The width of the list box should be sufficient to display the average width of an entry in the list. If that is not practical because of space or the variability of what the list might include, consider one or more of the following options:

- Make the list box wide enough to allow the entries in the list to be sufficiently distinguished.

- Use an ellipsis (...) in the middle or at the end of long text entries to shorten them, while preserving the important characteristics needed to distinguish them. For example, for long paths, usually the beginning and end of the path are the most critical; you can use an ellipsis to shorten the entire name: \Sample\...\Example.

- Include a horizontal scroll bar. However, this option reduces some usability, because adding the scroll bar reduces the number of entries the user can view at one time. In addition, if most entries in the list box do not need to be horizontally scrolled, including a horizontal scroll bar accommodates the infrequent case.

When the user clicks an item in a list box, it becomes selected. Support for multiple selection depends on the type of list box you use. List boxes also include scroll bars when the number of items in the list exceeds the visible area of the control.

Arrow keys also provide support for selection and scrolling a list box. In addition, list boxes include support for keyboard selection using text keys. When the user presses a text key, the list navigates and selects the matching item in the list, scrolling the list if necessary to keep the user's selection visible. Subsequent key presses continue the matching process. Some list boxes support sequential matches based on timing; each time the user presses a key, the control matches the next character in a word if the user presses the key within the system's time-out setting. If the time-out elapses, the control is reset to matching based on the first character. Other list box controls, such as combo boxes and drop-down combo boxes, do sequential character matching based on the characters typed into the text box component of the control. These controls may be preferable because they do not require the user to master the timing sequence. However, they do take up more space and potentially allow the user to type in entries that do not exist in the list box.

When the list is scrolled to the beginning or end of data, disable the corresponding scroll bar arrow button. If all items in the list are visible, disable both scroll arrows. If the list box never includes more items that can be shown in the list box, so that the user will not need to scroll the list, you may remove the scroll bar.

When incorporating a list box into a window's design, consider supporting both command (Cut, Copy, and Paste) and direct manipulation (drag and drop) transfers for the list box. For example, if the list displays icons or values that the user can move or copy to other locations, such as another list box, support transfer operations for the list. The list view control automatically supports this; however, the system provides support for you to enable this for other list boxes as well.

For more information about disabling scroll bar arrows, see Chapter 6, "Windows."

List boxes can be classified by how they display a list and by the type of selection they support.

Single Selection List Boxes

A *single selection list box* is designed for the selection of only one item in a list. Therefore, the control provides a mutually exclusive operation similar to a group of option buttons, except that a list box can more efficiently handle a large number of items.

Define a single selection list box to be tall enough to show at least three to eight choices, as shown in Figure 7.13 — depending on the design constraints of where the list box is used. Always include a vertical scroll bar. If all the items in the list are visible, then follow the window scroll bar guidelines for disabling the scroll arrows and enlarging the scroll box to fill the scroll bar shaft.

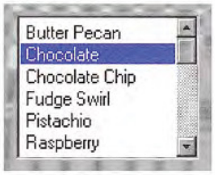


Figure 7.13 A single selection list box

The currently selected item in a single selection list box is highlighted using selection appearance.

The user can select an entry in a single selection list box by clicking on it with mouse button 1 (for pens, tapping). This also sets the input focus on that item in the list. Because this type of list box supports only single selection, when the user chooses another entry any other selected item in the list becomes unselected. The scroll bar in the list box allows the mouse user to scroll through the list of entries, following the interaction defined for scroll bars.

For more information about the interaction techniques of scroll bars, see Chapter 6, "Windows."

The keyboard interface uses navigation keys, such as the arrow keys, HOME, END, PAGE UP, and PAGE DOWN. It also uses text keys, with matches based on timing; for example, when the user presses a text key, an entry matching that character scrolls to the top of the list and becomes selected. These keys not only navigate to an entry in the list, but also select it. If no item in the list is currently selected, when the user chooses a list navigation key, the first item in the list that corresponds to that key is selected. For example, if the user presses the DOWN ARROW key, the first entry in the list is selected, instead of navigating to the second item in the list.

If the choices in the list box represent values for the property of a selection, then make the current value visible and highlighted when displaying the list. If the list box reflects mixed values for a multiple selection, then no entry in the list should be selected.

Drop-down List Boxes

Like a single selection list box, a *drop-down list box* provides for the selection of a single item from a list of items; the difference is that the list is displayed upon demand. In its closed state, the control displays the current value for the control. The user opens the list to change the value. Figure 7.14 shows the drop-down list box in its closed and opened state.

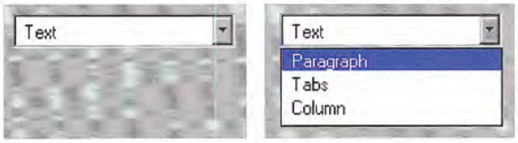


Figure 7.14 A drop-down list box (closed and opened state)

While drop-down list boxes are an effective way to conserve space and reduce clutter, they require more user interaction for browsing and selecting an item than a single selection list box.

Make the width of a closed drop-down list box a few spaces larger than the average width of the items in its list. The open list component of the control should be tall enough to show three to eight items, following the same conventions of a single selection list box. The width of the list should be wide enough not only to display the choices in the list, but also to allow the user to drag directly into the list.

The interface for drop-down list boxes is similar to that for menus. For example, the user can press the mouse button on the current setting portion of the control or on the control's menu button to display the list. Choosing an item in the list automatically closes the list.

If the user navigates to the control using an access key, the TAB key or arrow keys, an UP ARROW or DOWN ARROW, or ALT+UP ARROW or ALT+DOWN ARROW displays the list. Arrow keys or text keys navigate and select items in the list. If the user presses ALT+UP ARROW, ALT+DOWN ARROW, a navigation key, or an access key to move to another control, the list automatically closes. When the list is closed, preserve any selection made while the list was open. The ESC key also closes the list.

If the choices in a drop-down list represent values for the property of a multiple selection and the values for that property are mixed, then display no value in the current setting component of the control.

Extended and Multiple Selection List Boxes

Although most list boxes are single selection lists, some contexts require the user to choose more than one item. *Extended selection list boxes* and *multiple selection list boxes* support this functionality.

Extended and multiple selection list boxes follow the same conventions for height and width as single selection list boxes. The height should display no less than three items and generally no more than eight, unless the size of the list varies with the size of the window. Base the width of the box on the average width of the entries in the list.

Extended selection list boxes support conventional navigation, and contiguous and disjoint selection techniques. That is, extended selection list boxes are optimized for selecting a single item or a single range, while still providing for disjoint selections.

For more information about contiguous and disjoint selection techniques, see Chapter 5, "General Interaction Techniques."

When you want to support user selection of several disjoint entries from a list, but an extended selection list box is too cumbersome, you can define a multiple selection list box. Whereas extended selection list boxes are optimized for individual item or range selection, multiple selection list boxes are optimized for independent selection. However, because simple multiple selection list boxes are not visually distinct from extended selection list boxes, consider designing them to appear similar to a scrollable list of check boxes, as shown in Figure 7.15. This requires providing your own graphics for the items in the list (using the owner-drawn list box style). This appearance helps the user to distinguish the difference in the interface of the list box with a familiar convention. It also serves to differentiate keyboard navigation from the state of a choice. Because the check box controls are nested, you use the flat appearance style for the check boxes. You may also create this kind of a list box using a list view control.

For more information about the flat appearance style for controls in a list box, see Chapter 13, "Visual Design."

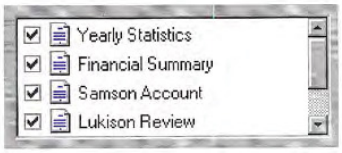


Figure 7.15 A multiple selection list box

List View Controls

A *list view control* is a special extended selection list box that displays a collection of items, each item consisting of an icon and a label. List view controls can display content in four different views.

View	Description
Icon	Each item appears as a full-sized icon with a label below it. The user can drag the icons to any location within the view.
Small Icon	Each item appears as a small icon with its label to the right. The user can drag the icons to any location within the view.
List	Each item appears as a small icon with its label to the right. The icons appear in a columnar, sorted layout.
Report	Each item appears as a line in a multicolumn format with the leftmost column including the icon and its label. The subsequent columns contain information supplied by the application displaying the list view control.

The control also supports options for alignment of icons, selection of icons, sorting of icons, and editing of the icon's labels. It also supports drag and drop interaction.

Use this control where the representation of objects as icons is appropriate. In addition, provide pop-up menus on the icons displayed in the views. This provides a consistent paradigm for how the user interacts with icons elsewhere in the Windows interface.

Selection and navigation in this control work similarly to that in folder windows. For example, clicking on an icon selects it. After selecting the icon, the user can use extended selection techniques, including region selection, for contiguous or disjoint selections. Arrow keys and text keys (time-out based matching) support keyboard navigation and selection.

As an option, the standard control also supports the display of graphics that can be used to represent state information. For example, you can use this functionality to include check boxes next to items in a list.

Tree View Controls

A *tree view control* is a special list box control that displays a set of objects as an indented outline based on their logical hierarchical relationship. The control includes buttons that allow the outline to be expanded and collapsed, as shown in Figure 7.16. You can use a tree view control to display the relationship between a set of containers or other hierarchical elements.

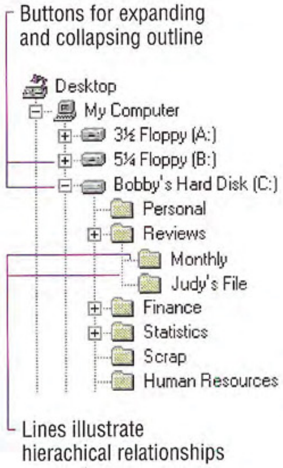


Figure 7.16 A tree view control

You can optionally include icons with the text label of each item in the tree. Different icons can be displayed when the user expands or collapses the item in the tree. In addition, you can also include a graphic, such as a check box, that can be used to reflect state information about the item.

The control also supports drawing lines that define the hierarchical relationship of the items in the list and buttons for expanding and collapsing the outline. It is best to include these features (even though they are optional) because they make it easier for the user to interpret the outline.

Arrow keys provide keyboard support for navigation through the control; the user presses UP ARROW and DOWN ARROW to move between items and LEFT ARROW and RIGHT ARROW to move along a particular branch of the outline. Pressing RIGHT ARROW can also expand the outline at a branch if it is not currently displayed. Text keys can also be used to navigate and select items in the list, using the matching technique based on timing.

When you use this control in a dialog box, if you use the ENTER key or use double-clicking to carry out the default command for an item in the list, make certain that the default command button in your dialog box matches. For example, if you use double-clicking an entry in the outline to display the item's properties, then define a Properties button to be the default command button in the dialog box when the tree view control has the input focus.

Text Fields

Windows includes a number of controls that facilitate the display, entry, or editing of a text value. Some of these controls combine a basic text-entry field with other types of controls.

Text fields do not include labels as a part of the control. However, you can add one using a static text field. Including a label helps identify the purpose of a text field and provides a means of indicating when the field is disabled. Use sentence capitalization for multiple word labels. You can also define access keys for the text label to provide keyboard access to the text field. When using a static text label, define keyboard access to move the input focus to the text field with which the label is associated rather than the static text field itself. You can also support keyboard navigation to text fields by using the TAB key (and, optionally, arrow keys).

For more information about static text fields, see the section, "Static Text Fields," later in this chapter.

When using a text field for input of a restricted set of possible values, for example, a field where only numbers are appropriate, validate user input immediately, either by ignoring inappropriate characters or by providing feedback indicating that the value is invalid or both.

For more information about validation of input, see Chapter 8, "Secondary Windows."

Text Boxes

A *text box* (also referred to as an edit control) is a rectangular control where the user enters or edits text, as shown in Figure 7.17. It can be defined to support a single line or multiple lines of text. The outline border of the control is optional, although the border is typically included when displaying the control in a toolbar or a secondary window.

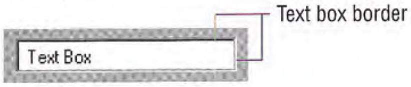


Figure 7.17 A standard text box

The standard text box control provides basic text input and editing support. Editing includes the insertion or deletion of characters and the option of text wrapping. Although individual font or paragraph properties are not supported, the entire control can support a specific font setting.

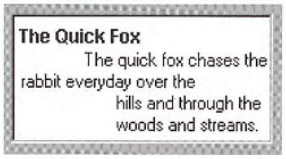
You can also use text boxes to display read-only text that is not editable, but still selectable. When setting this option with the standard control, the system automatically changes the background color of the field to indicate to the user the difference in behavior.

A text box supports standard interactive techniques for navigation and contiguous selection. Horizontal scrolling is available for single line text boxes, and horizontal and vertical scroll bars are supported for multiple line text boxes.

You can limit the number of characters accepted as input for a text box to whatever is appropriate for the context. In addition, you can support *auto-exit* for text boxes defined for fixed-length input; that is, as soon as the last character is typed in the text box, the focus moves to the next control. For example, you can define a five-character auto-exit text box to facilitate the entry of zip code, or three two-character auto-exit text boxes to support the entry of a date. Use auto-exit text boxes sparingly; the automatic shift of focus can surprise the user. They are best limited to situations involving extensive data entry.

Rich-Text Boxes

A *rich-text box*, as shown in Figure 7.18, provides the same basic text editing support as a standard text box. In addition, a rich-text box supports font properties, such as typeface, size, color, bold, and italic format, for each character and paragraph format property, such as alignment, tabs, indents, and numbering. The control also supports printing of its content and embedding of OLE objects.



The Quick Fox

The quick fox chases the rabbit everyday over the hills and through the woods and streams.

Figure 7.18 A rich-text box

Combo Boxes

A *combo box* is a control that combines a text box with a list box, as shown in Figure 7.19. This allows the user to type in an entry or choose one from the list.

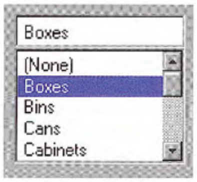


Figure 7.19 A combo box

The text box and its associated list box have a dependent relationship. As text is typed into the text box, the list scrolls to the nearest match. In addition, when the user selects an item in the list box, it automatically uses that entry to replace the content of the text box and selects the text.

The interface for the control follows the conventions supported for each component, except that the UP ARROW and DOWN ARROW keys move only in the list box. LEFT ARROW and RIGHT ARROW keys operate solely in the text box.

Drop-down Combo Boxes

A *drop-down combo box*, as shown in Figure 7.20, combines the characteristics of a text box with a drop-down list box. A drop-down combo box is more compact than a regular combo box; it can be used to conserve space, but requires additional user interaction required to display the list.

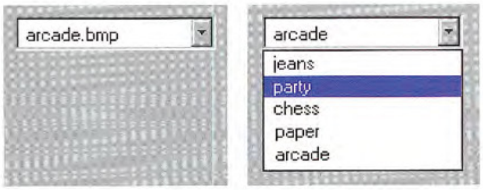


Figure 7.20 A drop-down combo box (closed and opened state)

The closed state of a drop-down combo box is similar to that of a drop-down list, except that the text box is interactive. When the user clicks the control's menu button the list is opened. Clicking the menu button a second time, choosing an item in the list, or clicking another control closes the list.

Provide a static text field label for the control and assign an access key. Use the access key so the user can navigate to the control. You can also support the TAB key or arrow keys for navigation to the control. When the control has the input focus, when the user presses the UP ARROW or DOWN ARROW or ALT+UP ARROW or ALT+DOWN ARROW key, the list is displayed.

