that apply to the window and its

associated view. For example, clicking on the title bar with mouse button 2 displays the pop-up menu for the window. Pressing the ALT+SPACEBAR key combination also displays the pop-up menu for the window.

Title Bar Icons

A primary window includes the small version of the object's icon. The small icon appears in the upper left corner of the title bar and represents the object being viewed in the window. If the window represents a "tool" or utility application (that is, an application that does not create, load, and save its own data files), use the small version of the application's icon in its title bar, as shown in Figure 6.2.

For information about how to register icons for your application and data file types, see Chapter 10, "Integrating with the System." For more information about designing icons, see Chapter 13, "Visual Design."

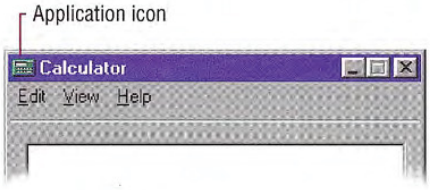


Figure 6.2 "Tool" title bar

If the application creates, loads and saves documents or data files and the window represents the view of one of its files, use the icon that represents its document or data file type in the title bar, as shown in Figure 6.3. Display the data file icon even if the user has not saved the file yet, rather than displaying the application icon and then the data file icon once the user saves the file.

Figure 6.3 Document title bar

If an application uses the multiple document interface (MDI) design, place the application's icon in the parent window's title bar, and place an icon that reflects the application's data file type in the child window's title bar, as shown in Figure 6.4.

For more information about MDI, see Chapter 9, "Window Management."

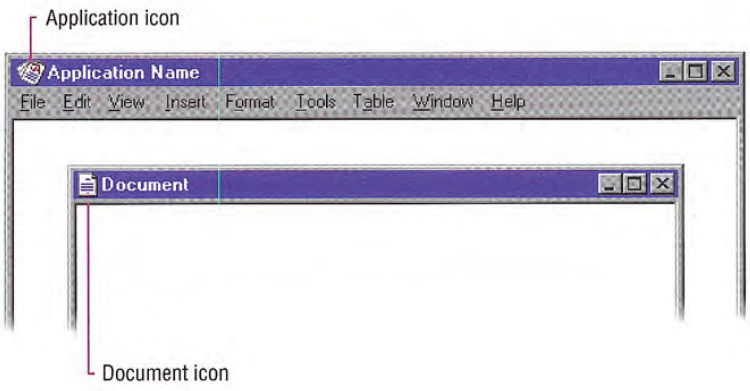


Figure 6.4 MDI application and document title bars

However, when a user maximizes the child window, and you hide its title bar and merge its title information with the parent, display the icon from the child window's title bar in the menu bar of the parent window. If multiple child windows are open within the MDI parent window, display only the icon from the active (topmost) child window.

When the user clicks the title bar icon with mouse button 2, display the pop-up menu for the object. Typically, the menu contains a similar set of commands that are available for the icon from which the window was opened, except that Close replaces Open. Also define Close as the default command, so when the user double-clicks the title bar icon, the window closes. Clicking elsewhere with button 2 on the title bar displays the pop-up menu for the window.

When the user clicks the title bar icon with mouse button 1, the system also displays the pop-up menu for the window. However, this behavior is only supported for backward compatibility with Windows 3.1. Avoid documenting it as the primary way to access the pop-up menu for the window. Instead, document the use of button 2 as the correct way to display the pop-up menu for the window.

Title Text

The window title text identifies the name of the object being viewed in the window. It should always correspond to the icon of the type you display in the title bar. It should also match the label of the icon in the file system that represents the object. For example, if the user opens a data file named "My Document" in the resulting window,

you display the icon for that document type followed by the name of the data file. You may also include the name of the application in use; however, if it is used, display the name of the data file first, followed by a dash and the application name, as shown in Figure 6.5.

Figure 6.5 Title text order: document name — application name

The order of the document (or data) filename and application name differs from the Windows 3.1 guidelines. The new convention is better suited for the design of a data-centered interface.

If the window represents a "tool" application that does not create or edit its own data files, such as the Windows Calculator, display the application's name, as displayed for the application's icon label, in the title bar. If the tool application operates as a utility for other files created by other applications — such as a special viewer or browser application — where the view displayed is not the primary open view of the file, or where the "tool" application requires an additional specification to indicate its context — such as the Windows Explorer — place the name of the application first, then include a dash and the specification text. For example, the title text of the Windows Explorer includes the name of the current container displayed in the browser.

For an MDI application, use the application's name in the parent window and the data file's name in the child windows. When the user maximizes the file's child window, format the title text following the same convention as a tool application, with the application's name first, followed by the data filename, as shown in Figure 6.6.

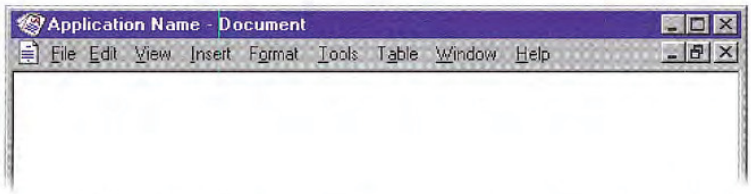


Figure 6.6 Document follows application name for maximized child window

When the user directly opens an application that displays a new data file, supply a name for the file and place it in the title bar, even if the user has not saved the file yet. Use the type name — for example Document (*n*), Sheet (*n*), Chart (*n*), where *n* is a number, as in Document (1). Make certain that the proposed name does not conflict with an existing name in the current directory. Also use this name as the proposed default filename for the object in the Save As dialog box. If it is impractical or inappropriate to supply a default name, display a placeholder in the title, such as (Untitled).

For more information about type names, see Chapter 10, "Integrating with the System." For more information about the Save As dialog box, see Chapter 8, "Secondary Windows."

Follow the same convention if your application includes a New command that creates new files. Avoid prompting the user for a name. Instead, you can supply a Save As dialog box that allows the user to confirm or change your proposed name when they save or close the file or attempt to create a new file.

Display a filename in the title bar exactly as it appears to the user in the file system, using both uppercase and lowercase letters. However, avoid displaying the file's extension or the path in the title bar. This information is not meaningful for most users and can make it more difficult for them to identify the file. However, because the system provides an option for users to display filename extensions, use the system-supplied functions to format a filename, which will display the filename appropriately based on the user's preference.

The **GetFileTitle** and **SHGetFileInfo** functions automatically format names correctly. For more information about these functions, see the documentation included in the Microsoft Win32 Software Development Kit (SDK).

If your application supports multiple windows for viewing the same file, you may use the title text to distinguish between the views — but use a convention that will not be confused as part of the filename. For example, you may want to append :*n*, where *n* represents the instance of the window, as in Document:2. Make certain you do not include this view designation as part of the filename you supply in the Save As dialog box.

If the name of the displayed object in the window changes — for example, when the user edits the name in the object's property sheet — update the title text to reflect that change. Always try to maintain a clear association between an object and its open window.

The title text and title bar icon should always represent the outmost container — the object that was opened — even if the user selects an embedded object or navigates the internal hierarchy of the object being viewed in the window. If you need an additional specification to clarify what the user is viewing, place this specification after the filename and separate it clearly from the filename, such as enclosing it in parentheses — for example, My HardDisk (C:). Because the system now supports long filenames, avoid additional specification whenever possible. Complex or verbose additions to the title text also make it more difficult for the user to easily read and identify the window.

When the width of the window does not allow you to display the complete title text, you may abbreviate the title text, being careful to maintain the essential information that allows the user to quickly identify the window.


For more information about abbreviating names, see Chapter 10, "Integrating with the System."

Avoid drawing directly into the title bar or adding other controls. Such added items can make reading the name in the title difficult, particularly because the size of the title bar varies with the size of the window. In addition, the system uses this area for displaying special controls. For example, in some international versions of Windows, the title area provides information or controls associated with the input of certain languages.

Title Bar Buttons

Include command buttons associated with the common commands of the primary window in the title bar. These act as shortcuts to specific window commands. Clicking a title bar button with mouse button 1 invokes the command associated with the command button. Optionally, you can also support clicking a title bar command button

with mouse button 2 to display the pop-up menu for the window. For the pen, tapping a window button invokes its associated command, and optionally you may support barrel-tapping it (or using the pen menu gesture) to display the pop-up menu for the window.

In a typical situation, one or more of the following buttons appear in a primary window (provided that the window supports the respective functions).

6.1 Title Bar Buttons

Button	Command	Operation
	Close	Closes the window.
	Minimize	Minimizes the window.
	Maximize	Maximizes the window.
	Restore	Restores the window.

The system does not support the inclusion of the context-sensitive Help button available for secondary windows. Applications wishing to provide this functionality can do so by including a Help toolbar button. Similarly, avoid including Maximize, Minimize, and Restore buttons in the title bars of secondary windows because those commands do not apply to those windows.

When displaying these buttons, use the following guidelines:

- When a command is not supported by a window, do not display its corresponding button.
- The Close button always appears as the rightmost button. Leave a gap between it and any other buttons.
- The Minimize button always precedes the Maximize button.
- The Restore button always replaces the Maximize button or the Minimize button when that command is carried out.

Basic Window Operations

The basic operations for a window include: activation and deactivation, opening and closing, moving, sizing, scrolling, and splitting. The following sections describe these operations.

Activating and Deactivating Windows

While the system supports the display of multiple windows, the user generally works within a single window at a time. This window is called the *active window*. The active window is typically at the top of the window Z order. It is also visually distinguished by its title bar that is displayed in the active window title color. All other windows are *inactive* with respect to the user's input; that is, while other windows can have ongoing processes, only the active window receives the user's input. The title bar of an inactive window displays the system inactive window color. Your application can query the system for the current values for the active title bar color and the inactive title bar color.

