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

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

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

Common Pop-up Menus

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

The Window Pop-up Menu

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

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

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

Petitioner Exhibit 1031, Page 151

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

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

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

Icon Pop-up Menus

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

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

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

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

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

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

Table 7.1 Application File Icon Pop-up Menu Commands

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

Petitioner Exhibit 1031, Page 153

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

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

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

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

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

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

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

Petitioner Exhibit 1031, Page 154

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

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

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

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

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

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

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

Cascading Menus

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

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

Petitioner Exhibit 1031, Page 155

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

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

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

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

Menu Titles

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

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

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

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

Petitioner Exhibit 1031, Page 156

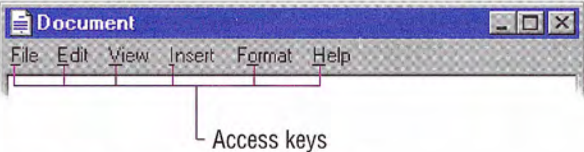


Figure 7.4 Access keys in a menu bar

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

Menu Items

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

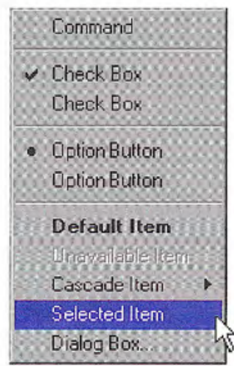


Figure 7.5 Formats for different menu items

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

Petitioner Exhibit 1031, Page 157

Figure 7.6 Inappropriate separator

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

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

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

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

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

Types of Menu Items

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

Petitioner Exhibit 1031, Page 158

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

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

Menu items for state settings can be independent or interdependent:

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

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

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

Petitioner Exhibit 1031, Page 159

However, if the two states of the setting are not obvious opposites, use a pair of alternating menu item names to indicate the two states. For example, a naive user might guess that the opposite of a menu item called Full Duplex is Empty Duplex. Because of this ambiguity, pair the command with the alternative name Half Duplex, rather using a mark to indicate the alternative states, and consider the following guidelines for how to display those alternatives:

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

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

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

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

Menu Item Labels

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

Petitioner Exhibit 1031, Page 160

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

- Define unique item names within a menu. However, item names can be repeated in different menus to represent similar or different actions.

- Use a single word or multiple words, but keep the wording brief and succinct. Verbose menu item names can make it harder for the user to scan the menu.

- Define unique access keys for each menu item within a menu. This provides the user direct keyboard access to the menu item. The guidelines for selecting an access key for menu items are the same as for menu titles, except that the access key for a menu item can also be a number included at the beginning of the menu item name. This is useful for menu items that vary, such as file-names. Where possible, also define consistent access keys for common commands.

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

- Follow book title capitalization rules for menu item names. For English language versions, capitalize the first letter of every word, except for articles, conjunctions, and prepositions that occur other than at the beginning or end of a multiple-word name. For example, the following menu names are correct: New Folder, Go To, Select All, and Table of Contents.

- Avoid formatting individual menu item names with different text properties. Even though these properties illustrate a particular text style, they also may make the menu cluttered, illegible, or confusing. For example, it may be difficult to indicate an access key if an entire menu entry is underlined.

Shortcut Keys in Menu Items

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

Petitioner Exhibit 1031, Page 161

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

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

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

Controls

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

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

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

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

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

Petitioner Exhibit 1031, Page 162

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

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

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

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

Buttons

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

Command Buttons

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

Figure 7.7 Command buttons

Petitioner Exhibit 1031, Page 163

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

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

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

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

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

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

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

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

Petitioner Exhibit 1031, Page 164

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

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

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

Table 7.4 Command Button Appearance

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

Petitioner Exhibit 1031, Page 165

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

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

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

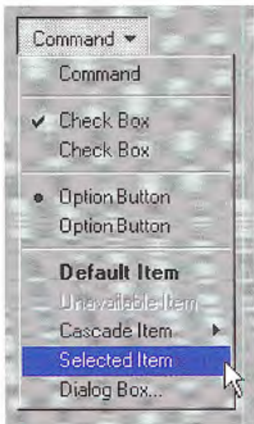


Figure 7.8 A menu button

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

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

Petitioner Exhibit 1031, Page 166

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

Option Buttons

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

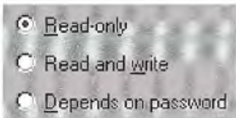


Figure 7.9 A set of option buttons

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

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

Petitioner Exhibit 1031, Page 167

Figure 7.10 Option buttons with mixed-value appearance

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

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

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

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

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

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

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

Petitioner Exhibit 1031, Page 168

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

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

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

Check Boxes

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

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

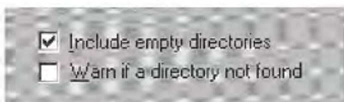


Figure 7.11 A set of check boxes

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

Petitioner Exhibit 1031, Page 169

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

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

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

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

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

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

Figure 7.12 A mixed-value check box (magnified)