When the control has the input focus, pressing a navigation key, such as the TAB key, or an access key or ALT+UP ARROW or ALT+DOWN ARROW to navigate to another control closes the list. When the list is closed, preserve any selection made while the list was open, unless the user presses a Cancel command button. The ESC key also closes the list.

When the list is displayed, the interdependent relationship between the text box and list is the same as it is for standard combo boxes when the user types text into the text box. When the user chooses an item in the list, the interaction is the same as for drop-down lists — the selected item becomes the entry in the text box.

Spin Boxes

Spin boxes are text boxes that accept a limited set of discrete ordered input values that make up a circular loop. A spin box is a combination of a text box and a special control that incorporates a pair of buttons (also known as an up-down control), as shown in Figure 7.21.

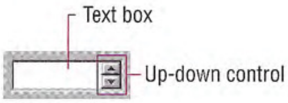


Figure 7.21 A spin box

When the user clicks on the text box or the buttons, the input focus is set to the text box component of the control. The user can type a text value directly into the control or use the buttons to increment or decrement the value. The unit of change depends on what you define the control to represent.

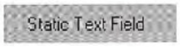
Use caution when using the control in situations where the meaning of the buttons may be ambiguous. For example, with numeric values, such as dates, it may not be clear whether the top button increments the date or changes to the previous date. Define the top button to increase the value by one unit and the bottom button to decrease the value by one unit. Typically, wrap around at either end of the set of values. You may need to provide some additional information to communicate how the buttons apply.

By including a static text field as a label for the spin box and defining an associated access key, you can provide direct keyboard access to the control. You can also support keyboard access using the TAB key (or, optionally, arrow keys). Once the control has the input focus, the user can change the value by pressing UP ARROW or DOWN ARROW.

You can also use a single set of spin box buttons to edit a sequence of related text boxes, for example, time as expressed in hours, minutes, and seconds. The buttons affect only the text box that currently has the input focus.

Static Text Fields

You can use static text fields to present read-only text information. Unlike read-only text box controls, the text is not selectable. However, your application can still alter read-only static text to reflect a change in state. For example, you can use static text to display the current directory path or the status information, such as page number, key states, or time and date. Figure 7.22 illustrates a static text field.



Static Text Field

Figure 7.22 A static text field

You can also use static text fields to provide labels or descriptive information for other controls. Using static text fields as labels for other controls allows you to provide access-key activation for the control with which it is associated. Make certain that the input focus moves to its associated control and not to the static field. Also remember to include a colon at the end of the text. Not only does this help communicate that the text represents the label for a control, it is also used by screen review utilities.

For more information about the layout of static text fields, see Chapter 13, "Visual Design." For information about the use of static text fields as labels and screen review utilities, see Chapter 14, "Special Design Considerations."

Shortcut Key Input Controls

A *shortcut key input control* (also known as a hot key control) is a special kind of text box to support user input of a key or key combination to define a shortcut key assignment. Use it when you provide an interface for the user to customize shortcut keys supported by your application. Because shortcut keys carry out a command directly, they provide a more efficient interface for common or frequently used actions.

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

The control allows you to define invalid keys or key combinations to ensure valid user input; the control will only access valid keys. You also supply a default modifier to use when the user enters an invalid key. The control displays the valid key or key combination including any modifier keys.

When the user clicks a shortcut key input control, the input focus is set to the control. Like most text boxes, the control does not include its own label, so use a static text field to provide a label and assign an appropriate access key. You can also support the TAB key to provide keyboard access to the control.

Other General Controls

The system also provides support for controls designed to organize other controls and controls for special types of interfaces.

Group Boxes

A *group box* is a special control you can use to organize a set of controls. A group box is a rectangular frame with an optional label that surrounds a set of controls, as shown in Figure 7.23. Group boxes generally do not directly process any input. However, you can provide navigational access to items in the group using the TAB key or by assigning an access key to the group label.

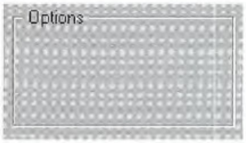
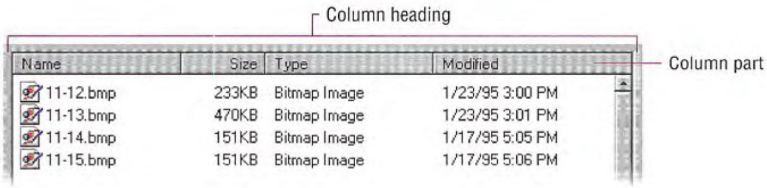


Figure 7.23 A group box

You can make the label for controls that you place in a group box relative to the group box's label. For example, a group labeled Alignment can have option buttons labeled Left, Right, and Center. Use sentence capitalization for a multiple word label.

Column Headings

Using a *column heading* control, also known as a header control, you can display a heading above columns of text or numbers. You can divide the control into two or more parts to provide headings for multiple columns, as shown in Figure 7.24. The list view control also provides support for a column heading control.



Name	Size	Type	Modified
11-12.bmp	233KB	Bitmap Image	1/23/95 3:00 PM
11-13.bmp	470KB	Bitmap Image	1/23/95 3:01 PM
11-14.bmp	151KB	Bitmap Image	1/17/95 5:05 PM
11-15.bmp	151KB	Bitmap Image	1/17/95 5:06 PM

Figure 7.24 A column heading divided into four parts

Each header part label can include text and a graphic image. Use the graphic image to show information such as the sort direction. You can align the title elements left, right, or centered.

You can configure each part to behave like a command button to support a specific function when the user clicks on it. For example, consider supporting sorting the list by clicking on a particular header part. Also, you can support clicking on the part with button 2 to display a pop-up menu containing specific commands, such as Sort Ascending and Sort Descending.

The control also supports the user dragging on the divisions that separate header parts to set the width of each column. As an option, you can support double-clicking on a division as a shortcut to a command that applies to formatting the column, such as automatically sizing the column to the largest value in that column.

Tabs

A *tab* control is analogous to a divider in a file cabinet or notebook, as shown in Figure 7.25. You can use this control to define multiple logical pages or sections of information within the same window.

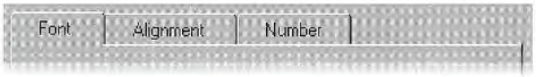


Figure 7.25 A tab control

Tab labels can include text or graphic information, or both. Usually, the control automatically sizes the tab to the size of its label; however, you can define your tabs to have a fixed width. Use the system font for the text labels of your tabs and use the same capitalization for multiple word labels as you use for menus and command buttons (in English versions, book title capitalization). If you use only graphics as your tab label, support tooltips for your tabs.

By default, a tab control displays only one row of tabs. While the control supports multiple rows or scrolling a single row of tabs, avoid these alternatives because they add complexity to the interface by making it harder to read and access a particular tab. You may want to consider alternatives such as separating the tabbed pages into sets and using another control to move between the sets. However, if scrolling the tabs seems appropriate, follow the conventions documented in this guide.

When the user clicks a tab with mouse button 1, the input focus moves and switches to that tab. When a tab has the input focus, LEFT ARROW or RIGHT ARROW keys move between tabs. CTRL+TAB also switches between tabs. Optionally, you can also define access keys for navigating between tabs. If the user switches pages using the tab, you can place the input focus on the particular control on that page. If there is no appropriate control or field in which to place the tab, leave the input focus on the tab itself.

Property Sheet Controls

A *property sheet control* provides the basic framework for defining a property sheet. It provides the common controls used in a property sheet and accepts modeless dialog box layout definitions to automatically create tabbed property pages.

The property sheet control also includes support for creating wizards. Wizards are a special form of user assistance that guide the user through a sequence of steps in a specific operation or process. When using the control as a wizard, tabs are not included, and the standard OK, Cancel, and Apply buttons are replaced with a Back, Next, or Finish button, and a Cancel button.

For more information about property sheets, see Chapter 8, "Secondary Windows." For more information about wizards, see Chapter 12, "User Assistance."

Scroll Bars

Scroll bars are horizontal or vertical scrolling controls you can use to create scrollable areas other than on the window frame or list box where they can be automatically included. Use scroll bar controls only for supporting scrolling contexts. For contexts where you want to provide an interface for setting or adjusting values, use a slider or other control, such as a spin box. Because scroll bars are designed for scrolling information, using a scroll bar to set values may confuse the user as to the purpose or interaction of the control.

When using scroll bar controls, follow the recommended conventions for disabling the scroll bar arrows. Disable a scroll bar arrow button when the user scrolls the information to the beginning or end of the data, unless the structure permits the user to scroll beyond the data. For more information about scroll bar conventions, see Chapter 6, "Windows."

While scroll bar controls can support the input focus, avoid defining this type of interface. Instead, define the keyboard interface of your scrollable area so that it can scroll without requiring the user to move the input focus to a scroll bar. This makes your scrolling interface more consistent with the user interaction for window and list box scroll bars.

Sliders

Use a slider for setting or adjusting values on a continuous range of values, such as volume or brightness. A *slider* is a control, sometimes called a trackbar control, that consists of a bar that defines the extent or range of the adjustment, and an indicator that both shows the current value for the control and provides the means for changing the value, as shown in Figure 7.26.

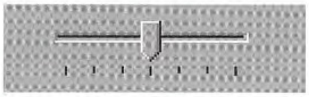


Figure 7.26 A slider

Because a slider does not include its own label, use a static text field to create one. You can also add text and graphics to the control to help the user interpret the scale and range of the control.

Sliders support a number of options. You can set the slider orientation as vertical or horizontal, define the length and height of the slide indicator and the slide bar component, define the increments of the slider, and whether to display tick marks for the control.

The user moves the slide indicator by dragging to a particular location or clicking in the hot zone area of the bar, which moves the slide indicator directly to that location. To provide keyboard interaction, support the TAB key and define an access key for the static text field you use for its label. When the control has the input focus, arrow keys can be used to move the slide indicator in the respective direction represented by the key.

Progress Indicators

A *progress indicator* is a control, also known as a progress bar control, you can use to show the percentage of completion of a lengthy operation. It consists of a rectangular bar that "fills" from left to right, as shown in Figure 7.27.

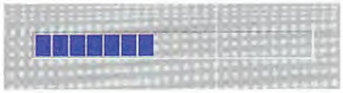


Figure 7.27 A progress indicator

Because a progress indicator only displays information, it is typically noninteractive. However, it may be useful to add static text or other information to help communicate the purpose of the progress indicator. If you do include text, place it outside of the progress indicator control.

Use the control as feedback for long operations or background processes as a supplement to changing the pointer. The control provides more visual feedback to the user about the progress of the process. You can also use the control to reflect the progression of a background process, leaving the pointer's image to reflect interactivity for foreground activities. When determining whether to use a progress indicator in message box or status bar, consider how modal the operation or process the progress indicator represents.

For more information about message boxes, see Chapter 9, "Secondary Windows." For more information about status bars, see the section, "Toolbars and Status Bars," later in this chapter.

Tooltip Controls

A tooltip control provides the basic functionality of a tooltip. A tooltip is a small pop-up window that includes descriptive text displayed when the user moves the pointer over a control, as shown in Figure 7.28. The tooltip appears after a short time-out and is automatically removed when the user clicks the control or moves the pointer off the control.

For more information about the use of tooltips, see Chapter 12, "User Assistance." For more information about the use of tooltips in toolbars, see the section, "Toolbars and Status Bars," later in this chapter.

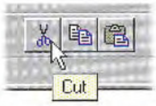


Figure 7.28 A tooltip control

The system displays a tooltip control at the lower right of the pointer, but automatically adjusts the tooltip to avoid displaying it offscreen. However for text boxes, the tooltip should be displayed centered under the control it identifies. The control supports an option to support this behavior.

Wells

A *well* is a special field similar to a group of option buttons, but facilitates user selection of graphic values such as a color, pattern, or images, as shown in Figure 7.29. This control is not currently provided by the system; however, its purpose and interaction guidelines are described here to provide a consistent interface.

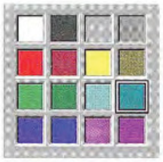


Figure 7.29 A well control for selection colors

Like option buttons, use well controls for values that have two or more choices and group the choices to form a logical arrangement. When the control is interactive, use the same border pattern as a check box or text box. When the user chooses a particular value in the group, indicate the set value with a special selection border drawn around the edge of the control.

For more information about how to display well controls, see Chapter 13, "Visual Design."

Follow the same interaction techniques as option buttons. When the user clicks a well in the group the value is set to that choice. Provide a group box or static text to label the group and define an access key for that label and supporting the TAB key to navigate to a group. Use arrow keys to move between values in the group.

Pen-Specific Controls

When the user installs a pen input device, single line text boxes and combo boxes automatically display a writing tool button described in Chapter 5, "General Interaction Techniques." In addition, the system provides special controls for supporting pen input.

Boxed Edit Controls

A *boxed edit* control provides the user with a discrete area for entering characters. It looks and operates similarly to a writing tool window without some of the writing tool window's buttons, as shown in Figure 7.30.

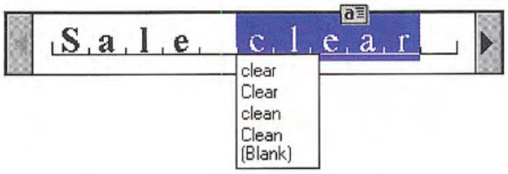


Figure 7.30 A single line boxed edit control

Both single and multiple line boxed edit controls are supported. Figure 7.31 shows a multiple line boxed edit control.

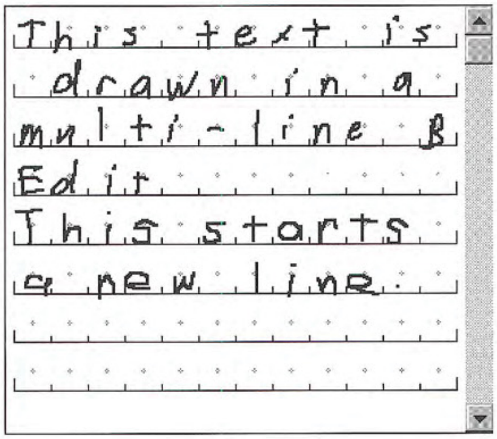


Figure 7.31 A multiple line boxed edit control

Like the writing tool window, these controls provide a pen selection handle for selection of text and an action handle for operations on a selection. They also provide easy correction by overwriting and selecting alternative choices.

Ink Edit Controls

The *ink edit* is a pen control in which the user can create and edit lines drawn as ink; no recognition occurs here. It is a drawing area designed for ink input, as shown in Figure 7.32.

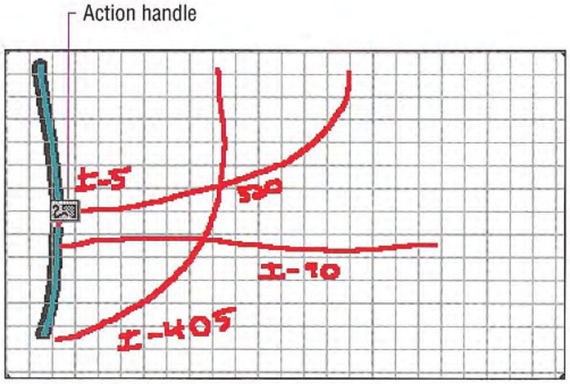


Figure 7.32 An ink edit control

The control provides support for an optional grid, optional scroll bars, and optional display of a frame border. Selection is supported using tapping to select a particular stroke; lasso-tapping is also supported for selecting single or multiple strokes. After the user makes a selection, an action handle is displayed. Tapping on the action handle displays a pop-up menu that includes commands for Undo, Cut, Copy, Paste, Delete, Use Eraser, Resize, What's This?, and Properties. Choosing the Properties command displays a property sheet associated with the selection — this allows the user to change the stroke width and color.