For more information about using the **GetSysColor** function to access the COLOR_ACTIVECAPTION and COLOR_INACTIVECAPTION constants, see the documentation included in the Win32 SDK.

The user activates a primary window by switching to it; this inactivates any other primary windows. To activate a window with the mouse or pen, the user clicks or taps on any part of the window, including its interior. If the window is minimized, the user clicks (taps) the button representing the window in the taskbar. From the keyboard, the system provides the ALT+TAB key combination for switching between primary windows. The SHIFT+ALT+TAB key also switches between windows, but in reverse order. (The system also supports ALT+ESC for switching between windows.) The reactivation of a window should not affect any pre-existing selection within it; the selection and focus are restored to the previously active state.

When the user reactivates a primary window, the window and all its secondary windows come to the top of the window order and maintain their relative positions. If the user activates a secondary window, its primary window comes to the top of the window order along with the primary window's other secondary windows.

When a window becomes inactive, hide the selection feedback (for example, display of highlighting or handles) of any selection within it to prevent confusion over which window is receiving keyboard input. A direct manipulation transfer (drag and drop) is an exception. Here, you can display transfer feedback if the pointer is over the window during the drag operation. Do not activate the window unless the user releases the mouse button (the pen tip is lifted) in that window.

Opening and Closing Windows

When the user opens a primary window, include an entry for it on the taskbar. If the window has been opened previously, restore the window to its size and position when it was last closed. If possible and appropriate, reinstate the other related view information, such as selection state, scroll position, and type of view. When opening a primary window for the first time, open it to a reasonable default size and position as best defined by the object or application. For details about storing state information in the system registry, see Chapter 10, "Integrating with the System."

Only primary windows, not secondary windows, should include an entry on the taskbar.

Because display resolution and orientation varies, your software should not assume a fixed display size, but rather adapt to the shape and size defined by the system. If you use standard system interfaces the system automatically places your windows relative to the current display configuration.

The **SetWindowPlacement** function is an example of a system interface that will automatically place windows correctly relative to the current display. For more information about this function, see the documentation included in the Win32 SDK.

Opening the primary window activates that window and places it at the top of the window order. If the use attempts to open a primary window that is already open within the same desktop, activate the existing window using the following recommendations. If the existing window is minimized, restore it when you activate it.

File type	Action when repeating an Open operation
Document or data file	Activates the existing window of the object and displays it at the top of the window Z order.
Application file	Displays a message box indicating that an open window of that application already exists and offers the user the option to switch to the open window or to open another window. Either choice activates the selected window and brings it to the top of the window Z order.
Document file that is already open in an MDI application window	Activates the existing window of the file. Its MDI parent window comes to the top of the window Z order, and the file appears at the top of the Z order within its MDI parent window.
Document file that is not already open, but its associated MDI application is already running (open)	Opens a new instance of the file's associated MDI application at the top of the window Z order and displays the child window for the file. Optionally, as an alternative, displays a message box indicating that an open window of that application already exists and offers the user the option to use the existing window or to open a new parent window.

For more information about MDI, see Chapter 9, "Window Management."

The user closes a primary window by clicking (for a pen, tapping the screen) the Close button in the title bar or choosing the Close command from the window's pop-up menu. Although the system supports double-clicking (with a pen, double-tapping the screen) on the title bar icon as a shortcut for closing the window for compatibility with previous versions of Windows, avoid documenting this as the primary way to close a primary window. Instead, document the Close button.

When the user chooses the Close command, if your application does not automatically save these changes and pending transactions or edits that have not yet been saved to file remain, display a message asking the user whether to save any changes, discard any changes, or cancel the Close operation before closing the window. If there are no pending transactions, just close the window. Follow this same convention for any other command that results in closing the primary window (for example, Exit or Shut Down).

For more information about supporting the Close command, see Chapter 5, "General Interaction Techniques."

When closing the primary window, close any of its dependent secondary windows as well. The design of your application determines whether closing the primary window also ends the application processes. For example, closing the window of a text document typically halts any application code or processes remaining for inputting or formatting text. However, closing the window of a printer has no effect on the jobs in the printer's queue. In both cases, closing the window removes its entry from the taskbar.

Moving Windows

The user can move a window either by dragging its title bar using the mouse or pen or by using the Move command on the window's pop-up menu. On most configurations, an outline representation moves with the pointer during the operation, and the window is redisplayed in the new location after the completion of the move. (The system also provides a display property setting that redraws the window dynamically as it is moved.) After choosing the Move command, the user can move the window with the keyboard interface by using arrow keys and pressing the ENTER key to end the operation and establish the window's new location. Never allow the user to reposition a window so that it cannot be accessed.

A window need not be active before the user can move it. The action of moving the window implicitly activates it.

Moving a window may clip or reveal information shown in the window. In addition, activation can affect the view state of the window — for example, the current selection can be displayed. However, when the user moves a window, avoid making any changes to the content being viewed in that window.

Resizing Windows

Make your primary windows resizable unless the information displayed in the window is fixed, such as in the Windows Calculator program. The system provides several conventions that support user resizing of a window.

Sizing Borders

The user resizes a primary window by dragging the sizing border with the mouse or pen at the edge of a window or by using the Size command on the window's menu. An outline representation of the window moves with the pointer. (On some configurations, the system may include a display option to dynamically redraw the window as it is sized.) After completing the size operation, the window assumes its new size. Using the keyboard, the user can size the window by choosing the Size command, using the arrow keys, and pressing the ENTER key.

A window does not need to be active before the user can resize it. The action of sizing the window implicitly makes it active, and it remains active after the sizing operation.

When the user resizes a window to be smaller, you must determine how to display the information viewed in that window. Use the context and type of information to help you choose your approach. The most common approach is to clip the information. However, in other situations where you want the user to see as much information as possible, you may want to consider using different methods, such as rewrapping or scaling the information. Use these variations carefully because they may not be consistent with the resizing behavior of most windows. In addition, avoid these methods when readability or maintaining the structural relationship of the information is important.

Although the size of a primary window can vary, based on the user's preference, you can define a window's maximum size. When defining this size, consider the reasonable usage within the window, and the size and orientation of the screen.

Maximizing Windows

Although the user may be able to directly resize a window to its maximum size, the Maximize command optimizes this operation. Include this command on a window's pop-up menu, and as the Maximize command button in the title bar of the window.

Maximizing a window increases the size of the window to its largest, optimum size. The system default setting for the maximum size is as large as the display, excluding the space used by the taskbar (or other application-defined desktop toolbars). For an MDI child window, the default maximize size fills the parent window. But, you can define the size to be less or, in some cases, more than the default dimensions.

When the user maximizes a window, replace the Maximize button with a Restore button. Then, disable the Maximize command and enable the Restore command on the pop-up menu for the window.

Minimizing Windows

Minimizing a window reduces it to its smallest size. To support this command, include it on the pop-up menu for the window and as the Minimize command button in the title bar of the window.

For primary windows, minimizing removes the window from the screen, but leaves its entry in the taskbar. For MDI child windows, the window resizes to a minimum size within its parent window. To minimize a window, the user chooses the Minimize command from the window's pop-up menu or the Minimize command button on the title bar.

When the user minimizes a window, disable the Minimize command on the pop-up menu and enable the Restore command.

Because the representation of minimized windows changed in Microsoft Windows 95, using an icon as the way to reflect state information may not be appropriate. As an alternative, you may want to consider creating a status indicator for the taskbar. For more information about status notification, see Chapter 10, "Integrating with the System."

Restoring Windows

Support the Restore command to restore a window to its previous size and position after the user has maximized or minimized the window. For maximized windows, enable this command on the window's pop-up menu and replace the Maximize button with the Restore button in the title bar of the window.

For minimized windows, also enable the Restore command in the pop-up menu of the window. The user restores a minimized primary window to its former size and position by clicking (for pens, tapping the screen) on its button in the taskbar that represents the window, selecting the Restore command on the pop-up menu of the window's taskbar button, or using the ALT+TAB (or the SHIFT+ALT+TAB) key combination.

Size Grip

When you define a sizable window, you may include a size grip. A *size grip* is a special handle for sizing a window. It is not exclusive to the sizing border. To size the window, the user drags the grip and the window resizes following the same conventions as the sizing border.

Always locate the size grip in the lower right corner of the window. Typically, this means you place the size grip at the right end of a horizontal scroll bar or the bottom of a vertical scroll bar. However, if you include a status bar in the window, display the size grip at the far corner of the status bar instead. Never display the size grip in both locations at the same time.

For more information about the use of the size grip in a status bar, see Chapter 7, "Menus, Controls, and Toolbars."

Scrolling Windows

When the information viewed in a window exceeds the size of that window, the window should support scrolling. Scrolling enables the user to view portions of the object that are not currently visible in a window. Scrolling is commonly supported through the use of a scroll bar. A *scroll bar* is a rectangular control consisting of *scroll arrows*, a *scroll box*, and a *scroll bar shaft*, as shown in Figure 6.7.

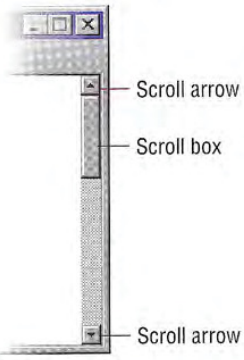


Figure 6.7 Scroll bar and its components

You can include a vertical scroll bar, a horizontal scroll bar, or both. Align a scroll bar with the vertical or horizontal edge of the window orientation it supports. If the content is never scrollable in a particular direction, do not include a scroll bar for that direction.