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

List Boxes

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

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

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

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

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

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

Petitioner Exhibit 1031, Page 171

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.

Petitioner Exhibit 1031, Page 172

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.

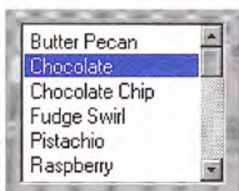


Figure 7.13 A single selection list box

Petitioner Exhibit 1031, Page 173

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.

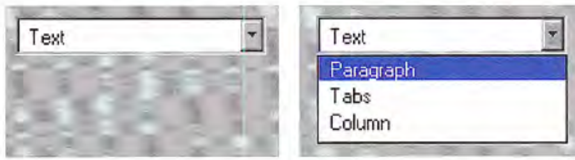


Figure 7.14 A drop-down list box (closed and opened state)

Petitioner Exhibit 1031, Page 174

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.

Petitioner Exhibit 1031, Page 175

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."

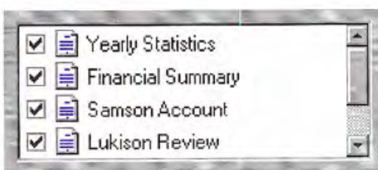


Figure 7.15 A multiple selection list box

Petitioner Exhibit 1031, Page 176

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.

Petitioner Exhibit 1031, Page 177

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.

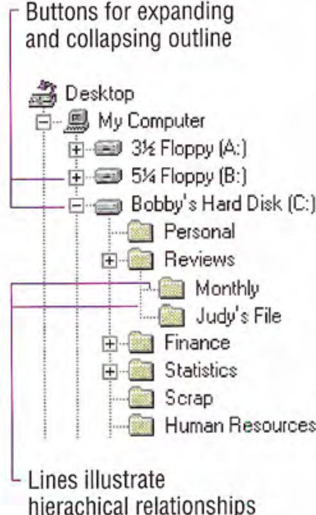


Figure 7.16 A tree view control

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.

Petitioner Exhibit 1031, Page 178

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."

Petitioner Exhibit 1031, Page 179

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.

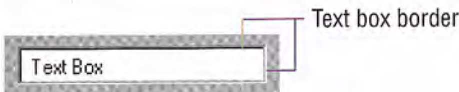


Figure 7.17 A standard text box

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.

Petitioner Exhibit 1031, Page 180

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.

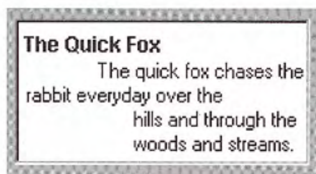


Figure 7.18 A rich-text box

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.

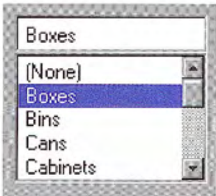


Figure 7.19 A combo box

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.

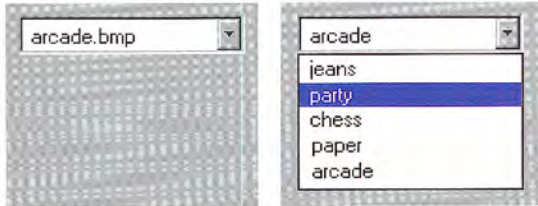


Figure 7.20 A drop-down combo box (closed and opened state)

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.

Petitioner Exhibit 1031, Page 182

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.

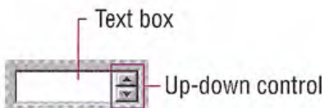


Figure 7.21 A spin box

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.

Petitioner Exhibit 1031, Page 183

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."

Petitioner Exhibit 1031, Page 184

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.

Petitioner Exhibit 1031, Page 185

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.

Column heading

Column part

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.

Petitioner Exhibit 1031, Page 186

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.