If you use an ink edit control, you may also want to include some controls for special functions. For example, a good addition is an Eraser button, as shown in Figure 7.33.

Figure 7.33 The eraser toolbar button

Implement the Eraser button to operate as a "spring-loaded" mode; that is, choosing the button causes the pen to act as an eraser while the user presses the pen to the screen. As soon as it is lifted, the pen reverts to its drawing mode.

Toolbars and Status Bars

Like menu bars, toolbars and status bar are special interface constructs for managing sets of controls. A *toolbar* is a panel that contains a set of controls, as shown in Figure 7.34, designed to provide quick access to specific commands or options. Specialized toolbars are sometimes called ribbons, tool boxes, and palettes.

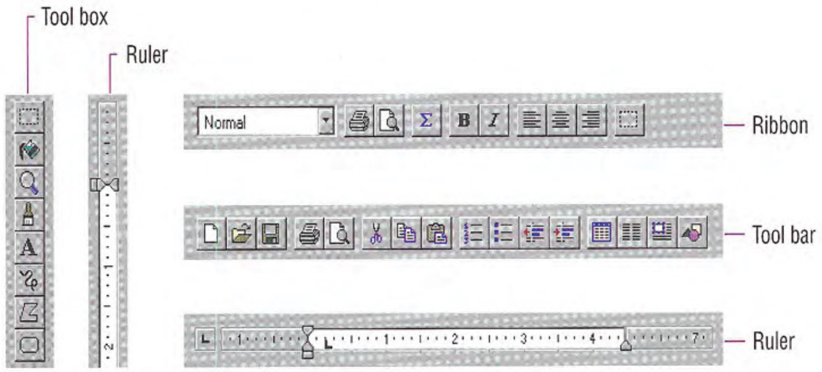


Figure 7.34 Examples of toolbars

A *status bar*, shown in Figure 7.35, is a special area within a window, typically the bottom, that displays information about the current state of what is being viewed in the window or any other contextual information, such as keyboard state. You can also use the status bar to provide descriptive messages about a selected menu or toolbar button. Like a toolbar, a status bar can contain controls; however, typically include read-only or noninteractive information.

For more information about status bar messages, see Chapter 12, "User Assistance."

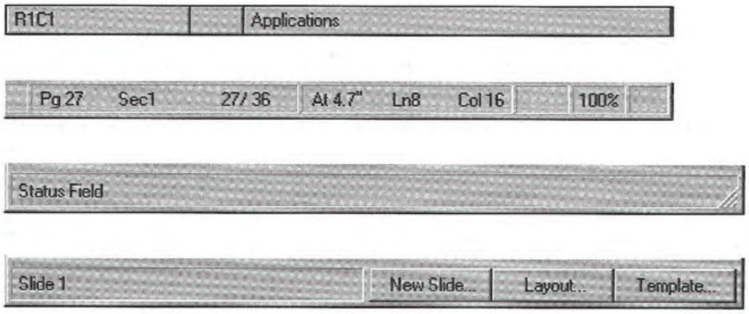


Figure 7.35 Examples of status bars

Interaction with Controls in Toolbars and Status Bars

The user can access the controls included in a toolbar or status bar with the mouse or pen through the usual means of interaction for those controls. You can provide keyboard access using either shortcut keys or access keys. If a control in a toolbar or status bar does not have a text label, access keys may not be as effective. Furthermore, if a particular access key is already in use in the primary window, it may not be available for accessing the control in the toolbar. For example, if the menu bar of the primary window is already using a particular access key, then the menu bar receives the key event.

When the user interacts with controls in a toolbar or status bar that reflect properties, any change is directly applied to the current selection. For example, if a button in a toolbar changes the property of text to bold, choosing that button immediately changes the text to bold; no further confirmation or transaction action is required. The only exception is if the control, such as a button, requires additional input from the user; then the effect may not be realized until the user provides the information for those parameters. An example of such an exception would be the selection of an object or a set of input values through a dialog box.

Always provide a tooltip for controls you include in a toolbar or status bar that do not have a text label. The system provides support for tooltips in the standard toolbar control and a tooltip control for use in other contexts.

Support for User Options

To provide maximum flexibility for users and their tasks, design your toolbars and status bars to be user configurable. Providing the user with the option to display or hide toolbars and status bars is one way to do this. You can also include options that allow the user to change or rearrange the elements included in toolbars and status bars.

Provide toolbar buttons in at least two sizes: 24 by 22 and 32 by 30 pixels. To fit a graphic label in these button sizes, design the images no larger than 16 by 16 and 24 by 24 pixels, respectively. In addition, support the user's the option to change between sizes by providing a property sheet for the toolbar (or status bar).

For more information about designing toolbar buttons, see Chapter 13, "Visual Design."

Consider also making the location of toolbars user adjustable. While toolbars are typically *docked* by default — aligned to the edge of a window or pane to which they apply — design your toolbars to be moveable so that the user can dock them along another edge or display them as a palette window.

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

To undock a toolbar from its present location, the user must be able to click anywhere in the "blank" area of the toolbar and drag it to its new location. If the new location is within the hot zone of an edge, your application should dock the toolbar at the new edge when the user releases the mouse button. If the new location is not within the hot zone of an edge, redisplay the toolbar in a palette window. To redock the window with an edge, the user drags the window by its title bar until the pointer enters the hot zone of an edge. Return the toolbar to a docked state when the user releases the mouse button.

As the user drags the toolbar, provide visual feedback, such as a dotted outline of the toolbar. When the user moves the pointer into a hot zone of a dockable location, display the outline in its docked configuration to provide a cue to the user about what will happen when the drag operation is complete. You can also support user options such as resizing the toolbar by dragging its border or docking multiple toolbars side by side, reconfiguring their arrangement and size as necessary.

When supporting toolbar and status bar configuration options, avoid including controls whose functionality is not available elsewhere in the interface. In addition, always preserve the current position and size, and other state information, of toolbar and status bar configuration so that they can be restored to their state when the user reopens the window.

Toolbar and Status Bar Controls

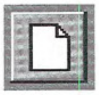
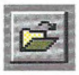
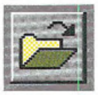
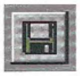
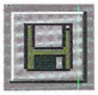
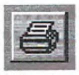
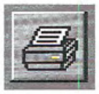
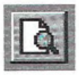
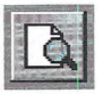
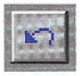
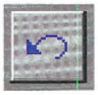
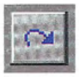
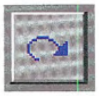
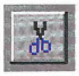
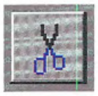
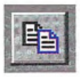
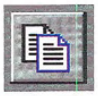
The system includes toolbar and status bar controls that you can use to implement these interfaces in your applications. The toolbar control supports docking and windowing functionality. It also supports a dialog box for allowing the user to customize the toolbar. You define whether the customization features are available to the user and what features the user can customize. The system also supports creation of desktop toolbars. For more information about desktop toolbars, see Chapter 10, "Integrating with the System."

The standard status bar control also includes the option of including a size grip control for sizing the window, described in Chapter 6, "Windows." When the status bar size grip is displayed, if the window displays a size grip at the junction of the horizontal and vertical scroll bars of a window, that grip should be hidden so that it does not appear in both locations at the same time. Similarly, if the user hides the status bar, restore the size grip at the corner of the scroll bars.

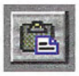
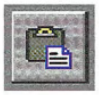
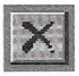
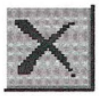
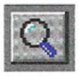
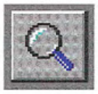
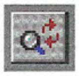
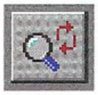
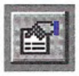
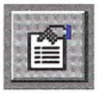
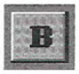
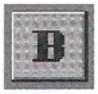
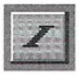
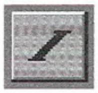
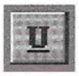
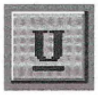
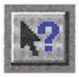
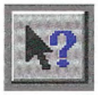
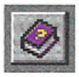
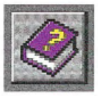
Common Toolbar Buttons

Table 7.5 illustrates the button images that you can use for common functions.

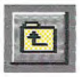
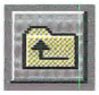
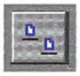
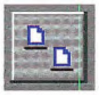
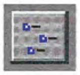
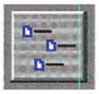
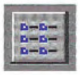
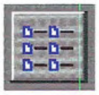
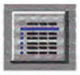
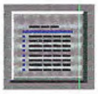
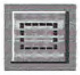
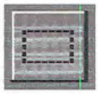
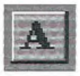
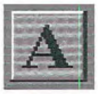
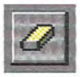
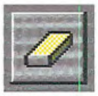
Table 7.5 Common Toolbar Buttons

16 x 16 button	24 x 24 button	Function
		New
		Open
		Save
		Print
		Print Preview
		Undo
		Redo
		Cut
		Copy

(Continued)

16 x 16 button	24 x 24 button	Function
		Paste
		Delete
		Find
		Replace
		Properties
		Bold
		Italic
		Underline
		What's This? (context-sensitive Help mode)
		Show Help Topics

(Continued)

16 x 16 button	24 x 24 button	Function
		Open parent folder
		View as large icons
		View as small icons
		View as list
		View as details
		Region selection tool
		Writing tool (pen)
		Eraser tool (pen)

Use these images only for the function described. Consistent use of these common tool images allows the user to transfer their learning and skills from product to product. If you use one of the standard images for a different function, you may confuse the user. When designing your own toolbar buttons, follow the conventions supported by the standard system controls.

For more information about the design of toolbar buttons, see Chapter 13, "Visual Design."

CHAPTER

8

Secondary Windows

Most primary windows require a set of secondary windows to support and supplement a user's activities in the primary windows. Secondary windows are similar to primary windows but differ in some fundamental aspects. This chapter covers the common uses of secondary windows, such as property sheets, dialog boxes, palette windows, and message boxes.

Characteristics of Secondary Windows

Although secondary windows share some characteristics with primary windows, they also differ from primary windows in their behavior and use. For example, secondary windows should not appear on the taskbar. Secondary windows obtain or display supplemental information which is often related to the objects that appear in a primary window.

Appearance and Behavior

A typical secondary window, as shown in Figure 8.1, includes a title bar and frame; a user can move it by dragging its title bar. However, a secondary window does not include Maximize and Minimize buttons because these sizing operations typically do not apply to a secondary window. A Close button can be included to dismiss the window. The title text is a label that describes the purpose of the window; the content of the label depends on the use of the window. The title bar does not include icons.

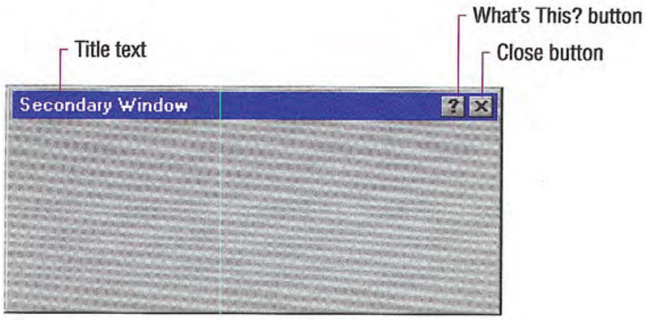


Figure 8.1 A secondary window

You can include status information in secondary windows, but avoid including a status bar control used in primary windows.

Like a primary window, a secondary window includes a pop-up menu with commands that apply to the window. A user can access the pop-up menu for the window using the same interaction techniques as primary windows.

A secondary window can also include a What's This? button in its title bar. This button allows a user to display context-sensitive Help information about the components displayed in the window.

Interaction with Other Windows

Secondary windows that are displayed because of commands chosen within a primary window depend on the state of the primary window; that is, when the primary window is closed or minimized, its secondary windows are also closed or hidden. When the user reopens or restores the primary window, restore the secondary windows to their former positions and states. However, if opening a secondary window is the result of an action outside of the object's primary window — for example, if the user chooses the Properties command on an icon in a folder or on the desktop — then the property sheet window is independent and appears as a peer with any primary windows, though it should not appear in the taskbar.

When the user opens or switches to a secondary window, it is activated or deactivated like any other window. With the mouse or pen, the user activates a secondary window in the same way as a primary window. With the keyboard, the ALT+F6 key combination switches between a secondary window and its primary window, or other peer secondary windows that are related to its primary window. A secondary window must be modeless to support this form of switching.

When the user activates a primary window, bringing it to the top of the window Z order, all of its dependent secondary windows also come to the top, maintaining their same respective order. Similarly, activating a dependent secondary window brings its primary window and related peer windows to the top.

A dependent secondary window always appears on top of its associated primary window, layered with any related window that is a peer secondary window. When activated, the secondary window appears on top of its peers. When a peer is activated, the secondary window appears on top of its primary window, but behind the newly activated secondary window that is a peer.

You can design a secondary window to always appear at the top of its peer secondary windows. Typically, you should use this technique only for palette windows and, even in this situation, make this feature configurable by the user by providing an Always On Top property setting for the window. If you support this technique for multiple secondary windows, then the windows are managed in their own Z order within the collection of windows of which they are a part.

Avoid having a secondary window with the Always On Top behavior appear on top of another application's primary window (or any of the other application's dependent secondary windows) when the user activates a window of that application, unless the Always On Top window can also be applied to that application's windows.

When the user chooses a command that opens a secondary window, use the context of the operation to determine how to present information in that window. In property sheets, for example, set the values of the properties in that window to represent the selection.

In general, display the window in the same state as the user last accessed it. For example, an Open dialog box should preserve the current directory setting between the openings of a window. Similarly, if you use tabbed pages for navigating through information in a secondary window, display the last page the user was viewing when the user closed the window. This makes it easier for the user to repeat an operation that is associated with the window. It also provides more stability in the interface.

However, if a command or task implies or requires that the user begin a process in a particular sequence or state, such as with a wizard window, you should present the secondary window using a fixed or consistent presentation. For example, entering a record into a database may require the user to enter the data in a particular sequence. Therefore, it may be more appropriate to present the input window always displaying the first entry field.

Unfolding Secondary Windows

Except for palette windows, avoid defining secondary windows to be resizable because their purpose is to provide concise, predefined information. However, you can use an unfold button to expand a window to reveal additional options as a form of progressive disclosure. An *unfold button* is a command button with a label that includes two "greater than" characters (>>). When the user chooses the button, the secondary window expands to its alternative fixed size. As an option, you can use the button to "refold" the additional part of the window.

Cascading Secondary Windows

You can also provide the user access to additional options by including a command button that opens another secondary window. If the resulting window is independent in its operation, close the secondary window from which the user opened it and display only the new window. However, if the intent of the subsequent window is to obtain information for a field in the original secondary window, then the original should remain displayed and the dependent window should appear on top, offset slightly to the right and below the original secondary window. When using this latter method, limit the number of secondary windows to a single level to avoid creating a cluttered cascading chain of hierarchical windows.