The common practice is to display scroll bars if the view requires some scrolling under any circumstances. If the window becomes inactive or resized so that its content does not require scrolling, you should continue to display the scroll bars. While removing the scroll bars when the window is inactive potentially allows the display of more information and feedback about the state of the window, it also requires the user to explicitly activate the window to scroll. Consistently displaying scroll bars provides a more stable environment.

Scroll bars are also available as separate window components. For more information about scroll bar controls, see Chapter 7, "Menus, Controls, and Toolbars."

Scroll Arrows

Scroll arrow buttons appear at each end of a 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. When the user clicks (for pens, tapping the screen) a scroll arrow, the data in the window moves, revealing information in the direction of the arrow in appropriate increments. The granularity of the increment depends on the nature of the content and context, but it is typically based on the size of a standard element. For example, you can

The default system support for scroll bars does not disable the scroll arrow buttons when the region or area is no longer scrollable in this direction. However, it does provide support for you to disable the scroll arrow button under the appropriate conditions.

use one line of text for vertical scrolling, one row for spreadsheets. You can also use an increment based a fixed unit of measure. Whichever convention you choose, maintain the same scrolling increment throughout a window. The objective is to provide an increment that provides smooth but efficient scrolling. When a window cannot be scrolled any further in a particular direction, disable the scroll arrow corresponding to that direction.

When scroll arrow buttons are pressed and held, they exhibit a special auto-repeat behavior. This action causes the window to continue scrolling in the associated direction as long as the pointer remains over the arrow button. If the pointer is moved off the arrow button while the user presses the mouse button, the auto-repeat behavior stops and does not continue unless the pointer is moved back over the arrow button (also when the pen tip is moved off the control).

Scroll Box

The scroll box, sometimes referred to as the elevator, thumb, or slider, moves along the scroll bar to indicate how far the visible portion is from the top (for vertical scroll bars) or from the left edge (for horizontal scroll bars). For example, if the current view is in the middle of a document, the scroll box in the vertical scroll bar is displayed in the middle of the scroll bar.

The size of the scroll box can vary to reflect the difference between what is visible in the window and the entire content of the file, as shown in Figure 6.8.

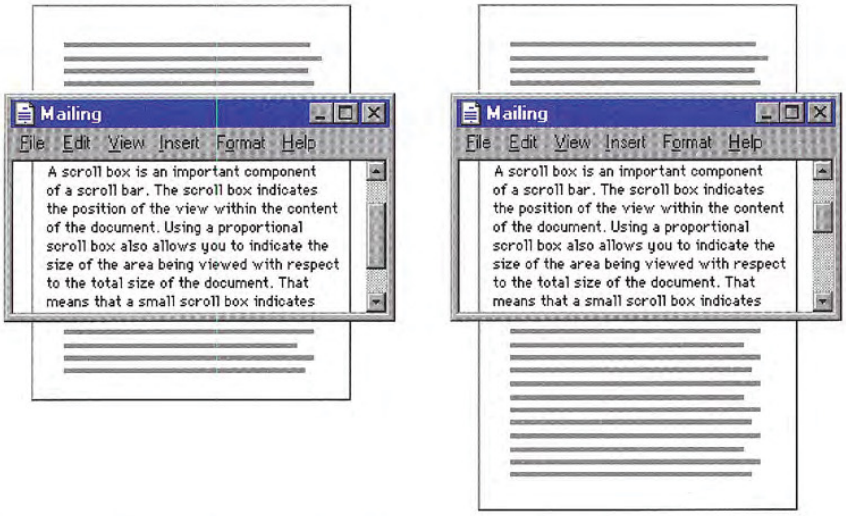


Figure 6.8 Proportional relationship between scroll box and content

For example, if the content of the entire document is visible in a window, the scroll box extends the entire length of the scroll bar, and the scroll arrows are disabled. Make the minimum size of the scroll box no smaller than the width of a window's sizing border.

The user can also scroll a window by dragging the scroll box. Update the view continuously as the user moves the scroll box. If you cannot support scrolling at a reasonable speed, you can scroll the information at the end of the drag operation as an alternative.

If the user starts dragging the scroll box and then moves the pointer outside of the scroll bar, the scroll box returns to its original position. The distance the user can move the pointer off the scroll bar before the scroll box snaps back to its original position is proportional to the width of the scroll bar. If dragging ends at this point, the scroll action is canceled — that is, no scrolling occurs. However, if the user moves the pointer back within the scroll-sensitive area, the scroll box returns to tracking the pointer movement. This behavior allows the user to scroll without having to remain within the scroll bar and to selectively cancel the initiation of a drag-scroll operation.

Dragging the scroll box to the end of the scroll bar implies scrolling to the end of that dimension; this does not always mean that the area cannot be scrolled further. If your application's document structure extends beyond the data itself, you can interpret dragging the scroll box to the end of its scroll bar as moving to the end of the data rather than the end of the structure. For example, a typical spreadsheet exceeds the data in it — that is, the spreadsheet may have 65,000 rows, with data only in the first 50 rows. This means you can implement the scroll bar so that dragging the scroll box to the bottom of the vertical scroll bar scrolls to the last row containing data rather than the last row of the spreadsheet. The user can use the scroll arrow buttons to scroll further to the end of the structure. This situation also illustrates why disabling the scroll arrow buttons can provide important feedback so that the user can distinguish between scrolling to the end of data from scrolling to the end of the extent or structure. In the example of the spreadsheet, when the user drags the scroll box to the end of the scroll bar, the arrow would still be shown as enabled because the user can still scroll further, but it would be disabled when the user scrolls to the end of the spreadsheet.

Scroll Bar Shaft

The scroll bar shaft not only provides a visual context for the scroll box, it also serves as part of the scrolling interface. Clicking in the scroll bar shaft should scroll the view an equivalent size of the visible area in the direction of the click. For example, if the user clicks in the shaft below the scroll box in a vertical scroll bar, scroll the view a distance equivalent to the height of the view. Where possible, allow overlap from the previous view, as shown in Figure 6.9. For example, if the user clicks below the scroll box, the bottom line becomes the top line of scrolled view. The same thing applies for clicking above the scroll box and horizontal scrolling. These conventions provide the user with a common reference point.

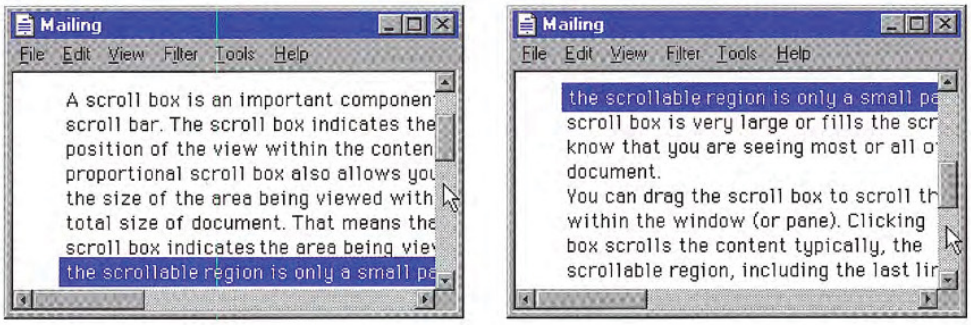


Figure 6.9 Scrolling with the scroll bar shaft by a screenful

Pressing and holding mouse button 1 with the pointer in the shaft auto-repeats the scrolling action. If the user moves the pointer outside the scroll-sensitive area while pressing the button, the scrolling action stops. The user can resume scrolling by moving the pointer back into the scroll bar area. (This behavior is similar to the effect of dragging the scroll box.)

Automatic Scrolling

The techniques previously summarized describe the explicit ways for scrolling. However, the user can also scroll as a secondary result of another user action. This type of scrolling is called *automatic scrolling*. The situations in which to support automatic scrolling are as follows:

- When the user begins or adjusts a selection and drags it past the edge of the scroll bar or window, scroll the area in the direction of the drag.

- When the user drags an object and approaches the edge of a scrollable area, scroll the area following the recommended auto-scroll conventions covered in Chapter 5, "General Interaction Techniques." Base the scrolling increment on the context of the destination and, if appropriate, on the size of the object being dragged.

- When the user enters text from the keyboard at the edge of a window or moves or copies an object into a location at the edge of a window, the view should scroll to allow the user to focus on the inserted information. The amount to scroll depends on context. For example, for text, vertically scroll a single line at a time. When scrolling horizontally, scroll in units greater than a single character to prevent continuous or uneven scrolling. Similarly, when the user transfers a graphic object near the edge of the view, base scrolling on the size of the object.

- If an operation results in a selection or moves the cursor, scroll the view to display the new selection. For example, for a Find command that selects a matching object, scroll the object into view because usually the user wants to focus on that location. In addition, other forms of navigation may cause scrolling. For example, completing an entry field in a form may result in navigating to the next field. In this case, if the field is not visible, the form can scroll to display it.

Keyboard Scrolling

Use navigation keys to support scrolling with the keyboard. When the user presses a navigation key, the cursor moves to the appropriate location. For example, in addition to moving the cursor, pressing arrow keys at the edge of a scrollable area scrolls in the corresponding direction. Similarly, the PAGE UP and PAGE DOWN keys are comparable to clicking in the scroll bar shaft, but they also move the cursor.

Optionally, you can use the SCROLL LOCK key to facilitate keyboard scrolling. In this case, when the SCROLL LOCK key is toggled on and the user presses a navigation key, scroll the view without affecting the cursor or selection.

Placing Adjacent Controls

It is sometimes convenient to locate controls or status bars adjacent to a scroll bar and position the end of the scroll bar to accommodate them. Take care when placing adjacent elements; too many can make it difficult for users to scroll, particularly if you reduce the scroll bar too much. If you need a large number of controls, consider using a conventional toolbar instead.

For more information about toolbars, see Chapter 7, "Menus, Controls, and Toolbars."

Splitting Windows

A window can be split into two or more separate viewing areas, which are called *panes*. For example, a split window allows the user to examine two parts of a document at the same time. You can also use a split window to display different, yet simultaneous, views of the same information, as shown in Figure 6.10.

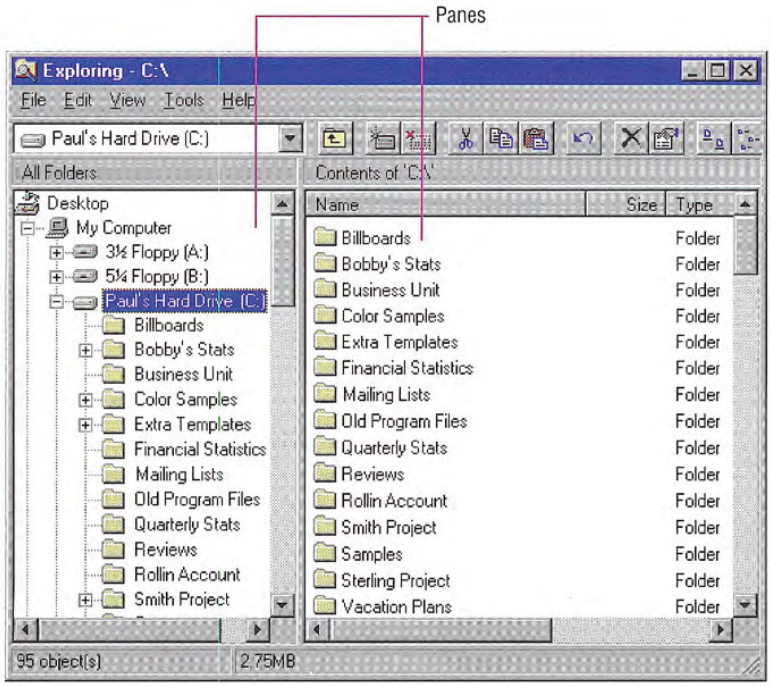


Figure 6.10 A split window

While you can use a split window pane to view the contents of multiple files or containers at the same time, displaying these in separate windows typically allows the user to better identify the files as individual elements. When you need to present views of multiple files as a single task, consider window management techniques, such as MDI.

The panes that appear in a split window can be implemented either as part of a window's basic design or as a user-configurable option. To support splitting a window that is not presplit by design, include a split box. A *split box* is a special control placed adjacent to the end of a scroll bar that splits or adjusts the split of a window. The size of the split box should be just large enough for the user to successfully target it with the pointer; the default size of a size handle, such as the window's sizing border, is a good guideline. Locate the split box at the top of the up arrow button of the vertical scroll bar, as shown in Figure 6.11, or to the left of the left arrow button of a horizontal scroll bar.

Figure 6.11 Split box location

The user splits a window by dragging the split box to the desired position. When the user positions the hot spot of the pointer over a split box, change the pointer's image to provide feedback and help the user target the split box. While the user drags the split box, move a representation of the split box and split bar with the pointer, as shown in Figure 6.12.

At the end of the drag, display a visual separator, called the *split bar*, that extends from one side of the window to the other, defining the edge between the resulting panes, as shown in Figure 6.12. Base the size for the split bar to be, at a minimum, the current setting for the size of window sizing borders. This allows you to appropriately adjust when a user adjusts size borders. If you display the split box after the split operation, place it adjacent to the split bar.

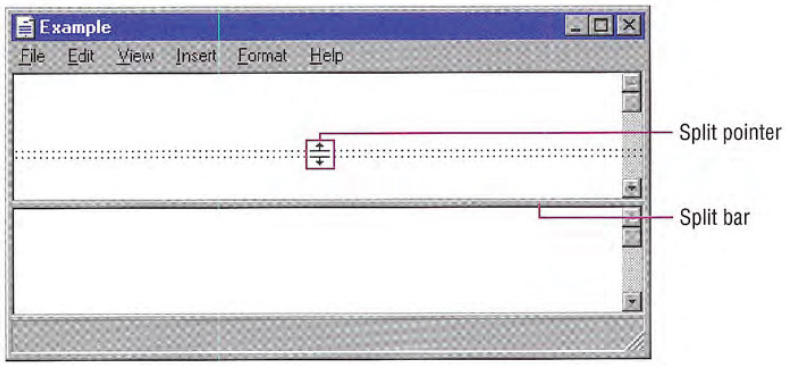


Figure 6.12 Moving the split bar

You can support dragging the split bar (or split box) to the end of the scroll bar to close the split. Optionally, you can also support double-clicking (or, for pens, double-tapping) as a shortcut technique for splitting the window at some default location (for example, in the middle of the window or at the last split location) or for removing the split. This technique works best when the resulting window panes display peer views. It may not be appropriate when the design of the window requires that it always be displayed as split or for some types of specialized views.

To provide a keyboard interface for splitting the window, include a Split command on the window or view's menu. When the user chooses the Split command, split the window in the middle or in a context-defined location. Support arrow keys for moving the split box up or down; pressing the ENTER key sets the split at the current location. Pressing the ESC key cancels the split mode.

You can also use other commands to create a split window. For example, you can define specialized views that, when selected by the user, split a window to a fixed or variable set of panes. Similarly, you can enable the user to remove the split of a window by closing a view pane or by selecting another view command.

When the user splits a window, add scroll bars if the resulting panes require scrolling. In addition, you may need to scroll the information in panes so that the split bar does not obscure the content over which it appears. Use a single scroll bar, at the appropriate side of the window, for a set of panes that scroll together. However, if the panes each require independent scrolling, a scroll bar should appear in each pane for that purpose. For example, the vertical scroll bars of a set of panes in a horizontally split window are typically controlled separately.

When you use split window panes to provide separate views, independently maintain each pane's view properties, such as view type and selection state. Display only the selection in the active pane. However, if the selection state is shared across the panes, display a selection in all panes and support selection adjustment across panes.

When a window is closed, save the window's split state (that is, the number of splits, the place where they appear, the scrolled position in each split, and its selection state) as part of the view state information for that window so that it can be restored the next time the window is opened.

CHAPTER

Menus, Controls, and Toolbars

Microsoft Windows provides a number of interactive components that make it easy to provide interfaces to carry out commands and specify values. These components also provide a consistent structure and set of interface conventions. This chapter describes the interactive elements of menus, controls, and toolbars, and how to use them.

Menus

Menus list commands available to the user. By making commands visible, menus leverage user recognition rather than depending on user recollection of command names and syntax.

There are several types of menus, including drop-down menus, pop-up menus, and cascading menus. The following sections cover these menus in more detail.

The Menu Bar and Drop-down Menus

A *menu bar*, one of the most common forms of a menu interface, is a special area displayed across the top of a window directly below the title bar (as shown in Figure 7.1). A menu bar includes a set of entries called *menu titles*. Each menu title provides access to a *drop-down menu* composed of a collection of *menu items*, or choices.

Figure 7.1 A menu bar

The content of the menu bar and its drop-down menus are determined by the functionality of your application and the context of a user's interaction. You can also optionally provide user configuration of the menu structure, including hiding the menu bar. If you provide this kind of option, supplement the interface with other components such as pop-up menus, handles, and toolbars, so that a user can access the functionality typically provided by the menu bar.

When displayed, a drop-down menu appears as a panel with its menu items arranged in a column. While the system supports multiple columns for a drop-down menu, avoid this form of presentation because it adds complexity to browsing and interaction of the menu.

Drop-down Menu Interaction

When the user chooses a menu title, it displays its associated drop-down menu. To display a drop-down menu with the mouse, the user points to the menu title and presses or clicks mouse button 1. This action highlights the menu title and opens the menu. Tapping the menu title with a pen has the same effect as clicking the mouse.

If the user opens a menu by pressing the mouse button while the pointer is over the menu title, the user can drag the pointer over menu items in the drop-down menu. As the user drags, each menu item is highlighted, tracking the pointer as it moves through the menu. Releasing the mouse button with the pointer over a menu item chooses the command associated with that menu item and the system removes the drop-down menu. If the user moves the pointer off the menu and then releases the mouse button, the menu is "canceled" and the drop-down menu is removed. However, if the user moves the pointer back onto the menu before releasing the mouse button, the tracking resumes and the user can still select a menu item.

If the user opens a menu by clicking on the menu title, the menu title is highlighted and the drop-down menu remains displayed until the user clicks the mouse again. Clicking a menu item in the drop-down menu or dragging over and releasing the mouse button on a menu item chooses the command associated with the menu item and re

moves the drop-down menu. When the system displays a drop-down menu, clicking its associated menu title again cancels the menu and removes the drop-down. Clicking another menu title also results in canceling any displayed drop-down menu, and displays the menu associated with that menu title.

The keyboard interface for drop-down menus uses the ALT key to activate the menu bar. When the user presses an alphanumeric key while holding the ALT key, or after the ALT key is released, the system displays the drop-down menu whose access key for the menu title matches the alphanumeric key (matching is not case sensitive). Pressing a subsequent alphanumeric key chooses the menu item in the drop-down menu with the matching access character.