Window Placement

When determining where to place a secondary window consider a number of factors, including the use of the window, the overall display dimensions, and the reason for the appearance of the window. In general, display a secondary window where it last appeared. If the user has not yet established a location for the window, place the window in a location that is convenient for the user to navigate to and that fully displays the window. If neither of these guidelines apply, horizontally center the secondary window within the primary window, just below the title bar, menu bar, and any docked toolbars.

Modeless vs. Modal

A secondary window can be modeless or modal. A *modeless* secondary window allows the user to interact with either the secondary window or the primary window, just as the user can switch between primary windows. It is also well suited to situations where the user wants to repeat an action — for example, finding the occurrence of a word or formatting the properties of text.

A *modal* secondary window requires the user to complete interaction within the secondary window and close it before continuing with any further interaction outside the window. A secondary window can be modal in respect to its primary window or the system. In the latter case, the user must respond and close the window before interacting with any other windows or applications.

Because modal secondary windows restrict the user's choice, use them sparingly. Limit their use to situations when additional information is required to complete a command or when it is important to prevent any further interaction until satisfying a condition. Avoid using system modal secondary windows unless your application operates as a system level utility and then only use them in severe situations — for example, when an impending fatal system error or unrecoverable condition occurs.

Default Buttons

When defining a secondary window, you can assign the ENTER key to activate a particular command button, called the *default button*, in the window. The system distinguishes the default button from other command buttons with a bold outline that appears around the button.

Define the default button to be the most likely action, such as a confirmation action or an action that applies transactions made in the secondary window. Avoid making a command button the default button if its action is irreversible or destructive. For example, in a text search and substitution window, do not use a Replace All button as the default button for the window.

You can change the default button as the user interacts with the window. For example, if the user navigates to a command button that is not the default button, the new button temporarily becomes the default. In such a case, the new default button takes on the default appearance, and the former default button loses the default appearance. Similarly, if the user moves the input focus to another control within the window that is not a command button, the original default button resumes being the default button.

The assignment of a default button is a common convention. However, when there is no appropriate button to designate as the default button or another control requires the ENTER key (for example, entering new lines in a multiline text control), you cannot define a default button for the window. In addition, when a particular control has the input focus and requires use of the ENTER key, you can temporarily have no button defined as the default. Then when the user moves the input focus out of the control, you can restore the default button.

Optionally, you can use double-clicking on single selection control, such as an option button or single selection list, as a shortcut technique to set or select the option and carry out the default button of the secondary window.

Navigation in Secondary Windows

With the mouse and pen, navigation to a particular field or control involves the user pointing to the field and clicking or tapping it. For button controls, this action also activates that button. For example, for check boxes, it toggles the check box setting and for command buttons, it carries out the command associated with that button.

The keyboard interface for navigation in secondary windows uses the TAB and SHIFT+TAB keys to move between controls, to the next and previous control, respectively. Each control has a property that determines its place in the navigation order. Set this property such that the user can move through the secondary window following the usual conventions for reading: in western countries, left-to-right and top-to-bottom, with the primary control the user interacts with located in the upper left area of the window. Order controls such that the user can progress through the window in a logical sequence, proceeding through groups of related controls. Command buttons for handling overall window transactions are usually at the end of the order sequence.

You need not provide TAB key access to every control in the window. When using static text as a label, set the control you associated with it as the appropriate navigational destination, not the static text field itself. In addition, combination controls such as combo boxes, drop-down combo boxes, and spin boxes are considered single controls for navigational purposes. Because option buttons typically appear as a group, use the TAB key for moving the input focus to the current set choice in that group, but not between individual options — use arrow keys for this purpose. For a group of check boxes, provide TAB navigation to each control because their settings are independent of each other.

Optionally, you can also use arrow keys to support keyboard navigation between controls in addition to the TAB navigation technique wherever the interface does not require those keys. For example, you can use the UP ARROW and DOWN ARROW keys to navigate between single-line text boxes or within a group of check boxes or command buttons. Always use arrow keys to navigate between option button choices and within list box controls.

You can also use access keys to provide navigation to controls within a secondary window. This allows the user to access a control by pressing and holding the ALT key and an alphanumeric key that matches the access key character designated in the label of the control.

For more information about guidelines for selecting access keys, see Chapter 4, "Input Basics."

Unmodified alphanumeric keys also support navigation if the control that currently has the input focus does not use these keys for input. For example, if the input focus is currently on a check box control and the user presses an alphanumeric key, the input focus moves to the control with the matching access key. However, if the input focus is in a text box or list box, an alphanumeric key is used as text input for that control so the user cannot use it for navigation within the window without modifying it with the ALT key.

Access keys not only allow the user to navigate to the matching control, they have the same effect as clicking the control with the mouse. For example, pressing the access key for a command button carries out the action associated with that button. To ensure the user direct access to all controls, select unique access keys within a secondary window.

You can also use access keys to support navigation to a control, but then return the input focus to the control from which the user navigated. For example, when the user presses the access key for a specific command button that modifies the content of a list box, you can return the input focus to the list box after the command has been carried out.

OK and Cancel command buttons are typically not assigned access keys if they are the primary transaction keys for a secondary window. In this case, the ENTER and ESC keys, respectively, provide access to these buttons.

Pressing ENTER always navigates to the default command button, if one exists, and invokes the action associated with that button. If there is no current default command button, then a control can use the ENTER key for its own use.

Validation of Input

Validate the user's input for a field or control in a secondary window as closely to the point of input as possible. Ideally, input is validated when it is entered for a particular field. You can either disallow the input, or use audio and visual feedback to alert the user that the data is not appropriate. You can also display a message box, particularly if the user repeatedly tries to enter invalid input. You can also reduce invalid feedback by using controls that limit selection to a specific set of choices — for example, check boxes, option buttons, drop-down lists — or preset the field with a reasonable default value.

If it is not possible to validate input at the point of entry, consider validating the input when the user navigates away from the control. If this is not feasible, then validate it when the transaction is committed, or whenever the user attempts to close the window. At that time, leave the window open and display a message; after the user dismisses the message, set the input focus to the control with the inappropriate data.

Property Sheets and Inspectors

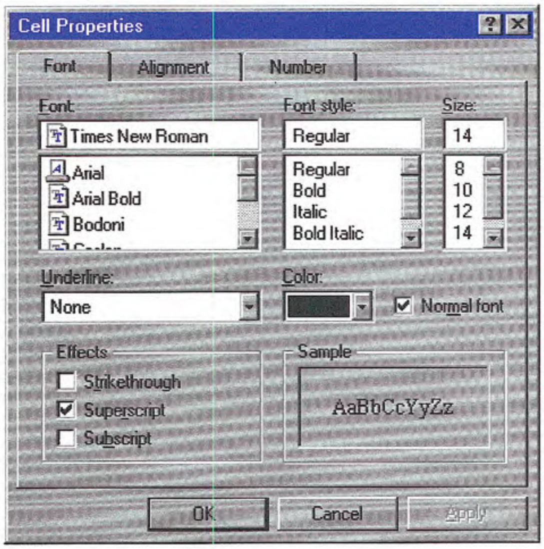
You can display the properties of an object in the interface in a number of ways. For example, some folder views display certain file system properties of an object. The image and name of an icon on the desktop also reflect specific properties of that object. You can also use other interface conventions, such as toolbars, status bars, or even scroll bars, to reflect certain properties. The most common presentation of an object's properties is a secondary window, called a property sheet. A *property sheet* is a modeless secondary window that displays the user-accessible properties of an object — that is, viewable, but not necessarily editable properties. Display a property sheet when the user chooses the Properties command for an object.

A *property inspector* is different from a property sheet — even when a property sheet window is modeless, the window is typically modal with respect to the object for which it displays properties. If the user selects another object, the property sheet continues to display the properties of the original object. A property inspector always reflects the current selection.

Property Sheet Interface

The title bar text of the property sheet identifies the displayed object. If the object has a name, use its name and the word "Properties". If the combination of the name plus "Properties" exceeds the width of the title bar, the system truncates the name and adds an ellipsis. If the object has no name, use the object's type name. If the property sheet represents several objects, then also use the objects' type name. Where the type name cannot be applied — for example, because the selection includes heterogeneous types — substitute the word "Selection" for the type name.

Because there can be numerous properties for an object and its context, you may need to categorize and group properties as sets within the property window. There are two techniques for supporting navigation to groups of properties in a property sheet. The first is a tabbed *property page*. Each set of properties is presented within the window as a page with a tab labeled with the name of the set. Use tabbed property pages for grouping peer-related property sets, as shown in Figure 8.2.



Cell Properties

Font: Times New Roman

Font style: Regular

Size: 14

Underline: None

Color:

☑ Normal font

Effects

□ Strikethrough

☑ Superscript

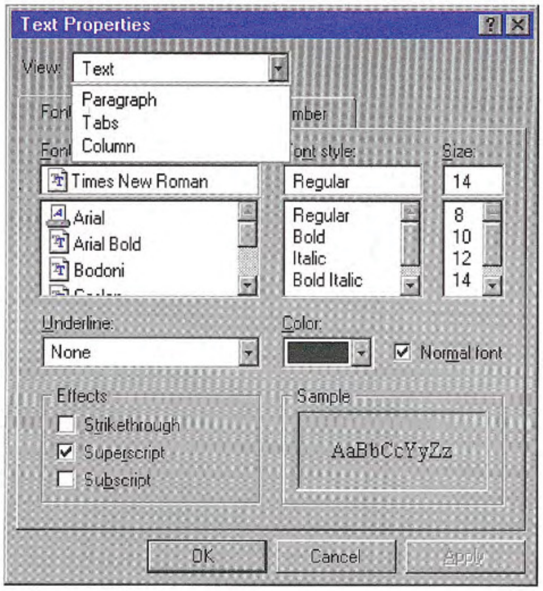
□ Subscript

Sample

AaBbCcYyZz

Figure 8.2 A property sheet with tabbed pages

When displaying the property sheet of an object, you can also provide access to the properties of the object's immediate context or hierarchically related properties in the property sheet. For example, if the user selects text, you may want to provide access to the properties of the paragraph of that text in the same property sheet. Similarly, if the user selects a cell in a spreadsheet, you may want to provide access to its related row and column properties in the same property sheet. Although you can support this with additional tabbed pages, better access may be facilitated using another control — such as a drop-down list — to switch between groups of tabbed pages, as shown in Figure 8.3. This technique can also be used instead of multiple rows of tabs.



View: Text

Font: Times New Roman

Font style: Regular

Size: 14

Underline: None

☑ Normal font

☐ Strikethrough

☑ Superscript

☐ Subscript

Figure 8.3 A drop-down list for access to hierarchical property sets

Where possible, make the values for properties found in property sheets transferable. You can support special transfer completion commands to enable copying only the properties of an object to another object. For example, you may want to support transferring data for text boxes or items in a list box.

For more details on transfer operations, see Chapter 5, "General Interaction Techniques."

Property Sheet Commands

Property sheets typically allow the user to change the values for a property and then apply those transactions. Include the following common command buttons for handling the application of property changes.

Command	Action
OK	Applies all pending changes and closes the property sheet window.
Apply	Applies all pending changes but leaves the property sheet window open.
Cancel	Discards any pending changes and closes the property sheet window. Does not cancel or undo changes that have already been applied.

Optionally, you can also support a Reset command for canceling pending changes without closing the window.

You can also include other command buttons in property sheets. However, the location of command buttons within the property sheet window is very important. If you place a button on a property page, apply the action associated with the button to that page. For command buttons placed outside the page but still inside the window, apply the command to the entire window.

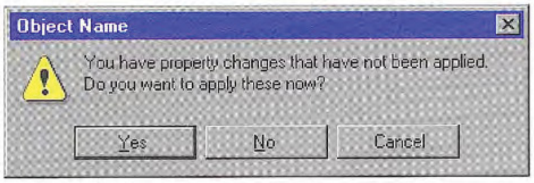
For the common property sheet transaction buttons — OK, Cancel, and Apply — it is best to place the buttons outside the pages because users consider the pages to be just a simple grouping or navigation technique. This means that if the user makes a change on one page, the change is not applied when the user switches pages. However, if the user makes a change on the new page and then chooses the OK or Apply command buttons, both changes are applied — or, in the case of Cancel, discarded.

If your design requires groups of properties to be applied on a page-by-page basis, then place OK, Cancel, and Apply command buttons on the property pages, always in the same location on each page. When the user switches pages, any property value changes for that page are applied, or you can prompt the user with a message box whether to apply or discard the changes.

You can include a sample in a property sheet window to illustrate a property value change that affects the object when the user applies the property sheet. Where possible, include the aspect of the object that will be affected in the sample. For example, if the user selects text and displays the property sheet for the text, include part of the text selection in the property sheets sample. If displaying the actual object — or a portion of it — in the sample is not practical, use an illustration that represents the object's type.

Closing a Property Sheet

If the user closes a property sheet window, follow the same convention as closing the content view of an object, such as a document. Avoid interpreting the Close button as Cancel. If there are pending changes that have not been committed, prompt the user to apply or discard the changes through a message box, as shown in Figure 8.4. If there are no unsaved changes, just close the window.



You have property changes that have not been applied.
Do you want to apply these now?

Figure 8.4 Prompting for pending property changes

If the user chooses the Yes button, the properties are applied and the message box window and the property sheet window are removed. If the user chooses the No button, the pending changes are discarded and the message box and property sheet windows are closed. Include a Cancel button in the message box, to allow the user to cancel the closing of the property sheet window.

Property Inspectors

You can also display properties of an object using a dynamic viewer or browser that reflects the properties of the current selection. Such a property window is called a property inspector. When designing a property inspector, use a toolbar or palette window, or preferably a toolbar that the user can configure as a docked toolbar or palette window, as shown in Figure 8.5.

For more information about supporting docked and windowed toolbars, see Chapter 7, "Menus, Controls, and Toolbars." For more information about palette windows, see the section, "Palette Windows," later in this chapter.

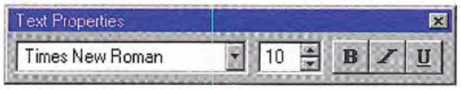


Figure 8.5 A property inspector

Apply property transactions that the user makes in a property inspector dynamically. That is, change the property value in the selected object as soon as the user makes the change in the control reflecting that property value.

Property inspectors and property sheets are not exclusive interfaces; you can include both. Each has its advantages. You can choose to display only the most common or frequently accessed properties in a property inspector and the complete set in the property sheet. You also can include multiple property inspectors, each optimized for managing certain types of objects.

As an option, you also can provide an interface for the user to change the behavior between a property sheet and a property inspector form of interaction. For example, you can provide a control on a property inspector that "locks" its view to be modal to the current object rather than tracking the selection.

Properties of a Multiple Selection

When a user selects multiple objects and requests the properties for the selection, reflect the properties of all the objects in a single property sheet or property inspector rather than opening multiple windows. Where the property values differ, display the controls associated with those values using the mixed value appearance — sometimes referred to as the indeterminate state. However, also support the display of multiple property sheets when the user displays the property sheet of the objects individually. This convention provides the user with sufficient flexibility. If your design still requires access to individual properties when the user displays the property sheet of a multiple selection, include a control such as a list box or drop-down list in the property window for switching between the properties of the objects in the set.

Properties of a Heterogeneous Selection

When a multiple selection includes different types of objects, include the intersection of the properties between the objects in the resulting property sheet. If the container of those selected objects treats the objects as if they were of a single type, the property sheet includes properties for that type only. For example, if the user selects text and an embedded object, such as a circle, and in that context an embedded object is treated as an element within the text stream, present only the text properties in the resulting property sheet.