The user can also use arrow keys to access drop-down menus from the keyboard. When the user presses the ALT key, but has not yet selected a drop-down menu, LEFT ARROW and RIGHT ARROW keys highlight the previous or next menu title, respectively. At the end of the menu bar, pressing another arrow key in the corresponding direction wraps the highlight around to the other end of the menu bar. Pressing the ENTER key displays the drop-down menu associated with the selected menu title. If a drop-down menu is already displayed on that menu bar, then pressing LEFT ARROW or RIGHT ARROW navigates the highlight to the next drop-down menu in that direction, unless the drop-down menu displays its content in multiple columns, in which case the arrow keys move the highlight to the next column in that direction, and then to the next drop-down menu.

Pressing UP ARROW or DOWN ARROW in the menu bar also displays a drop-down menu if none is currently open. In an open drop-down menu, pressing these keys moves to the next menu item in that direction, wrapping the highlight around at the top or bottom. If the drop-down menu has multiple columns, then pressing the arrow keys first wraps the highlight around to the next column.

The user can cancel a drop-down menu by pressing the ALT key whenever the menu bar is active. This not only closes the drop-down menu, it also deactivates the menu bar. Pressing the ESC key also cancels a drop-down menu. However, the ESC key cancels only the current menu level. For example, if a drop-down menu is open, pressing ESC closes the drop-down menu, but leaves its menu title highlighted. Pressing ESC a second time unhighlights the menu title and deactivates the menu bar, returning input focus to the content information in the window.

You can assign shortcut keys to commands in drop-down menus. When the user presses a shortcut key associated with a command in the menu, the command is carried out immediately. Optionally, you can also highlight its menu title, but do not display the drop-down.

Common Drop-down Menus

This section describes the conventions for drop-down menus commonly used in applications. While these menus are not required for all applications, apply these guidelines when including these menus in your software's interface.

The File Menu

The File menu provides an interface for the primary operations that apply to a file. Your application should include commands such as Open, Save, Send To, and Print. These commands are often also included on the pop-up menu of the icon displayed in the title bar of the window.

For more information about the commands in the pop-up menu for a title bar icon, see the section, "Icon Pop-up Menus," later in this chapter.

If your application supports an Exit command, place this command at the bottom of the File menu preceded by a menu separator. When the user chooses the Exit command, close any open windows and files, and stop any further processing. If the object remains active even when its window is closed — for example, like a folder or printer — then include the Close command instead of Exit.

The Edit Menu

Include general purpose editing commands on the Edit menu. These commands include the Cut, Copy, and Paste transfer commands, OLE object commands, and the following commands (if they are supported).

For more information about transfer commands, see Chapter 5, "General Interaction Techniques."

Command	Function
Undo	Reverses last action.
Repeat	Repeats last action.
Find and Replace	Searches for and substitutes text.
Delete	Removes the current selection.
Duplicate	Creates a copy of the current selection.

Include these commands on this menu and on the pop-up menu of the selected object.

The View Menu

Put commands on the View menu that change the user's view of data in the window. Include commands on this menu that affect the view and not the data itself — for example, Zoom or Outline. Also include commands for controlling the display of particular interface elements in the view — for example, Show Ruler. Also place these commands on the pop-up menu of the window or pane.

The Window Menu

Use the Window menu in multiple document interface-style (MDI) applications for commands associated with managing the windows within an MDI workspace. Also include these commands on the pop-up menu of the parent MDI window.


For more information about the design of MDI software, see Chapter 9, "Window Management."

The Help Menu

Use the Help menu for commands that provide access to Help information. Include a Help Topics command; this command provides access to the Help Topics browser, which displays topics included in your application's Help file. Alternatively, you can provide individual commands that access specific pages of the Help Topics browser, such as Contents, Index, and Find Topic. You can also include other user assistance commands on this drop-down menu.


For more information about the Help Topics browser and support for user assistance, see Chapter 12, "User Assistance."

If you provide access to copyright and version information for your application, include an About *application name* command on this menu. When the user chooses this command, display a window containing the application's name, version number, copyright information, and any other informational properties related to the application. Display this information in a dialog box or alternatively as a copyright page of the property sheet of the application's main executable file. Do not use an ellipsis at the end of this command, because the resulting window does not require the user to provide any further parameters.

Pop-up Menus

Even if you include a menu bar in your software's interface, you should also support pop-up menus. Pop-up menus provide an efficient way for the user to access the operations of objects, as shown in Figure 7.2. Because pop-up menus are displayed at the pointer's current location, they eliminate the need for the user to move the pointer to the menu bar or a toolbar. In addition, because you populate pop-up menus with commands specific to the object or its immediate context, they reduce the number of commands the user must browse through. Pop-up menus also minimize screen clutter because they are displayed only upon demand and do not require dedicated screen space.

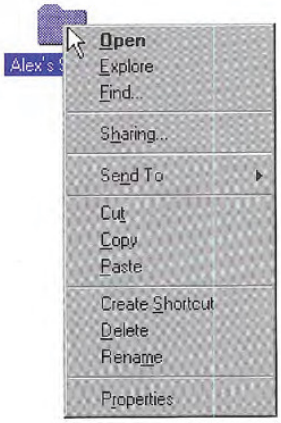


Figure 7.2 A pop-up menu

While a pop-up menu looks similar to a drop-down menu, a pop-up menu should only contain commands that apply to the selected object or objects and its context, rather than commands grouped by function. For example, a pop-up menu for a text selection can include commands for moving and copying the text and access to the font properties of the text and the paragraph properties of which the selection is a part. However, keep the size of the pop-up menu as small as possible by limiting the items on the menu to common, frequent actions. It is better to include a single Properties command and allow the user to navigate among properties in the resulting property sheet than to list individual properties in the pop-up menu.

The container or the composition of which a selection is a part typically supplies the pop-up menu for the selection. Similarly, the commands included on a pop-up menu may not always be supplied by the object itself, but rather be a combination of those commands provided by the object and by its current container. For example, the pop-up menu for a file in a folder includes transfer commands. In this case, the folder (container) supplies the commands, not the files. Pop-up menus for OLE objects follow these same conventions.

Avoid using a pop-up menu as the exclusive means to a particular operation. At the same time, the items in a pop-up menu need not be limited only to commands that are provided in drop-down menus.

When ordering the commands in a pop-up menu, use the following guidelines:

- Place the object's primary commands first (for example, commands such as Open, Play, and Print), transfer commands, other commands supported by the object (whether provided by the object or by its context), and the What's This? command (when supported).
- Order the transfer commands as Cut, Copy, Paste, and other specialized Paste commands.
- Place the Properties command, when present, as the last command on the menu.

For more information about transfer commands and the Properties command, see Chapter 5, "General Interaction Techniques." For more information about the What's This? command, see Chapter 12, "User Assistance."

Pop-up Menu Interaction

With a mouse, the user displays a pop-up menu by clicking an object with button 2. The down transition of the mouse button selects the object. Upon the up transition, display the menu to the right and below the hot spot of the pointer adjusted to avoid the menu being clipped by the edge of the screen.

If the pointer is over an existing selection when the user invokes a pop-up menu, display the menu that applies to that selection. If the menu is outside a selection but within the same selection scope, then establish a new selection (usually resetting the current selection in that scope) at the button down point and display the menu for the new selection. Dismiss the pop-up menu when the user clicks outside the menu with button 1 or if the user presses the ESC key.

You can support pop-up menus for objects that are implicitly selected or cannot be directly selected, such as scroll bars or items in a status bar. When providing pop-up menus for objects such as controls, include commands for the object that the control represents, rather than for the control itself. For example, a scroll bar represents a navigational view of a document, so commands might include Beginning of Document, End of Document, Next Page, and Previous Page. But when a control represents itself as an object, as in a forms layout or window design environment, you can include commands that apply to the control—for example, commands to move or copy the control.

The pen interface uses an action handle in pen-enabled controls to access the pop-up menu for the selection. Tapping the action handle displays the pop-up menu, as shown in Figure 7.3.

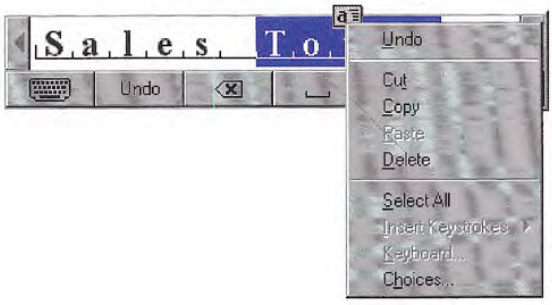


Figure 7.3 Using an action handle to provide pen access to pop-up menus

In addition, you can use techniques like barrel-tapping or the pop-up menu gesture to display a pop-up menu. This interaction is equivalent to a mouse button 2 click.

Use SHIFT+F10 and the Application key (for keyboards that support the Windows keys specification) to provide keyboard access for pop-up menus. In addition, menu access keys, arrow keys, ENTER, and ESC keys all operate in the same fashion in the menu as they do in drop-down menus. To enhance space and visual efficiency, avoid including shortcut keys in pop-up menus.

The system provides a message, WM_CONTEXTMENU, when the user presses a system defined pop-up menu key. For more information about this message, see documentation included in the Microsoft Win32 Software Development Kit (SDK).

Common Pop-up Menus

The pop-up menus included in any application depend on the objects and context supplied by that application. The following sections describe common pop-up menus for Windows-based applications.

The Window Pop-up Menu

The window pop-up menu is the pop-up menu associated with a window — do not confuse it with the Window drop-down menu found in MDI applications. The window pop-up menu replaces the Windows 3.1 Control menu, also referred to as the System menu. For example, a typical primary window includes Close, Restore, Move, Size, Minimize, and Maximize.