Properties of Grouped Items

When displaying properties, do not equate a multiple selection with a grouped set of objects. A group is a stronger relationship than a simple selection, because the aggregate resulting from the grouping can itself be considered an object, potentially with its own properties and operations. Therefore, if the user requests the properties of a grouped set of items, display the properties of the group or composite object. The properties of its individual members may or may not be included, depending on what is most appropriate.

Dialog Boxes

A *dialog box* provides an exchange of information or dialog between the user and the application. Use a dialog box to obtain additional information from the user — information needed to carry out a particular command or task.

Because dialog boxes generally appear after choosing a particular menu item (including pop-up or cascading menu items) or a command button, define the title text for the dialog box window to be the name of the associated command. Do not include an ellipsis in the title text, even if the command menu name may have included one. Also, avoid including the command's menu title unless necessary to compose a reasonable title for the dialog box. For example, for a Print command on the File menu, define the dialog box window's title text as Print, not Print... or File Print. However, for an Object... command on an Insert menu, you can title the dialog box as Insert Object.

Dialog Box Commands

Like property sheets, dialog boxes commonly include OK and Cancel command buttons. Use OK to apply the values in the dialog box and close the window. If the user chooses Cancel, the changes are ignored and the window is closed, canceling the operation the user chose. OK and Cancel buttons work best for dialog boxes that allow the user to set the parameters for a particular command. Typically, define OK to be the default command button when the dialog box window opens.

You can include other command buttons in a dialog box in addition to or replacing the OK and Cancel buttons. Label your command buttons to clearly define the button's purpose, but be as concise as possible. Long, wordy labels make it difficult for the user to easily scan and interpret a dialog box's purpose. Follow the design conventions for command buttons.

For more information about command buttons, see Chapter 7, "Menus, Controls, and Toolbars," and Chapter 13, "Visual Design."

Layout

Orient controls in dialog boxes in the direction people read. In countries where roman alphabets are used, this means left to right, top to bottom. Locate the primary field with which the user interacts as close to the upper left corner as possible. Follow similar guidelines for orienting controls within a group in the dialog box.

Lay out the major command buttons either stacked along the upper right border of the dialog box or lined up across the bottom of the dialog box. Position the most important button — typically the default command — as the first button in the set. If you use the OK and Cancel buttons, group them together. You can use other arrangements if there is a compelling reason, such as a natural mapping relationship. For example, it makes sense to place buttons labeled North, South, East, and West in a compass-like layout. Similarly, a command button that modifies or provides direct support for another control may be grouped or placed next to those controls. However, avoid making that button the default button because the user will expect the default button to be in the conventional location.

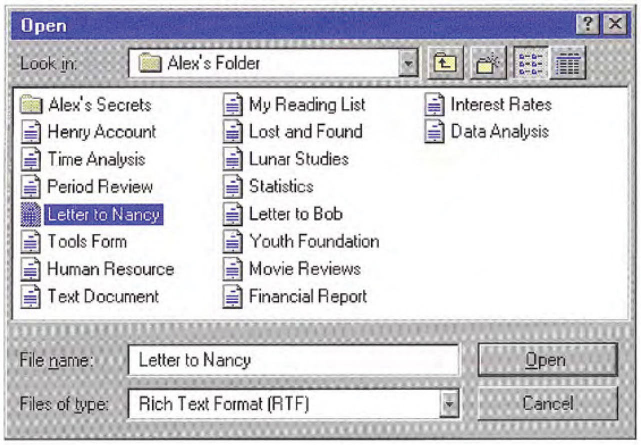
Common Dialog Box Interfaces

The system provides prebuilt interfaces for many common operations. Use these interfaces where appropriate. They can save you time while providing a high degree of consistency. If you customize or provide your own interfaces, maintain consistency with the basic functionality supported in these interfaces and the guidelines for their use. For example, if you provide your own property sheet for font properties, model your design to be similar in appearance and design to the common Font dialog box. Consistent visual and operational styles will allow users to more easily transfer their knowledge and skills.

The common dialog box interfaces have been revised from the ones provided in previous releases of Microsoft Windows.

Open Dialog Box

The Open dialog box, as shown in Figure 8.6, allows the user to browse the file system, including direct browsing of the network, and includes controls to open a specified file. Use this dialog box to open files or browse for a filename, such as the File Open menu command or a Browse command button. Always set the title text to correctly reflect the command that displays the dialog box.



Look in: Alex's Folder

File name: Letter to Nancy

Files of type: Rich Text Format (RTF)

Figure 8.6 The Open dialog box

The system-supplied dialog box automatically handles the display of long filenames, direct manipulation transfers — such as drag and drop — and access to an icon's pop-up menus. The dialog box only displays filename extensions for files of registered types when the user selects this viewing option.

To open a file, the user selects a file from the list in the dialog box, or types a name in the File Name field and then chooses the Open command. The user can also display the pop-up menu for the file and choose its Open command. As a shortcut, double-clicking also opens the file. Choosing the Cancel button closes the window without opening the file.

When the user opens a shortcut icon, the dialog box opens the file of the object to which the link refers. In other words, the effect is the same as if the user directly opened the original file. Therefore, the name of the original file — not the name of the file link — should appear in the primary window's title bar.

The files listed in the dialog box reflect the current directory path and the type filter set in the Files Of Type drop-down list box. The list of files also includes shortcut icons in the current directory; these shortcut icons refer to file types that match the type filter.

The Look In drop-down list box displays the current directory. Displaying the list allows the user to view the hierarchy of the directory path and to navigate up the path tree. Tool buttons that are adjacent to this control provide the user with easy access to common functions. The dialog box also supports pop-up menus for the icons, the view in the list of files box, and the other controls in the window.

Set the default directory based on context. If the user opened the file directly, either from its location from the file system or using the Open dialog box, set the directory path to that location. If the user opened the application directly, then you can set the path as best fits the application. For example, an application may set up a default directory for its data files.

The user can change the directory path by selecting a different item in the Look In list, selecting a file system container (such as a folder) in the list of files, or entering a valid path in the File Name field and choosing the Open button. Choosing the Cancel button should not change the path. Always preserve the latest directory path between subsequent openings of the dialog box. If the application supports opening multiple files, such as in MDI design, set the directory path to the last file opened, not the currently active child window. However, for multiple instances of an application, maintain the path separately for each instance.

Your application determines the default Files Of Type filter for the Open dialog box. This can be based on the last file opened, the last file type set by the user, or always a specific type, based on what most appropriately fits the context of the application.

The user can change the type filter by selecting a different type in the Files Of Type drop-down list box or by typing a filter into the File Name text box and choosing the Open button. Filters can include filename extensions. For example, if the user types in **.txt* and chooses the Open button, the list displays only files with the type extension of .TXT. Typing an extension into this text box also changes the respective type setting for the Files Of Type drop-down list box. If the application does not support that type, display the Files Of Type control with the mixed-case (indeterminate) appearance.

Include the types of files your application supports in the Files Of Type drop-down list box. For each item in the list, use a type description preferably based on the registered type names for the file types. For example, for text files, the type descriptor should be "Text Documents". You can also include an "All Files" entry to display all files in the current directory, regardless of type.

When the user types a filename into the Open dialog box and chooses the Open button, the following conventions apply:

- The string includes no extension: the system attempts to use your application's default extension or the current setting in the Files Of Type drop-down list box. For example, if the user types in *My Document*, and the application's default extension is .DOC, then the system attempts to open My Document.doc. (The extension is not displayed.) If the user changes the type setting to Text Documents (*.txt), the file specification is interpreted as My Document.txt. If using the application's default type or the type setting fails to find a matching file, the system attempts to open a file that appears in the list of files with the same name (regardless of extension). If more than one file matches, the first will be selected and the system displays a message box indicating multiple files match.

- The string includes an extension: the system first checks to see if it matches the application's default type, any other registered types, or any extension in the Files Of Type drop-down list box. If it does not match, the system attempts to open it using the application's default type or the current type setting in the Files Of Type drop-down list box. For example, Microsoft WordPad will open the file A Letter to Dr. Jones provided that: the file's type matches the .DOC extension or the current type setting, and because the characters Jones (after the period) do not constitute a registered type. If this fails, the system follows the same behavior as for a file without an extension, checking for a match among the files that appear in the list of files.

- The string includes double-quotes at the beginning and end: the system interprets the string exactly, without the quotes and without appending any extension. For example, "My Document" is interpreted as My Document.

- The system fails to find a file: when the system cannot find a file, it displays a message box indicating that the file could not be found and advises the user to check the filename and path specified. However, your application may chose to handle this condition itself.

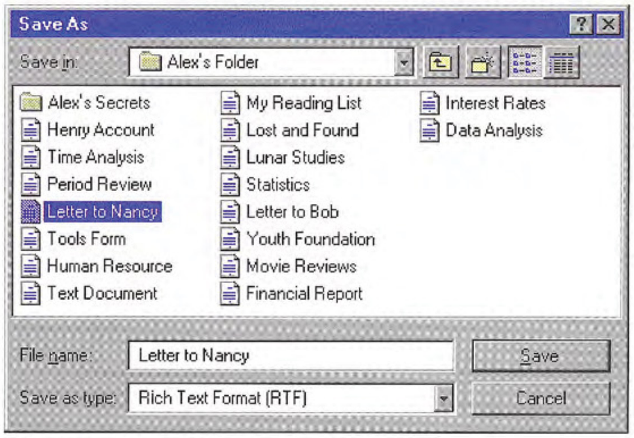
- The string the user types in includes invalid characters for a filename: the system displays a message box advising the user of this condition.

The Open dialog only handles the matching of a name to a file. It is your application's responsibility to ensure the format of the file is valid, and if not, to appropriately notify the user.

Save As Dialog Box

The Save As dialog box, as shown in Figure 8.7, is designed to save a file using a particular name, location, type, and format. Typically, applications that support the creation of multiple user files provide this command. However, if your application maintains only private data files and automatically updates those files, this dialog box may not be appropriate.

Display this dialog box when the user chooses the Save As command or file-oriented commands with a similar function, such as the Export File command. Also display the Save As dialog box when the user chooses the Save command, and has not supplied or confirmed a filename. If you use this dialog box for other tasks that require saving files, define the title text of the dialog box to appropriately reflect that command.



Save in: Alex's Folder

File name: Letter to Nancy

Save as type: Rich Text Format (RTF)

Figure 8.7 The Save As dialog box

The appearance and operation of the Save As dialog box is similar to the Open dialog box, except that the type field — the Save As Type drop-down list box — defines the default type for the saved file; it also filters the list of files displayed in the window.

To save a file, the user chooses the Save button and saves the file with the name that appears in the File Name text box. Although the user can type in a name or select a file from the list of files, your application should preset the field to the current name of the file. If the file has not been named yet, propose a name based on the registered type name for the file — for example, Text Document (2).

For more information about naming files, see Chapter 6, "Windows," and Chapter 10, "Integrating with the System."

The Save In drop-down list box indicates the immediate container in the directory path (or folder). The user can change the path using this control and the list of files box. If the file already exists, always save the file to its original location. This means that the current path for the Save As dialog box should always be set to the path where the file was last saved. If the file has never been saved, save the file with your application's default path setting or to the location defined by the user, either by typing in the path or by using the controls in the dialog box.

If the user chooses the Cancel button in the Save As dialog box, do not save the file or other settings. Restore the path to its original setting.

Include the file types supported by your application in the Save As Type drop-down list box. You may need to include a format description as part of a type name description. Although a file's format can be related to its type, a format and a type are not the same thing. For example, a bitmap file can be stored in monochrome, 16 , 256 or 24-bit color format, but the file's type is the same for all of them. Consider using the following convention for the items you include as type descriptions in the Save As Type drop-down list box.

Type Name [*Format Description*]

When the user supplies a name of the file, the Save As dialog box follows conventions similar to the Open dialog box. If the user does not include an extension, the system uses the setting in the Save As Type drop-down list or your application's default file type. If the user includes an extension, the system checks to see if the extension matches your application's default extension or a registered extension. If it does, the system saves the file as the type matching that extension. (The extension is hidden unless the system is set to display extensions.) Otherwise, the system interprets the user-supplied extension as part of the filename and appends the extension set in the Save As Type field. Note that this only means that the type (extension) is set. The format may not be correct for that type. It is your application's responsibility to write out the correct format.

Make certain you preserve the creation date for files that the user opens and saves. If your application saves files by creating a temporary file then deletes the original, renaming the temporary file to the original filename, be certain you copy the creation date from the original file. Certain system file management functionality may depend on preserving the identity of the original file.

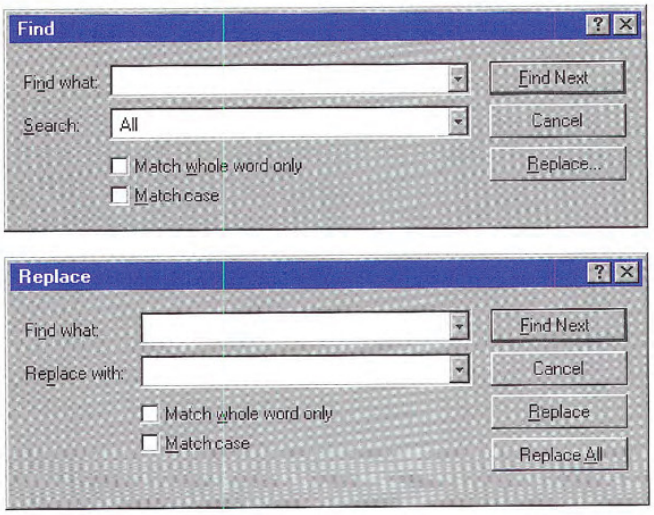
If the user types in a filename beginning and ending with double quotes, the system saves the file without appending any extension. If the string includes a registered extension, the file appears as that type. If the user supplies a filename with invalid characters or the specified path does not exist, the system displays a message box, unless your application handles these conditions.

Here are some examples of how the system saves user supplied filenames. Examples assume .TXT as the application's default type or the Save As Type setting.

What the user types	How system saves the file	Description
My File	My File.txt	Type is based on the file type established in Save As Type drop-down list box or the application's default type.
My File.txt	My File.txt	Type must match the application's default type or a registered type.
My File for Mr. Jones	My File for Mr. Jones.txt	. Jones does not qualify as a registered type or a type included in the Save As Type drop-down list box, so the type is appended based on the Save As Type setting or the application's default type.
My File for Mr. Jones.txt	My File for Mr. Jones.txt	Type must match a registered type or a type included in the Save As Type drop-down list box.
"My File"	My File	Type will be unknown. The file is saved exactly as the string between the quotes appears.
"My File.txt"	My File.txt	No type is appended. The file is saved exactly as the string between the quotes appears.
My File.	My File..txt	Type is based on the Save As Type drop-down list box or the application's default type.
"My File."	My File.	Type will be unknown.
"My" File	File is not saved.	System (or application) displays a message box notifying the user of invalid filename.

Find and Replace Dialog Boxes

The Find and Replace dialog boxes provide controls that search for a text string specified by the user and optionally replace it with a second text string specified by the user. These dialog boxes are shown in Figure 8.8.