You can also include other commands on the window's menu that apply to the window or the view within the window. For example, an application can append a Split command to the menu to facilitate splitting the window into panes. Similarly, you can add commands that affect the view, such as Outline, commands that add, remove, or filter elements from the view, such as Show Ruler, or commands that open certain subordinate or special views in secondary windows, such as Show Color Palette.

A secondary window also includes a pop-up menu. Usually, because the range of operations are more limited than in a primary window, a secondary window's pop-up menu includes only Move and Close commands, or just Move. Palette windows can also include an Always on Top command that sets the window to always be on top of its parent window and secondary windows of its parent window.

The user displays a window's pop-up menu by clicking mouse button 2 anywhere in the title bar area, excluding the title bar icon. Clicking on the title bar icon with button 2 displays the pop-up menu for the object represented by the icon. To avoid confusing users, if you do not provide a pop-up menu for the title bar icon, do not display the pop-up for the window when the user clicks with button 2 on the title bar icon.

For the pen, performing barrel-tapping or the equivalent pop-up menu gesture on these areas displays the menu. Pressing ALT+SPACEBAR also displays the menu. The pop-up for the window can also be accessed from the keyboard by the user pressing the ALT key and then using the arrow keys to navigate beyond the first or last entry in the menu bar. In MDI applications, the pop-up menu for a child window can also be accessed this way or directly using ALT+HYPHEN.

For compatibility with previous versions of Windows, the system also supports clicking button 1 on the icon in the title bar to access the pop-up menu of a window. However, do not document this as the primary method for accessing the pop-up menu for the window. Document only the button 2 technique.

Icon Pop-up Menus

Pop-up menus displayed for icons include operations of the objects represented by those icons. Accessing the pop-up menu of an application or document icon follows the standard conventions for pop-up menus, such as displaying the menus with a mouse button 2 click.

An icon's container application supplies the pop-up menu for the icon. For example, pop-up menus for icons placed in standard folders or on the desktop are automatically provided by the system. However, your application supplies the pop-up menus for OLE embedded or linked objects placed in it — that is, placed in the document or data files your application supports.

For more information about supporting pop-up menus for OLE objects, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects."

The container populates the pop-up menu for an icon with commands the container supplies for its content, such as transfer commands and those registered by the object's type. For example, an application can register a New command that automatically generates a new data file of the type supported by the application.

For more information about registering commands, see Chapter 10, "Integrating with the System."

The pop-up menu of an application's icon, for example, the Microsoft WordPad executable file, should include the commands listed in Table 7.1.

Table 7.1 Application File Icon Pop-up Menu Commands

Command	Meaning
Open	Opens the application file.
Send To	Displays a submenu of destinations to which the file can be transferred. The content of the submenu is based on the content of the system's Send To folder.
Cut	Marks the file for moving. (Registers the file on the Clipboard.)
Copy	Marks the file for duplication. (Registers the file on the Clipboard.)
Paste	Attempts to open the file registered on the Clipboard with the application.
Create Shortcut	Creates a shortcut icon of the file.
Delete	Deletes the file.
Rename	Allows the user to edit the filename.
Properties	Displays the properties for the file.

An icon representing a document or data file typically includes the following common menu items for the pop-up menu for its icon.

Table 7.2 Document or Data File Icon Pop-up Menu Commands

Command	Meaning
Open	Opens the file's primary window.
Print	Prints the file on the current default printer.
Quick View	Displays the file using a special viewing tool window.
Send To	Displays a submenu of destinations to which the file can be transferred. The content of the submenu is based on the content of the system's Send To folder.
Cut	Marks the file for moving. (Registers the file on the Clipboard.)
Copy	Marks the file for duplication. (Registers the file on the Clipboard.)
Delete	Deletes the file.
Rename	Allows the user to edit the filename.
Properties	Displays the properties for the file.

With the exception of the Open and Print commands, the system automatically provides these commands for icons when they appear in system containers, such as the desktop or folders. If your application supplies its own containers for files, you need to supply these commands.

For the Open and Print commands to appear on the menu, your application must register these commands in the system registry. You can also register additional or replacement commands. For example, you can optionally register a Quick View command that displays the content of the file without running the application and a What's This? command that displays descriptive information for your data file types.

 For more information about registering commands and the Quick View command, see Chapter 10, "Integrating with the System." For more information about the What's This? command, see Chapter 12, "User Assistance."

The icon in the title bar of a window represents the same object as the icon the user opens. As a result, the application associated with the icon also includes a pop-up menu with appropriate commands for the title bar's icon. When the icon of an application appears in the

title bar, include the same commands on its pop-up menu as are included for the icon that the user opens, unless a particular command cannot be applied when the application's window is open. In addition, replace the Open command with Close.

Similarly, when the icon of the data or document file appears in the title bar, you also use the same commands as found on its file icon, with the following exceptions: replace the Open command with a Close command and add Save if the edits in the document require explicit saving to file.

For an MDI application, supply a pop-up menu for the application icon in the parent window, following the conventions for application title bar icons. Also consider including the following commands where they apply.


For more information about the design of MDI-style applications, see Chapter 9, "Window Management."

Table 7.3 Optional MDI Parent Window Title Bar Icon Pop-up Menu Commands

Command	Meaning
New	Creates a new data file or displays a list of data file types supported by the application from which the user can choose.
Save All	Saves all data files open in the MDI workspace, and the state of the MDI window.
Find	Displays a window that allows the user to specify criteria to locate a data file.

In addition, supply an appropriate pop-up menu for the title bar icon that appears in the child window's title bar. You can follow the same conventions for non-MDI data files.

Cascading Menus

A *cascading menu* (also referred to as a *hierarchical menu* or child menu) is a submenu of a menu item. The visual cue for a cascading menu is the inclusion of a triangular arrow display adjacent to the label of its parent menu item.

You can use cascading menus to provide user access to additional choices rather than taking up additional space in the parent menu. They may also be useful for displaying hierarchically related objects.

Be aware that cascading menus can add complexity to the menu interface by requiring the user to navigate further through the menu structure to get to a particular choice. Cascading menus also require more coordination to handle the changes in direction necessary to navigate through them.

In light of these design tradeoffs, use cascading menus sparingly. Minimize the number of levels for any given menu item, ideally limiting your design to a single submenu. Avoid using cascading menus for frequent, repetitive commands.

As an alternative, make choices available in a secondary window, particularly when the choices are independent settings; this allows the user to set multiple options in one invocation of a command. You can also support many common options as entries on a toolbar.

The user interaction for a cascading menu is similar to that of a drop-down menu from the menu bar, except a cascading menu displays after a short time-out. This avoids the unnecessary display of the menu if the user is browsing or navigating to another item in the parent menu. Once displayed, if the user moves the pointer to another menu item, the cascading menu is removed after a short time-out. This time-out enables the user to directly drag from the parent menu into an entry in its cascading menu.

Menu Titles

All drop-down and cascading menus have a menu title. For drop-down menus, the menu title is the entry that appears in the menu bar. For cascading menus, the menu title is the name of the parent menu item. Menu titles represent the entire menu and should communicate as clearly as possible the purpose of all items on the menu.

Use single words for menu bar menu titles. Multiple word titles or titles with spaces may be indistinguishable from two one-word titles. In addition, avoid uncommon compound words, such as Fontsize.

Define one character of each menu title as its access key. This character provides keyboard access to the menu. Windows displays the access key for a menu title as an underlined character, as shown in Figure 7.4.

For more information about keyboard input and defining access keys, see Chapter 4, "Input Basics."

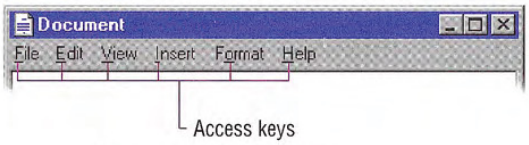


Figure 7.4 Access keys in a menu bar

Define unique access keys for each menu title. Using the same access key for more than one menu title may eliminate direct access to a menu.

Menu Items

Menu items are the individual choices that appear in a menu. Menu items can be text, graphics — such as icons — or graphics and text combinations that represent the actions presented in the menu. The format for a menu item provides the user with visual cues about the nature of the effect it represents, as shown in Figure 7.5.

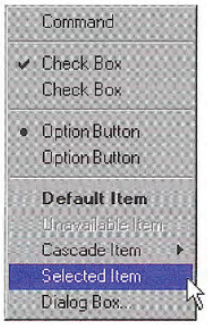


Figure 7.5 Formats for different menu items

Whenever a menu contains a set of related menu items, you can separate those sets with a grouping line known as a *separator*. The standard separator is a single line that spans the width of the menu. Avoid using menu items themselves as group separators, as shown in Figure 7.6.

Figure 7.6 Inappropriate separator

Always provide the user with a visual indication about which menu items can be applied. If a menu item is not appropriate or applicable in a particular context, then disable or remove it. Leaving the menu item enabled and presenting a message box when the user selects the menu item is a poor method for providing feedback.

In general, it is better to disable a menu item rather than remove it because this provides more stability in the interface. However, if the context is such that the menu item is no longer or never relevant, remove it. For example, if a menu displays a set of open files and one of those files is closed or deleted, it is appropriate to remove the corresponding menu item.

If all items in a menu are disabled, disable its menu title. If you disable a menu item or its title, it does not prevent the user from browsing or choosing it. If you provide status bar messages, you can display a message indicating that the command is unavailable and why.

The system provides a standard appearance for displaying disabled menu items. If you are supplying your own visuals for a disabled menu item, follow the visual design guidelines for how to display it with an unavailable appearance.

For more information about displaying commands with an unavailable appearance, see Chapter 13, "Visual Design."

Types of Menu Items

Many menu items take effect as soon as they are chosen. If the menu item is a command that requires additional information to complete its execution, follow the command with an *ellipsis* (...). The ellipsis informs the user that information is incomplete. When used with a command, it indicates that the user needs to provide more information to complete that command. Such commands usually result in the display of a dialog box. For example, the Save As command includes an ellipsis because the command is not complete until the user supplies or confirms a filename.

Not every command that produces a dialog box or other secondary window should include an ellipsis. For example, do not include an ellipsis with the Properties command because carrying out the Properties command displays a properties window. After completing the command, no further parameters or actions are required to fulfill the intent of the command. Similarly, do not include an ellipsis for a command that may result in the display of a message box.

While you can use menu items to carry out commands, you can also use menu items to switch a mode or set a state or property, rather than initiating a process. For example, choosing an item from a menu that contains a list of tools or views implies changing to that state. If the menu item represents a property value, when the user chooses the menu item, the property setting changes.

Menu items for state settings can be independent or interdependent:

- Independent settings are the menu equivalent of check boxes. For example, if a menu contains text properties, such as Bold and Italic, they form a group of independent settings. The user can change each setting without affecting the others, even though they both apply to a single text selection. Include a check mark to the left of an independent setting when that state applies.

- Interdependent settings are the menu equivalent of option buttons. For example, if a menu contains alignment properties — such as Left, Center, and Right — they form a group of interdependent settings. Because a particular paragraph can have only one type of alignment, choosing one resets the property to be the chosen menu item setting. When the user chooses an interdependent setting, place an option button mark to the left of that menu item.

When using the menu to represent the two states of a setting, if those states are obvious opposites, such as the presence or absence of a property value, you can use a check mark to indicate when the setting applies. For example, when reflecting the state of a text selection with a menu item labeled Bold, show a check mark next to the menu item when the text selection is bold and no check mark when it is not. If a selection contains mixed values for the same stat reflected in the menu, you also display the menu without the check mark.

However, if the two states of the setting are not obvious opposites, use a pair of alternating menu item names 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, pair the command with the alternative name Half Duplex, rather using a mark to indicate the alternative states, and consider the following guidelines for how to display those alternatives:

- If there is room in a menu, include both alternatives as individual menu items and interdependent choices. This avoids confusion because the user can view both options simultaneously. You can also use menu separators to group the choices.

- If there is not sufficient room in the menu for the alternative choices, you can use a single menu item and change its name to the alternative action when selected. In this case, the menu item's name does not reflect the current state; it indicates the state after choosing the item. Where possible, define names that use the same access key. For example, the letter D could be used for a menu item that toggles between Full Duplex and Half Duplex.

Avoid defining menu items that change depending on the state of a modifier key. Such techniques hide functionality from a majority of users.

A menu can also have a default item. A default menu item reflects a choice that is also supported through a shortcut technique, such as double-clicking or drag and drop. For example, if the default command for an icon is Open, define this as the default menu item. Similarly, if the default command for a drag and drop operation is Copy, display this command as the default menu item in the pop-up menu that results from a nondefault drag and drop operation (button 2). The system designates a default menu item by displaying its label as bold text.

Menu Item Labels

Include descriptive text or a graphic label for each menu item. Even if you provide a graphic for the label, consider including text as well. The text allows you to provide more direct keyboard access to the user and provides support for a wider range of users.

Use the following guidelines for defining text menu names for menu item labels:

- Define unique item names within a menu. However, item names can be repeated in different menus to represent similar or different actions.
- Use a single word or multiple words, but keep the wording brief and succinct. Verbose menu item names can make it harder for the user to scan the menu.
- Define unique access keys for each menu item within a menu. This provides the user direct keyboard access to the menu item. The guidelines for selecting an access key for menu items are the same as for menu titles, except that the access key for a menu item can also be a number included at the beginning of the menu item name. This is useful for menu items that vary, such as file-names. Where possible, also define consistent access keys for common commands.
- Follow book title capitalization rules for menu item names. For English language versions, capitalize the first letter of every word, except for articles, conjunctions, and prepositions that occur other than at the beginning or end of a multiple-word name. For example, the following menu names are correct: New Folder, Go To, Select All, and Table of Contents.
- Avoid formatting individual menu item names with different text properties. Even though these properties illustrate a particular text style, they also may make the menu cluttered, illegible, or confusing. For example, it may be difficult to indicate an access key if an entire menu entry is underlined.

For more information about defining access keys, see Chapter 4, "Input Basics." For more information about common access key assignments, see Appendix B, "Keyboard Interface Summary."

Shortcut Keys in Menu Items

If you define a keyboard shortcut associated with a command in a drop-down menu, display the shortcut in the menu. Display the shortcut key next to the item and align shortcuts with other shortcuts in the menu. Left align at the first tab position after the longest item in the menu that has a shortcut. Do not use spaces for alignment because they may not display properly in the proportional font used by the system to display menu text or when the font setting menu text changes.

You can match key names with those commonly inscribed on the keycap. Display CTRL and SHIFT key combinations as Ctrl+*key* (rather than Control+*key* or CONTROL+*key* or ^+*key*) and Shift+*key*. When using function keys for menu item shortcuts, display the name of the key as F*n*, where *n* is the function key number.

For more information about the selection of shortcut keys, see Chapter 4, "Input Basics."

Avoid including shortcut keys in pop-up menus. Pop-up menus are already a shortcut form of interaction and are typically accessed with the mouse. In addition, excluding shortcut keys makes pop-up menus easier for users to scan.

Controls

Controls are graphic objects that represent the properties or operations of other objects. Some controls display and allow editing of particular values. Other controls start an associated command.

Each control has a unique appearance and operation designed for a specific form of interaction. The system also provides support for designing your own controls. When defining your own controls, follow the conventions consistent with those provided by the system-supplied controls.

For more information about using standard controls and designing your own controls, see Chapter 13, "Visual Design."

Like most elements of the interface, controls provide feedback indicating when they have the input focus and when they are activated. For example, when the user interacts with controls using a mouse, each control indicates its selection upon the down transition of the mouse button, but does not activate until the user releases the button, unless the control supports auto-repeat.

Controls are generally interactive only when the pointer, actually the hot spot of the pointer, is over the control. If the user moves the pointer off the control while pressing a mouse button, the control no longer responds to the input device. If the user moves the pointer back onto the control, it once again responds to the input device. The hot zone, or boundary that defines whether a control responds to the pointer, depends on the type of control. For some controls, such as buttons, the hot zone coincides with the visible border of the control. For others, the hot zone may include the control's graphic and label (for example, check boxes) or some controls, such as scroll bars, as defined around the control's borders.

Many controls provide labels. Because labels help identify the purpose of a control, always label a control with which you want the user to directly interact. If a control does not have a label, you can provide a label using a static text field or a tooltip control. Define an access key for text labels to provide the user direct keyboard access to a control. Where possible, define consistent access keys for common commands.

For more information about defining access keys, see Chapter 4, "Input Basics."

While controls provide specific interfaces for user interaction, you can also include pop-up menus for controls. This can provide an effective way to transfer the value the control represents or to provide access to context-sensitive Help information. The interface to pop-up menus for controls follows the standard conventions for pop-up menus, except that it does not affect the state of the control; that is, clicking the control with button 2 does not trigger the action associated with the control when the user clicks it with button 1. The only action is the display of the pop-up menu.

A pop-up menu for a control is contextual to what the control represents, rather than the control itself. Therefore, avoid commands such as Set, Unset, Check, or Uncheck. The exception is in a forms design or window layout context, where the commands on the pop-up menu can apply to the control itself.

Buttons

Buttons are controls that start actions or change properties. There are three basic types of buttons: command buttons, option buttons, and check boxes.

Command Buttons

A *command button*, also referred to as a push button, is a control, commonly rectangular in shape, that includes a label (text, graphic, or sometimes both), as shown in Figure 7.7.

Figure 7.7 Command buttons

When the user chooses a command button with mouse button 1 (for pens, tapping), the command associated with the button is carried out. When the user presses the mouse button, the input focus moves to the button, and the button state changes to its pressed appearance. If the user moves the pointer off the command button while the mouse button remains pressed, the button returns to its original state. Moving the pointer back over the button while pressing the mouse button returns the button to its pressed state.

When the user releases the mouse button with the pointer on the command button, the command associated with the control starts. If the pointer is not on the control when the user releases the mouse button, no action occurs.

You can define access keys and shortcut keys for command buttons. In addition, you can use the TAB key and arrow keys to support user navigation to or between command buttons. The SPACEBAR activates a command button if the user moves the input focus to the button.

For more information about navigation and activation of controls, see Chapter 8, "Secondary Windows."

The effect of choosing a button is immediate with respect to its context. For example, in toolbars, clicking a button carries out the associated action. In a secondary window, such as a dialog box, activating a button may initiate a transaction within the window, or apply a transaction and close the window.

The command button's label represents the action the button starts. When using a text label, the text should follow the same capitalization conventions defined for menus. If the control is disabled, display the label of the button as unavailable.

Include an ellipsis (...) as a visual cue for buttons associated with commands that require additional information. Like menu items, the use of an ellipsis indicates that further information is needed, not simply that a window will appear. Some buttons, when clicked, can display a message box, but this does not imply that the command button's label should include an ellipsis.