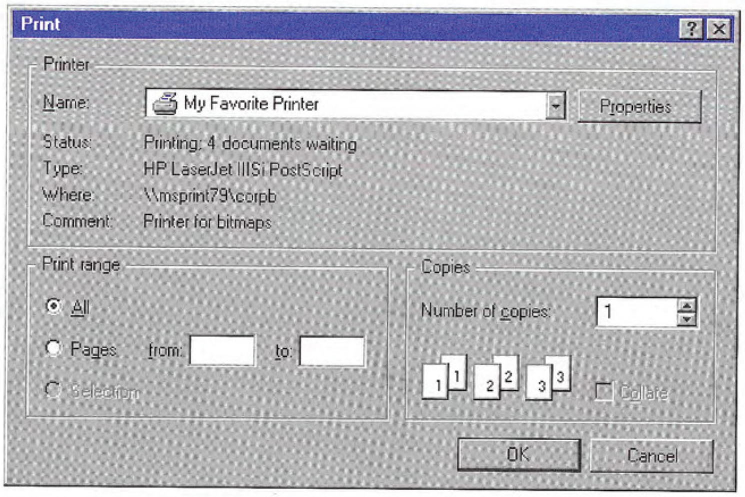


Search: All

Figure 8.8 The Find and Replace dialog boxes

Print Dialog Box

The Print dialog box, shown in Figure 8.9, allows the user to select what to print, the number of copies to print, and the collation sequence for printing. It also allows the user to choose a printer and provides a command button that provides shortcut access to that printer's properties.



Name: My Favorite Printer

Status: Printing: 4 documents waiting

Type: HP LaserJet IIISi PostScript

Where: \\msprint79\corpb

Comment: Printer for bitmaps

● All

○ Pages from: to:

○ Selection

Number of copies: 1

□ Collate

Figure 8.9 The Print dialog box

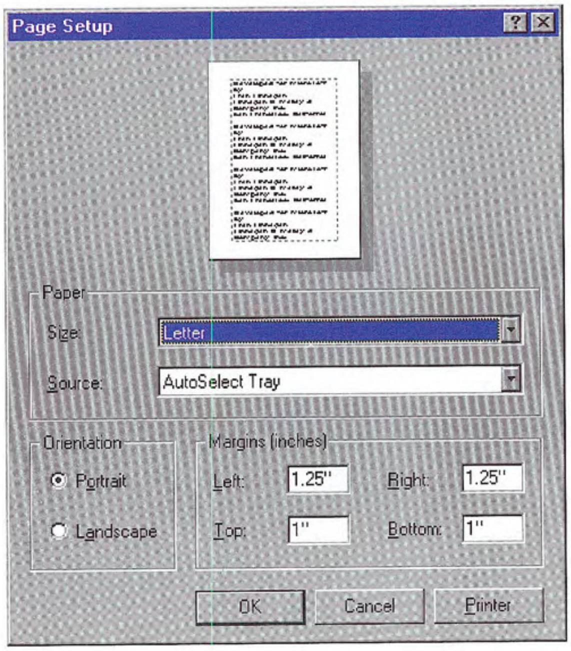
Print Setup Dialog Box

The Print Setup dialog box displays the list of available printers and provides controls for selecting a printer and setting paper orientation, size, source, and other printer properties.

Do not include this dialog box if you are creating or updating your application for Microsoft Windows 95 or later releases.

Page Setup Dialog Box

The Page Setup dialog box, as shown in Figure 8.10, provides controls for specifying properties about the page elements and layout.



Page Setup

Paper

Size: Letter

Source: AutoSelect Tray

Orientation

● Portrait

○ Landscape

Margins (inches)

Left: 1.25"

Right: 1.25"

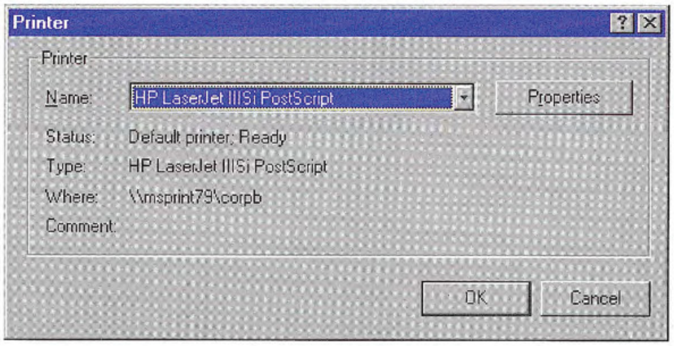
Top: 1"

Bottom: 1"

Figure 8.10 Page Setup interface used as a dialog box

In this context, page orientation refers to the orientation of the page and not the printer, which may also have these properties. Generally, the page's properties override those set by the printer, but only for the printing of that page or document.

The Printer button in the dialog box displays a supplemental dialog box (as shown in Figure 8.11) that provides information on the current default printer. Similarly to the Print dialog box, it displays the current property settings for the default printer and a button for access to the printer's property sheet.



Name: HP LaserJet IIISi PostScript

Status: Default printer; Ready

Type: HP LaserJet IIISi PostScript

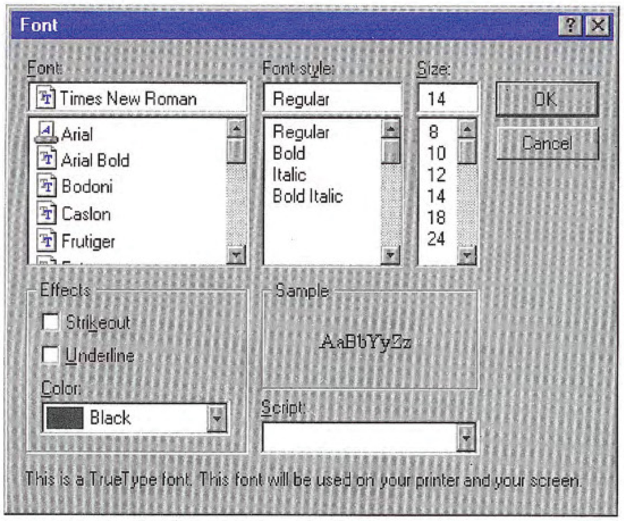
Where: \\msprint79\corpb

Comment:

Figure 8.11 The supplemental Printer dialog box

Font Dialog Box

This dialog box displays the available fonts and point sizes of the available fonts installed in the system. Your application can filter this list to show only the fonts applicable to your application. You can use the Font dialog box to display or set the font properties of a selection of text. Figure 8.12 shows the Font dialog box.



Font: Times New Roman

Arial
Arial Bold
Bodoni
Caslon
Frutiger

Font style: Regular

Regular
Bold
Italic
Bold Italic

Size: 14

8
10
12
14
18
24

□ Strikeout

□ Underline

Color: Black

Script:

This is a TrueType font. This font will be used on your printer and your screen.

Figure 8.12 The Font dialog box

Color Dialog Box

The Color dialog box (as shown in Figure 8.13) displays the available colors and includes controls that allow the user to define custom colors. You can use this control to provide an interface for users to select colors for an object.

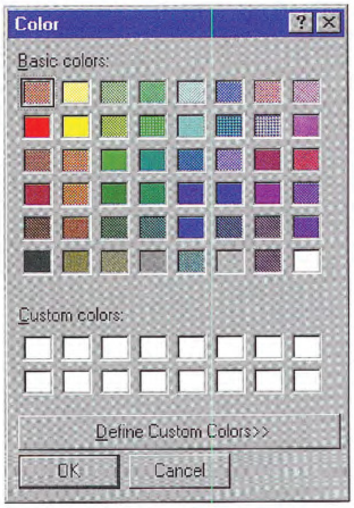
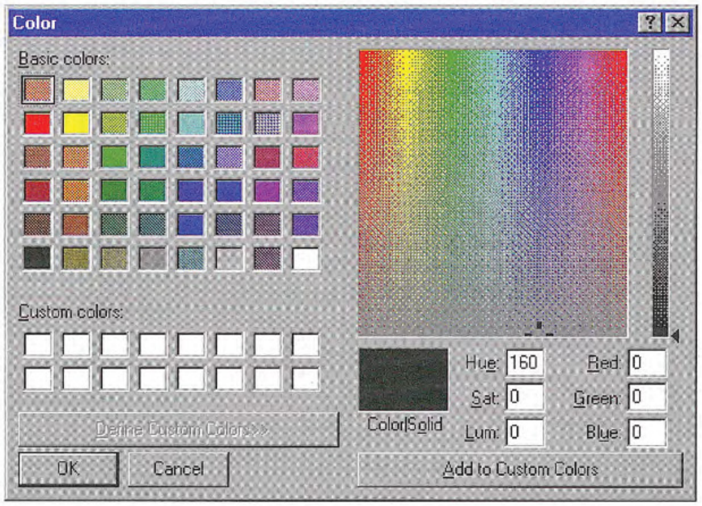


Figure 8.13 The Color dialog box (unexpanded appearance)

The Basic Colors control displays a default set of colors. The number of colors displayed here is determined by the installed display driver. The Custom Colors control allows the user to define more colors using the various color selection controls provided in the window.

Initially, you can display the dialog box as a smaller window with only the Basic Colors and Custom Colors controls and allow the user to expand the dialog box to define additional colors (as shown in Figure 8.14).



Hue: 160
Sat: 0
Lum: 0
Red: 0
Green: 0
Blue: 0

Figure 8.14 The Color dialog box (expanded)

Palette Windows

Palette windows are modeless secondary windows that present a set of controls. For example, when toolbar controls appear as a window, they appear in a palette window. Palette windows are distinguished by their visual appearance. The height of the title bar for a palette window is shorter, but it still includes only a Close button in the title area, as shown in 8.15.

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

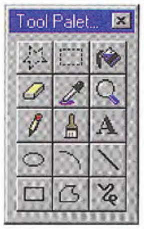


Figure 8.15 A palette window

Make the title text for a palette window the name of the command that displays the window or the name of the toolbar it represents. The system supplies default size and font settings for the title bar and title bar text for palette windows.

The title bar height and font size settings can be accessed using the **SystemParametersInfo** function. For more information about this function, see the documentation included in the Microsoft Win32 Software Development Kit (SDK).

You can define palette windows as a fixed size, or, more typically, sizable by the user. Two visual cues indicate when the window is sizable: changing the pointer image to the size pointer, and placing a Size command in the window's pop-up menu. Preserve the window's size and position so the window can be restored if it, or its associated primary window, is closed.

Like other windows, the title bar and the border areas provide an access point for the window's pop-up menu. Commands on a palette window's pop-up menu can include Close, Move, Size (if sizable), Always On Top, and Properties, as shown in Figure 8.16.

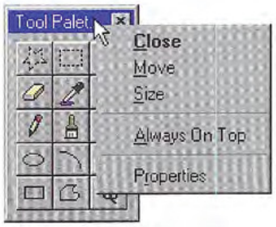


Figure 8.16 A pop-up menu for a palette window

Including the Always On Top command or property in the window's property sheet allows the user to configure the palette window to always stay at the top of the Z order of the window set of which it is a part. Turning off this option keeps the palette window within its set of related windows, but allows the user to have other windows of the set appear on top of the palette window. This feature allows the user to configure preferred access to the palette window.

You can also include a Properties command on the palette window's pop-up menu to provide an interface for allowing the user to edit properties of the window, such as the Always On Top property, or a means of customizing the content of the palette window.

Message Boxes

A message box is a secondary window that displays a message; information about a particular situation or condition. Messages are an important part of the interface for any software product. Messages that are too generic or poorly written frustrate users, increase support costs, and ultimately reflect on the quality of the product. Therefore, it is worthwhile to design effective message boxes.

However, it is even better to avoid creating situations that require you to display a message. For example, if there may be insufficient disk space to perform an operation, rather than assuming that you will display a message box, check before the user attempts the operation and disable the command.

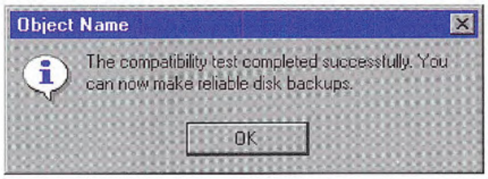
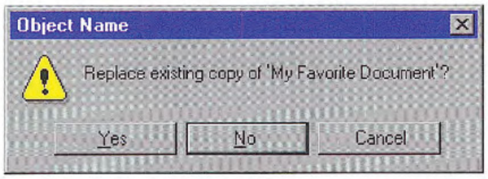
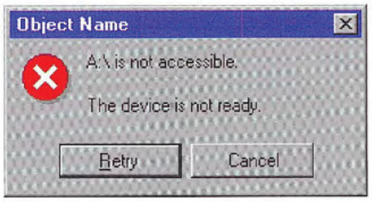
Title Bar Text

Use the title bar of a message box to appropriately identify the context in which the message is displayed — usually the name of the object. For example, if the message results from editing a document, the title text is the name of that document, optionally followed by the application name. If the message results from a nondocument object, then use the application name. Providing an appropriate identifier for the message is particularly important in the Windows multitasking environment, because message boxes might not always be the result of current user interaction. In addition, because OLE technology allows objects to be embedded, different application code may be running when the user activates the object for visual editing. Therefore, the title bar text of a message box provides an important role in communicating the source of a message. Do not use descriptive text for message box title text such as "warning" or "caution." The message symbol conveys the nature of the message. Never use the word "error" in the title text.

Message Box Types

Message boxes typically include a graphical symbol that indicates what kind of message is being presented. Most messages can be classified in one of the categories shown in Table 8.1.

Table 8.1 Message Types and Associated Symbols

Symbol	Message type	Description
ⓘ	Information	Provides information about the results of a command. Offers no user choices; the user acknowledges the message by clicking the OK button.  Object Name The compatibility test completed successfully. You can now make reliable disk backups.
⚠	Warning	Alerts the user to a condition or situation that requires the user's decision and input before proceeding, such as an impending action with potentially destructive, irreversible consequences. The message can be in the form of a question — for example, "Save changes to MyReport?".  Object Name Replace existing copy of 'My Favorite Document'?
⊗	Critical	Informs the user of a serious problem that requires intervention or correction before work can continue.  Object Name A:\ is not accessible. The device is not ready.

The system also includes a question mark message symbol. This message symbol (as shown in Figure 8.17) was used in earlier versions of Windows for cautionary messages that were phrased as a question.

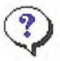


Figure 8. 17 Inappropriate message symbol

However, the message icon is no longer recommended as it does not clearly represent a type of message and the phrasing of a message as a question could apply to any message type. In addition, users can confuse the message symbol question mark with Help information. Therefore, do not use this question mark message symbol in your message boxes. The system continues to support its inclusion only for backward compatibility.

You can include your own graphics or animation in message boxes. However, limit your use of these types of message boxes and avoid defining new graphics to replace the symbols for the existing standard types.

Because a message box disrupts the user's current task, it is best to display a message box only when the window of the application displaying the message box is active. If it is not active, then the application uses its entry in the taskbar to alert the user. Once the user activates the application, the message box can be displayed. Display only one message box for a specific condition. Displaying a sequential set of message boxes tends to confuse users.

For more information about how to use the taskbar to notify the user when the application may not be active, see Chapter 10, "Integrating with the System."

You can also use message boxes to provide information or status without requiring direct user interaction to dismiss them. For example, message boxes that provide a visual representation of the progress of a particular process automatically disappear when the process is complete, as shown in Figure 8.18. Similarly, product start-up windows that identify the product name and copyright information when the application starts can be automatically removed once the application has loaded. In these situations, you do not need to include a message symbol. Use this technique only for noncritical, informational messages, as some users may not be able to read the message within the short time it is displayed.

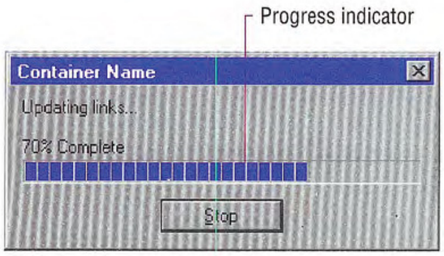


Figure 8.18 A progress message box

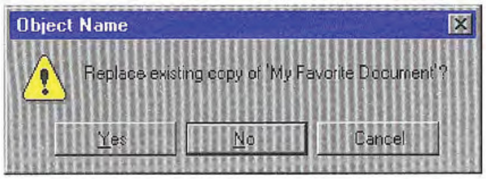
Command Buttons in Message Boxes

Typically, message boxes contain only command buttons as the appropriate responses or choices offered to the user. Designate the most frequent or least destructive option as the default command button. Command buttons allow the message box interaction to be simple and efficient. If you need to add other types of controls, always consider the potential increase in complexity.

If a message requires no choices to be made but only acknowledgment, use an OK button — and, optionally, a Help button. If the message requires the user to make a choice, include a command button for each option. The clearest way to present the choices is to state the message in the form of a question and provide a button for each response. When possible, phrase the question to permit Yes or No answers, represented by Yes and No command buttons. If these choices are too ambiguous, label the command buttons with the names of specific actions — for example, "Save" and "Delete."

You can include command buttons in a message box that correct the action that caused the message box to be displayed. For example, if the message box indicates that the user must switch to another application window to take corrective action, you can include a button that switches the user to that application window. Be sure, however, to make the result of any such button's action very clear.

Some situations may require offering the user not only a choice between performing or not performing an action, but an opportunity to cancel the process altogether. In such situations, use a Cancel button, as shown in Figure 8.19. Be sure, however, to make the result of any such button's action very clear.



Replace existing copy of 'My Favorite Document'?

Figure 8.19 Message box choices

When using Cancel as a command button in a message box, remember that to users, Cancel implies restoring the state of the process or task that started the message. If you use Cancel to interrupt a process and the state cannot be restored, use Stop instead.

Message Box Text

The message text you include in a message box should be clear, concise, and in terms that the user understands. This usually means using no technical jargon or system-oriented information.

In addition, observe the following guidelines for your message text:

- State the problem, its probable cause (if possible), and what the user can do about it — no matter how obvious the solution may seem to be. For example, instead of "Insufficient disk space," use "'Sample Document' could not be saved, because the disk is full. Try saving to another disk or freeing up space on this disk."

- Consider making the solution an option offered in the message. For example, instead of "One or more of your lines are too long. The text can only be a maximum of 60 characters wide," you might say, "One or more of your lines are too long. Text can be a maximum of 60 characters in Portrait mode or 90 characters wide in Landscape. Do you want to switch to Landscape mode now?" Offer Yes and No as the choices.

- Avoid using unnecessary technical terminology and overly complex sentences. For example, “picture” can be understood in context, whereas “picture metafile” is a rather technical concept.

- Avoid phrasing that blames the user or implies user error. For example, use “Cannot find filename” instead of “Filename error.” Avoid the word “error” altogether.

- Make messages as specific as possible. Avoid mapping more than two or three conditions to a single message. For example, there may be several reasons why a file cannot be opened; provide a specific message for each condition.

- Avoid relying on default system-supplied messages, such as MS-DOS® extended error messages and Kernel INT 24 messages; instead, supply your own specific messages wherever possible.

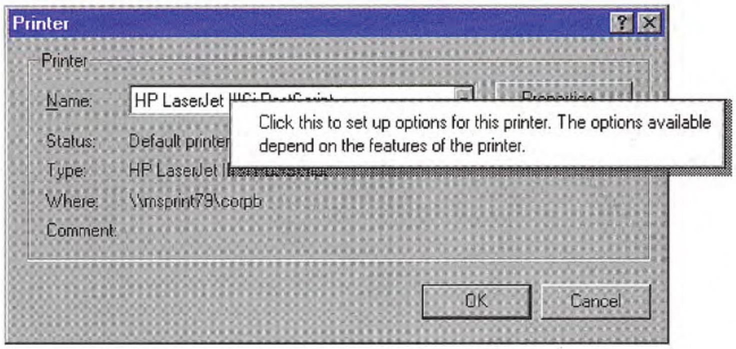
- Be brief, but complete. Provide only as much background information as necessary. A good rule of thumb is to limit the message to two or three lines. If further explanation is necessary, provide this through a command button that opens a Help window.

You may also include a message identification number as part of the message text for each message for support purposes. However, to avoid interrupting the user’s ability to quickly read a message, place such a designation at the end of the message text and not in the title bar text.

Pop-up Windows

Use pop-up windows to display additional information when an abbreviated form of the information is the main presentation. For example, you could use a pop-up window to display the full path for a field or control, when an entire path cannot be presented and must be abbreviated. Pop-up windows are also used to provide context-sensitive Help information, as shown in Figure 8.20.

For more information about using pop-up windows for Help information, see Chapter 12, "User Assistance."



Name: HP LaserJet

Status: Default printer

Type: HP LaserJet I

Where: \\msprint79\corpb

Comment:

Click this to set up options for this printer. The options available depend on the features of the printer.

Figure 8.20 A context-sensitive Help pop-up window

Tooltips are another example of a pop-up window used to display contextual information, by providing the names for controls in toolbars. The writing tool is also another example of the use of a pop-up window.

How pop-up windows are displayed depends on their use, but the typical means is by the user either pointing or clicking with mouse button 1 (for pens, tapping), or an explicit command. If you use pointing as the technique to display a pop-up window, display the window after a time-out. The system automatically handles time-outs if you use the standard tooltip controls. If you are providing your own implementation, you can use the current double-click speed setting as a metric for displaying and removing the pop-up window.

If you use clicking to display a pop-up window, change the pointer as feedback to the user indicating that the pop-up window exists and requires a click. From the keyboard, you can use the Select key (SPACEBAR) to open and close the window.

Part III

Design Specifications and Guidelines

CHAPTER 9

Window Management

User tasks can often involve working with different types of information, contained in more than one window or view. There are different techniques that you can use to manage a set of windows or views. This chapter covers some common techniques and the factors to consider for selecting a particular model.

Single Document Window Interface

In many cases, the interface of an object or application can be expressed using a single primary window with a set of supplemental secondary windows. The desktop and taskbar provide management of primary windows. Opening a window puts it at the top of the Z order and places an entry on the taskbar, making it easier for users to switch between windows without having to shuffle or reposition them.

By supporting a single instance model where you activate an existing window (within the same desktop) if the user reopens the object, you make single primary windows more manageable, and reduce the potential confusion for the user. This also provides a data-centered, one-to-one relationship between an object and its window.

In addition, Microsoft OLE supports the creation of compound documents or other types of information containers. Using these constructs, the user can assemble a set of different types of objects for a specific purpose within a single primary window, eliminating the necessity of displaying or editing information in separate windows.

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

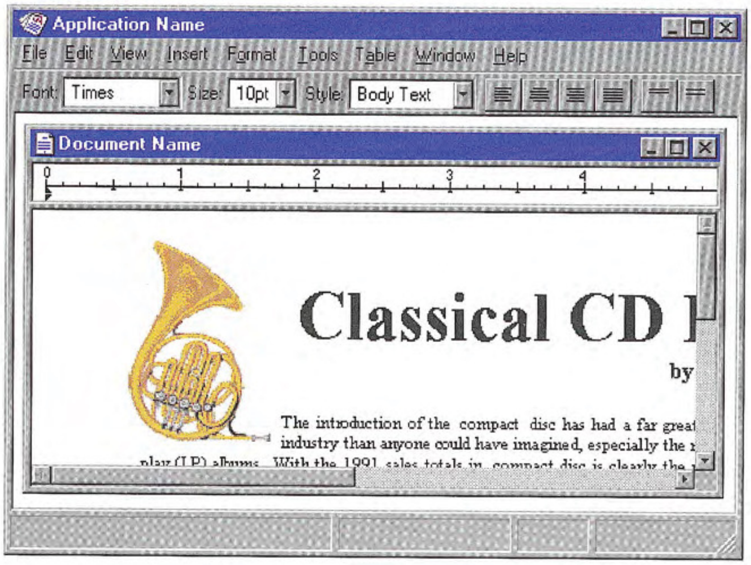
Some types of objects, such as device objects, may not even require a primary window and use only a secondary window for viewing and editing their properties. When this occurs, do not include the Open command in the menu for the object; instead, replace it with a Properties command, defined as the object's default command.

It is also possible for an object to have no windows; an icon is its sole representation. In this very rare case, make certain that you provide an adequate set of menu commands to allow a user to control its activity.

Multiple Document Interface

For some tasks, the taskbar may not be sufficient for managing a set of related windows; for example, it can be more effective to present multiple views of the same data or multiple views of related data in windows that share interface elements. You can use *multiple document interface* (MDI) for this kind of situation.

The MDI technique uses a single primary window, called a *parent window*, to visually contain a set of related *document* or *child windows*, as shown in Figure 9.1. Each child window is essentially a primary window, but is constrained to appear only within the parent window instead of on the desktop. The parent window also provides a visual and operational framework for its child windows. For example, child windows typically share the menu bar of the parent window and can also share other parts of the parent's interface, such as a toolbar or status bar. You can change these to reflect the commands and attributes of the active child window.



Classical CD I

by

The introduction of the compact disc has had a far great
industry than anyone could have imagined, especially the
play (LP) albums. With the 1991 sales totals in compact disc is clearly the

Figure 9.1 An MDI parent and child window

Secondary windows — such as dialog boxes, message boxes, or property sheets — displayed as a result of interaction within the MDI parent or child, are typically not contained or clipped by the parent window. These windows should be activated and displayed following the common conventions for secondary windows associated with a primary window, even if they apply to individual child windows.

For the title bar of an MDI parent window, include the icon and name of the application or the object that represents the work area displayed in the parent window. For the title bar of a child window, include the icon representing the document or data file type and its filename. Also support pop-up menus for the window and the title bar icon for both the parent window and any child windows.

For more information about the interaction between a primary window and its secondary windows, see Chapter 6, "Windows," and Chapter 8, "Secondary Windows."

Opening and Closing MDI Windows

The user starts an MDI application either by directly opening the application or by opening a document (or data file) of the type supported by the MDI application. If directly opening an MDI document, the MDI parent window opens first and then the child window for the file opens within it. To support the user opening other documents associated with the application, include an interface, such as an Open dialog box.

When the user directly opens an MDI document outside the interface of its MDI parent window — for example, by double-clicking the file — if the parent window for the application is already open, open another instance of the MDI parent window rather than the document's window in the existing MDI parent window. Although the opening of the child window within the existing parent window can be more efficient, the opening of the new window can disrupt the task environment already set up in that parent window. For example, if the newly opened file is a macro, opening it in the opened parent window could inadvertently affect other documents open in that window. If the user wishes to open a file as part of the set in a particular parent MDI window, the commands within that window provide that support.

Because MDI child windows are primary windows, support closing them following the same conventions for primary windows by including a Close button in their title bars and a Close command in their pop-up menu for the windows. When the user closes a child window, any unsaved changes are processed following these common conventions for primary windows. Do not close its parent window, unless the parent window does not provide context or operations without an open child window.

When the user closes the parent window, close all of its child windows. Where possible, preserve the state of a child window, such as its size and position within the parent window; restore the state when the user reopens the file.

Moving and Sizing MDI Windows

MDI allows the user to move or hide the child windows as a set by moving or minimizing the parent window. When the user moves an MDI parent window, maintain the relative positions of the open child windows within the parent window. Moving a child window constrains it to its parent window; in some cases, the size of the parent window's interior area may result in clipping a child window. Optionally, you can support automatic resizing of the parent window when the user moves or resizes a child window either toward or away from the edge of the parent window.

The recommended visual appearance of a minimized child window in Microsoft Windows is now that of a window that has been sized down to display only part of its title area and its border. This avoids potential confusion between minimized child window icons and icons that represent objects.

Although an MDI parent window minimizes as an entry on the taskbar, MDI child windows minimize within their parent window, as shown in Figure 9.2.

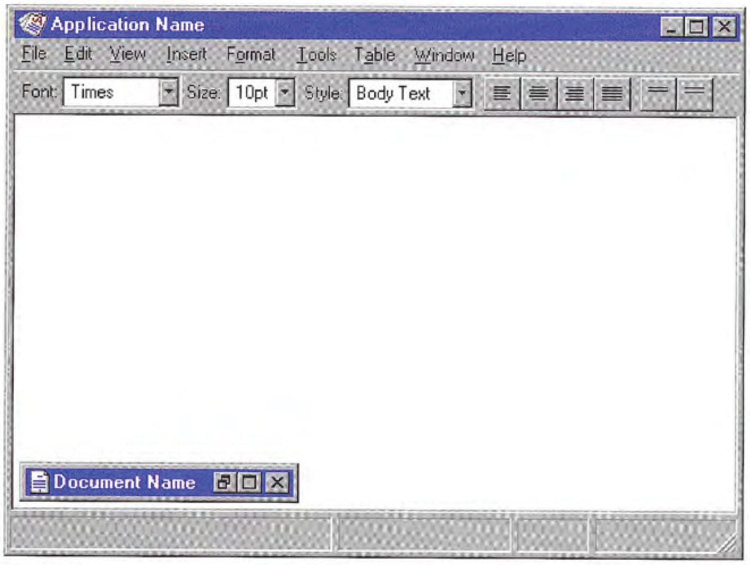
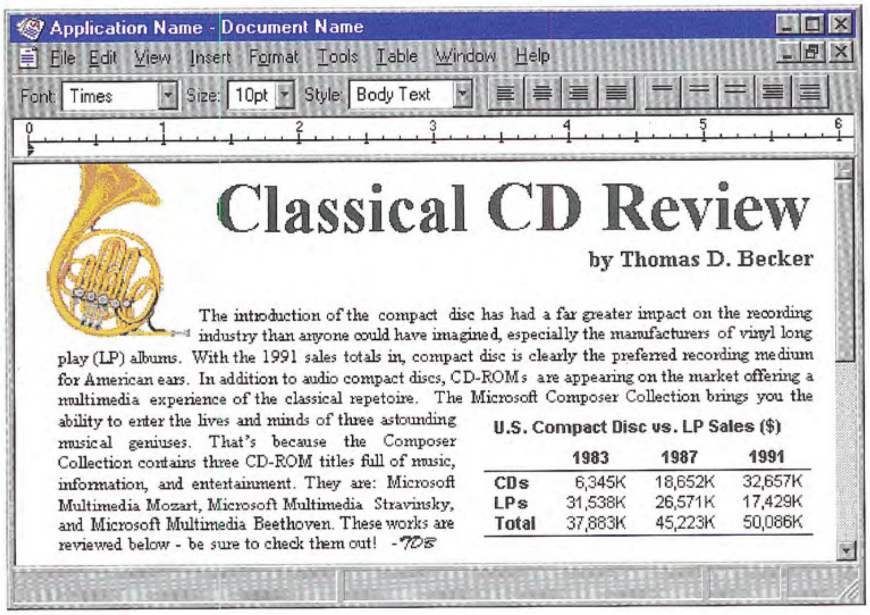


Figure 9.2 A minimized MDI child window

When the user maximizes an MDI parent window, expand the window to its maximum size, like any other primary window. When the user maximizes an MDI child window, also expand it to its maximum size. When this size exceeds the interior of its parent window, merge the child window with its parent window. The child window's title bar icon, Restore button, Close button, and Minimize button (if supported) are placed in the menu bar of the parent window in the same relative position as in the title bar of the child window, as shown in Figure 9.3. Append the child window title text to the parent window's title text.