You can use command buttons to enlarge a secondary window and display additional options, also known as an *unfold button*. An unfold button is not really a different type of control, but the use of a command button for this specific function. When using a command button for this purpose, include a pair of "greater than" (>>) characters as part of the button's label.

In some cases, a command button can represent an object and its default action. For example, the taskbar buttons represent an object's primary window and the Restore command. When the user clicks on the button with mouse button 1, the default command of the object is carried out. Clicking on a button with mouse button 2 displays a pop-up menu for the object the button represents.

You can also use command buttons to reflect a mode or property value similar to the use of option buttons or check boxes. While the typical interaction for a command button is to return to its normal "up" state, if you use it to represent a state, display the button in the option-set appearance, as shown in Table 7.4.

For more information about the appearance of different states of buttons, see Chapter 13, "Visual Design."

Table 7.4 Command Button Appearance

Appearance	Button state
	Normal appearance
	Pressed appearance
	Option-set appearance
	Unavailable appearance
	Option-set, unavailable appearance
	Mixed-value appearance
	Input focus appearance

You can also use command buttons to set tool modes — for example, in drawing or forms design programs for drawing out specific shapes or controls. In this case, design the button labels to reflect the tool's use. When the user chooses the tool (that is, clicks the button), display the button using the option-set appearance and change the pointer to indicate the change of the mode of interaction.

You can also use a command button to display a pop-up menu. This convention is known as a *menu button*. While this is not a specific control provided by the system, you can create this interface using the standard components.

A menu button looks just like a standard command button, except that, as a part of its label, it includes a triangular arrow similar to the one found in cascading menu titles, as shown in Figure 7.8.

Figure 7.8 A menu button

A menu button supports the same type of interaction as a drop-down menu; the menu is displayed when the button is pressed and allows the user to drag into the menu from the button and make menu selections. Like any other menu, use highlighting to track the movement of the pointer.

Similarly, when the user clicks a menu button, the menu is displayed. At this point, interaction with the menu is the same as with any menu. For example, clicking a menu item carries out the associated command. Clicking outside the menu or on the menu button removes the menu.

When pressed, display the menu button with the pressed appearance. When the user releases the mouse button and the menu is displayed, use the option-set appearance. Otherwise, the menu button's appearance is the same as a typical command button. For example, if the button is disabled, display the button using the unavailable appearance.

Option Buttons

An *option button*, also referred to as a radio button, represents a single choice within a limited set of mutually exclusive choices — that is, in any group of option buttons, only one option in the group can be set. Accordingly, always group option buttons in sets of two or more, as shown in Figure 7.9.

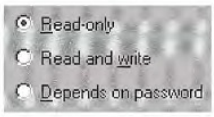


Figure 7.9 A set of option buttons

Option buttons appear as a set of small circles. When an option button choice is set, a dot appears in the middle of the circle. When the choice is not the current setting, the circle is empty. Avoid using option buttons to start an action other than the setting of a particular option or value represented by the option button. The only exception is that you can support double-clicking the option button as a shortcut for setting the value and carrying out the default command of the window in which the option buttons appear, if choosing an option button is the primary user action for the window.

You can use option buttons to represent a set of choices for a particular property. When the option buttons reflect a selection with mixed values for that property, display all the buttons in the group using the mixed-value appearance to indicate that multiple values exist for that property. The mixed-value appearance for a group of option buttons displays all buttons without a setting dot, as shown in Figure 7.10.

Figure 7.10 Option buttons with mixed-value appearance

If the user chooses any option button in a group with mixed-value appearance, that value becomes the setting for the group; the dot appears in that button and all the other buttons in the group remain empty.

Limit the use of option buttons to small sets of options, typically seven or less, but always at least two. If you need more choices, consider using another control, such as a single selection list box or drop-down list box.

Each option button includes a text label. (If you need graphic labels for a group of exclusive choices, consider using command buttons instead.) The standard control allows you to include multiple line labels. When implementing multiple line labels, use top alignment, unless the context requires an alternate orientation.

Define the option button's label to represent the value or effect for that choice. Also use the label to indicate when the choice is unavailable. Use sentence capitalization for an option button's label; only capitalize the first letter of the first word, unless it is a word in the label normally capitalized.

For more information about labeling or appearance states, see Chapter 13, "Visual Design."

Because option buttons appear as a group, you can use a group box control to visually define the group. You can label the option buttons to be relative to a group box's label. For example, for a group box labeled Alignment, you can label the option buttons as Left, Right, and Center.

As with command buttons, the mouse interface for choosing an option button uses a click with mouse button 1 (for pens, tapping) either on the button's circle or on the button's label. The input focus is moved to the option button's label when the user presses the mouse button, and the option button displays its pressed appearance. If the user moves the pointer off the option button before releasing the

mouse button, the option button is returned to its original state. The option is not set until the user releases the mouse button while the pointer is over the control. Releasing the mouse button outside of the option button or its label has no effect on the current setting of the option button. In addition, successive mouse clicks on the same option button do not toggle the button's state; the user needs to explicitly select an alternative choice in the group to change or restore a former choice.

Assign access keys to option button labels to provide a keyboard interface to the buttons. You can also define the TAB or arrow keys to allow the user to navigate and choose a button. Access keys or arrow keys automatically set an option button and set the input focus to that button.

For more information about the guidelines for defining access keys, see Chapter 4, "Input Basics." For more information about navigation and interaction with option buttons, see Chapter 8, "Secondary Windows."

Check Boxes

Like option buttons, check boxes support options that are either on or off; check boxes differ from option buttons in that you typically use check boxes for independent or nonexclusive choices. As in the case of independent settings in menus, use check boxes only when both states of the choice are clearly opposite and unambiguous. If this is not the case, then use option buttons or some other form of single selection choice control instead.

A check box appears as a square box with an accompanying label. When the choice is set, a check mark appears in the box. When the choice is not set, the check box is empty, as shown in Figure 7.11.

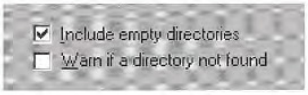


Figure 7.11 A set of check boxes

A check box's label is typically displayed as text and the standard control includes a label. (Use a command button instead of a check box when you need a nonexclusive choice with a graphic label.) Use a single line of text for the label as this makes the label easier to read. However, if you do use multiple lines, use top alignment, unless the context requires a different orientation.

Define a check box's label to appropriately express the value or effect of the choice. Use sentence capitalization for multiple word labels. The label also serves as an indication of when the control is unavailable.

Group related check box choices. If you group check boxes, it does not prevent the user from setting the check boxes on or off in any combination. While each check box's setting is typically independent of the others, you can use a check box's setting to affect other controls. For example, you can use the state of a check box to filter the content of a list. If you have a large number of choices or if the number of choices varies, use a multiple selection list box instead of check boxes.

When the user clicks a check box with mouse button 1 (for pens, tapping) either on the check box square or on the check box's label, that button is chosen and its state is toggled. When the user presses the mouse button, the input focus moves to the control and the check box assumes its pressed appearance. Like option buttons and other controls, if the user moves the pointer off the control while holding down the mouse button, the control's appearance returns to its original state. The setting state of the check box does not change until the mouse button is released. To change the control's setting, the pointer must be over the check box or its label when the user releases the mouse button.

Define access keys for check box labels to provide a keyboard interface for navigating to and choosing a check box. In addition, the TAB key and arrow keys can also be supported to provide user navigation to or between check boxes. In a dialog box, for example, the SPACEBAR toggles a check box when the input focus is on the check box.

For more information about guidelines for defining access keys, see Chapter 4, "Input Basics." For more information about navigation and supporting interaction for controls with the keyboard, see Chapter 8, "Secondary Windows."

If you use a check box to display the value for the property of a multiple selection whose values for that property differ (for example, for a text selection that is partly bold), display the check box in its mixed-value appearance, as shown in Figure 7.12.

Figure 7.12 A mixed-value check box (magnified)

If the user chooses a check box in the mixed-value state, the associated value is set and a check mark is placed in it. This implies that the property of all elements in the multiple selection will be set to this value when it is applied. If the user chooses the check box again, the setting to be unchecked is toggled. If applied to the selection, the value will not be set. If the user chooses the check box a third time, the value is toggled back to the mixed-value state. When the user applies the value, all elements in the selection retain their original value. This three-state toggling occurs only when the control represents a mixed set of values.

List Boxes

A *list box* is a convenient, preconstructed control for displaying a list of choices for the user. The choices can be text, color, icons, or other graphics. The purpose of a list box is to display a collection of items and, in most cases, support selection of a choice of an item or items in the list.

List boxes are best for displaying large numbers of choices that vary in number or content. If a particular choice is not available, omit the choice from the list. For example, if a point size is not available for the currently selected font, do not display that size in the list.

Order entries in a list using the most appropriate choice to represent the content in the list and to facilitate easy user browsing. For example, alphabetize a list of names, but put a list of dates in chronological order. If there is no natural or logical ordering for the content, use ascending or alphabetical ordering — for example, 0–9 or A–Z.

List box controls do not include their own labels. However, you can include a label using a static text field; the label enables you to provide a descriptive reference for the control and keyboard access to the control. Use sentence capitalization for multiple word labels and make certain that your support for keyboard access moves the input focus to the list box and not the static text field label.

For more information about navigation to controls in a secondary window, see Chapter 8, "Secondary Windows." For more information about defining access keys for control labels, see Chapter 4, "Input Basics." For more information about static text fields, see the section, "Static Text Fields," later in this chapter.

When a list box is disabled, display its label using an unavailable appearance. If possible, display all of the entries in the list as unavailable to avoid confusing the user as to whether the control is enabled or not.