Application Name - Document Name
File Edit View Insert Format Tools Table Window Help
Font: Times
Size: 10pt
Style: Body Text
0
1
2
3
4
5
6

Classical CD Review

by Thomas D. Becker

The introduction of the compact disc has had a far greater impact on the recording industry than anyone could have imagined, especially the manufacturers of vinyl long play (LP) albums. With the 1991 sales totals in, compact disc is clearly the preferred recording medium for American ears. In addition to audio compact discs, CD-ROMs are appearing on the market offering a multimedia experience of the classical repetoire. The Microsoft Composer Collection brings you the ability to enter the lives and minds of three astounding musical geniuses. That's because the Composer Collection contains three CD-ROM titles full of music, information, and entertainment. They are: Microsoft Multimedia Mozart, Microsoft Multimedia Stravinsky, and Microsoft Multimedia Beethoven. These works are reviewed below - be sure to check them out! -TDB

U.S. Compact Disc vs. LP Sales ($)

	1983	1987	1991
CDs	6,345K	18,652K	32,657K
LPs	31,538K	26,571K	17,429K
Total	37,883K	45,223K	50,086K

Figure 9.3 A maximized MDI child window

If the user maximizes one child window and it merges with the parent window and then switches to another, display that window as maximized. Similarly, when the user restores one child window from its maximized state, restore all other child windows to their previous sizes.

Switching Between MDI Child Windows

Apply the same common mouse conventions for activating and switching between primary windows for MDI child windows. CTRL+F6 and CTRL+TAB (and SHIFT+ modified combinations to cycle backwards) are the recommended keyboard shortcuts for switching between child windows. In addition, include a Window menu on the menu bar of the parent window with commands for switching between child windows and managing or arranging the windows within the MDI parent window — such as Tile or Cascade.

When the user switches child windows, you can change the interface of the parent window — such as its menu bar, toolbar, or status bar — to appropriately reflect the commands that apply to that child window. However, provide as much consistency as possible, keeping constant any menus that represent the document files and control the application or overall parent window environment, such as the File menu or the Window menu.

MDI Alternatives

MDI does have its limitations. MDI reinforces the visibility of the application as the primary focus for the user. Although the user can start an MDI application by directly opening one of its document or data files, to work with multiple documents within the same MDI parent window, the user uses the application's interface for opening those documents.

When the user opens multiple files within the same MDI parent window, the storage relationship between the child windows and the objects being viewed in those windows is not consistent. That is, although the parent window provides visual containment for a set of child windows, it does not provide containment for the files those windows represent. This makes the relationship between the files and their windows more abstract, making MDI more challenging for beginning users to learn.

Similarly, because the MDI parent window does not actually contain the objects opened within it, MDI cannot support an effective design for persistence. When the user closes the parent window and then reopens it, the context cannot be restored because the application state must be maintained independently from that of the files last opened in it.

MDI can make some aspects of the OLE interface unintentionally more complex. For example, if the user opens a text document in an MDI application and then opens a worksheet embedded in that text document, the task relationship and window management breaks down, because the embedded worksheet's window does not appear in the same MDI parent window.

Finally, the MDI technique of managing windows by confining child windows to the parent window can be inconvenient or inappropriate for some tasks, such as designing with window or form layout tools. Similarly, the nested nature of child windows may make it difficult for the user to differentiate between a child window in a parent window versus a primary window that is a peer with the parent window, but positioned on top.

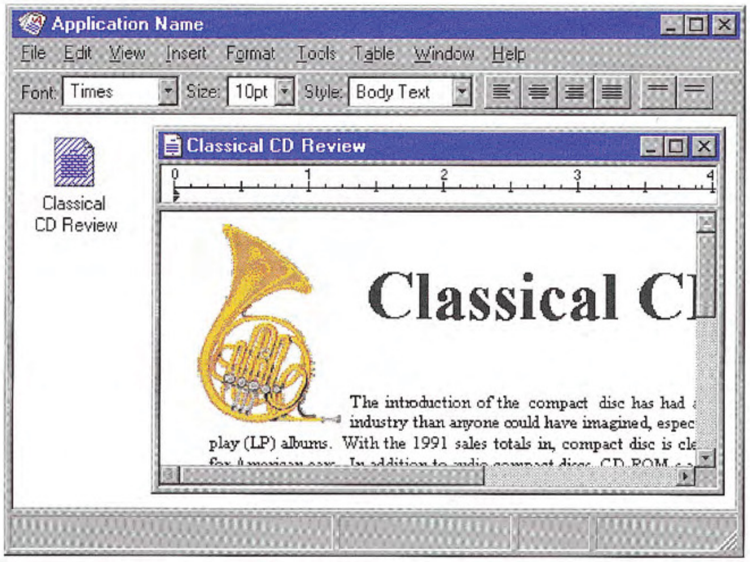
Although MDI provides useful conventions for managing a set of related windows, it is not the only means of supporting task management. Some of its window management techniques can be applied in some alternative designs. The following — workspaces, workbooks, and projects — are examples of some possible design alternatives. They present a single window design model, but in such a way that preserves some of the window and task management benefits found in MDI.

Although these examples suggest a form of containment of multiple objects, you can also apply some of these designs to display multiple views of the same data. Similarly, these alternatives may provide greater flexibility with respect to the types of objects that they contain. However, as with any container, you can define your implementation to hold and manage only certain types of objects. For example, an appointment book and an index card file are both containers that organize a set of information but may differ in the way they display that information and the type of information they manage. Whether you define a container to hold the same or different types of objects depends on the design and purpose of the container.

The following examples illustrate alternatives of data-centered window or task management. They are not exclusive of other possible designs. They are intended only as suggestive possibilities, rather than standard constructs. As a result, the system does not include these constructs and provides no explicit programming interfaces. In addition, some specific details are left to you to define.

Workspaces

A *workspace* shares many of the characteristics of MDI, including the association and management of a set of related windows within a parent window, and the sharing of the parent window's interface elements, such as menus, toolbars, and status bar. Figure 9.4 shows an example of a workspace.



Classical Cl

The introduction of the compact disc has had
industry than anyone could have imagined, espec
play (LP) albums. With the 1991 sales totals in, compact disc is cl
for American ears. In addition to audio compact discs, CD-ROM

Figure 9.4 Example of a workspace design

Workspaces as a Container

Based on the metaphor of a work area, like a table, desktop, or office, a workspace differs from an MDI by including the concept of containment. Objects contained or stored in the workspace can be presented in the same way files appear in folders. However, objects within a workspace open as child windows within the workspace parent window. In this way, a workspace's behavior is similar to that of the desktop, except that a workspace itself is an object that can be displayed as an icon and opened into a window. To have an object's window appear in the workspace, the object must reside there.

The actual storage mechanism you use depends on the type of container you implement. The content of the parent window can represent a single file, or you can devise your own mechanism to map the content into the file system. Consider using OLE in your implementation to facilitate interaction between your workspace, the shell, and other applications. For example, you may want to support the user moving objects from the workspace into other containers, such as the desktop and folders. However, if you do, when the user opens the object, it should appear in its own window, not the workspace window — with its interface elements, such as a menu bar — also appearing within its own window.

The workspace is an object itself and therefore you should define its specific commands and properties. You can also include commands for creating new objects within the workspace and, optionally, a Save All command that saves the state of all the objects opened in the workspace.

Workspaces for Task Grouping

Because a workspace visually contains and constrains the icons and windows of the objects placed in it, you can define workspaces to allow the user to organize a set of objects for particular tasks. Like MDI, this makes it easy for the user to move or switch to a set of related windows as a set.

Also similar to MDI, the child windows of objects opened in the workspace can share the interface of the parent window. For example, if the workspace includes a menu bar, the windows of any objects contained within the workspace share the menu bar. If the workspace does not have a menu bar, or if you provide an option for the user to hide the menu bar, the menu bar should appear within the document's child window. The parent window can also provide a framework for sharing toolbars and status bars.

Window Management in a Workspace

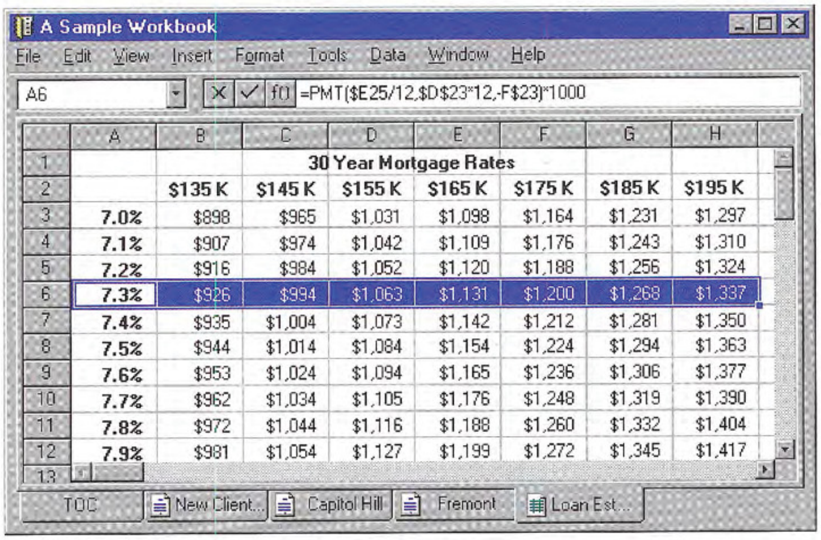
A workspace manages windows using the same conventions as MDI. When a workspace closes, all the windows within it close. You should retain the state of these windows, for example, their size and position within the workspace, so you can restore them when the user reopens the workspace.

Like most primary windows, when the user minimizes the workspace window, the window disappears from the screen but its entry remains on the taskbar. Minimized windows of icons opened within the workspace have the same behavior and appearance as minimized MDI child windows. Similarly, maximizing a window within a workspace can follow the MDI technique: if the window's maximized size exceeds the size of the workspace window, the child window merges with the workspace window and its title bar icon and window buttons appear in the menu bar of the workspace window.

A workspace should provide a means of navigating between the child windows within a workspace, such as listing the open child windows on a Window drop-down menu and on the pop-up menu for the parent window, in addition to direct window activation.

Workbooks

A *workbook* is another alternative for managing a set of views — one which uses the metaphor of a book or notebook instead of a work area. Within the workbook, you present views of objects as sections within the workbook's primary window rather than in individual child windows. Figure 9.5 illustrates one possible way of presenting a workbook.



A6

=PMT($E25/12,$D$23*12,-F$23)*1000

	A	B	C	D	E	F	G	H
1			30 Year Mortgage Rates					
2		$135 K	$145 K	$155 K	$165 K	$175 K	$185 K	$195 K
3	7.0%	$898	$965	$1,031	$1,098	$1,164	$1,231	$1,297
4	7.1%	$907	$974	$1,042	$1,109	$1,176	$1,243	$1,310
5	7.2%	$916	$984	$1,052	$1,120	$1,188	$1,256	$1,324
6	7.3%	$926	$994	$1,063	$1,131	$1,200	$1,268	$1,337
7	7.4%	$935	$1,004	$1,073	$1,142	$1,212	$1,281	$1,350
8	7.5%	$944	$1,014	$1,084	$1,154	$1,224	$1,294	$1,363
9	7.6%	$953	$1,024	$1,094	$1,165	$1,236	$1,306	$1,377
10	7.7%	$962	$1,034	$1,105	$1,176	$1,248	$1,319	$1,390
11	7.8%	$972	$1,044	$1,116	$1,188	$1,260	$1,332	$1,404
12	7.9%	$981	$1,054	$1,127	$1,199	$1,272	$1,345	$1,417
13								

Figure 9.5 Example of a workbook design

For a workbook, you can use tabs to serve as a navigational interface to move between different sections. Locate the tabs as best fits the content and organization of the information you present. Each section represents a view of data, which could be an individual document. Unlike a folder or workspace, a workbook may be better suited for ordered content; that is, where the order of the sections has significance. In addition, you can optionally include a special section listing the content of the workbook, like a table of contents. This view can also be included as part of the navigational interface for the workbook.

A workbook shares an interface similar to an MDI parent window with all of its child windows maximized. The sections can share the parent window's interface elements, such as the menu bar and status bar. When the user switches sections within the workbook, you can change the menu bar so that it applies to the current object. When the user closes a workbook, follow the common conventions for handling unsaved edits or unapplied transactions when any primary window closes.

Consider supporting OLE to support transfer operations so the user can move, copy, and link objects into the workbook. You may also want to provide an Insert command that allows the user to create new objects, including a new tabbed section in the workbook. You can also include a Save All command, which saves any uncommitted changes or prompts the user to save or discard those changes.

Projects

A *project* is another window management technique that provides for association of a set of objects and their windows, but without visually containing the windows. A project is similar to a folder in that the icons contained within it can be opened into windows that are peers with the parent window. As a result, each child window can also have its own entry on the taskbar. Unlike a folder, a project provides window management for the windows of its content. For example, when the user opens a document in a folder and then closes the folder, it has no effect on the window of the opened document. However, when the user closes a project window, all the child windows of objects contained in the project also close. In addition, when the user opens a project window, this action should restore the windows of objects contained within it to their previous state.

Similarly, to facilitate window management, when the user minimizes a project window, you may want to minimize any windows of the objects the project contains. Taskbar entries for these windows remain. Allow the user to restore a specific child window without restoring the project window or other windows within the project. In addition, support the user independently minimizing any child window without affecting the project window. Figure 9.6 shows an example of a project.

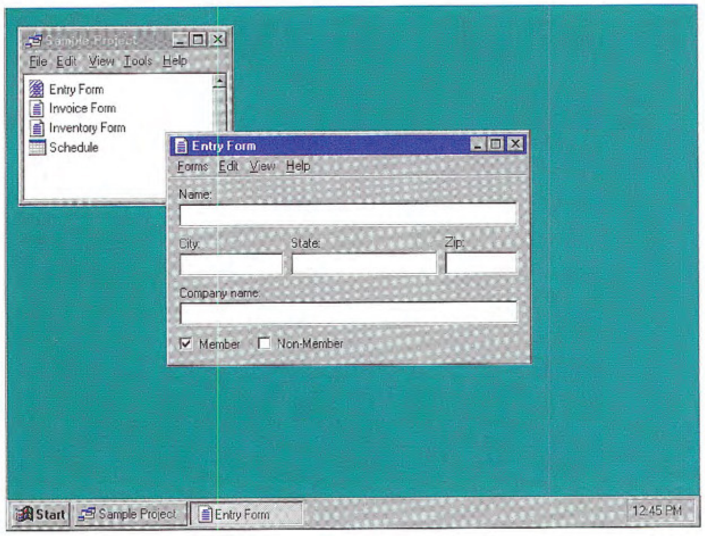


Figure 9.6 Example of a project design

The windows of objects stored in the project do not share the menu bar or other areas within the project window. Instead, include the interface elements for each object in its own window. However, you can provide toolbar palette windows that can be shared among the windows of the objects in the project.

Just as in workspaces and workbooks, a project should include commands for creating new objects within the project, for transferring objects in and out of the project, and for saving any changes for the objects stored in the project. In addition, a project should include commands and properties for the project object itself.