Petitioner Exhibit 1031, Page 187

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.

Petitioner Exhibit 1031, Page 188

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.

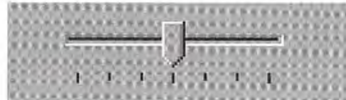


Figure 7.26 A slider

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.

Petitioner Exhibit 1031, Page 189

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.

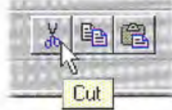


Figure 7.28 A tooltip control

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.

Petitioner Exhibit 1031, Page 190

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.

Petitioner Exhibit 1031, Page 191

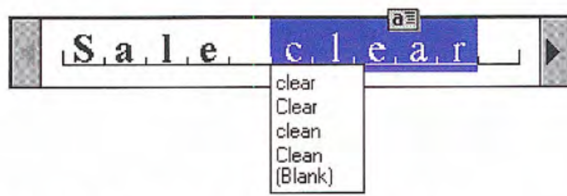


Figure 7.30 A single line boxed edit control

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

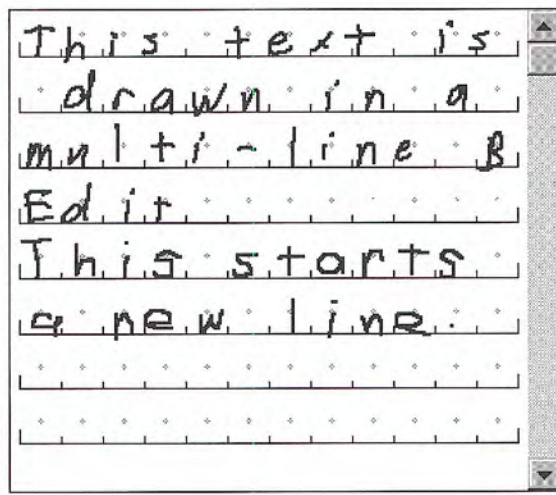


Figure 7.31 A multiple line boxed edit control

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.

Petitioner Exhibit 1031, Page 192

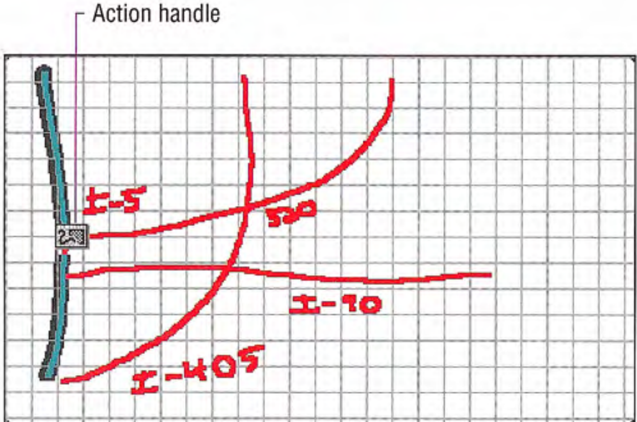


Figure 7.32 An ink edit control

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.

Petitioner Exhibit 1031, Page 193

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.

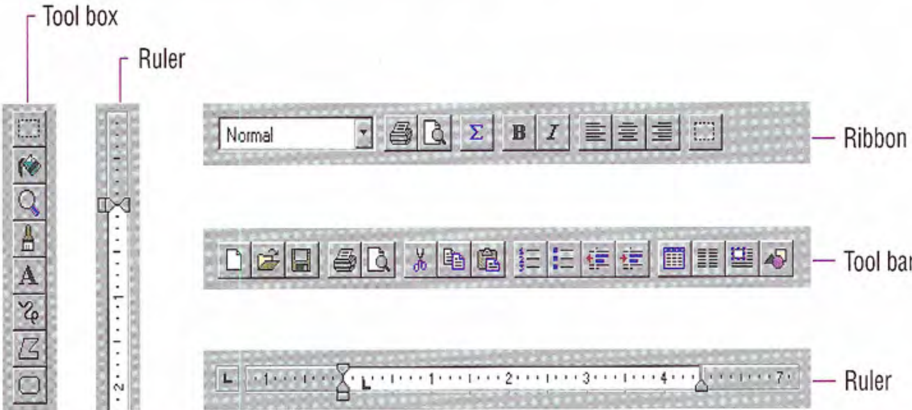


Figure 7.34 Examples of toolbars

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."

Petitioner Exhibit 1031, Page 194

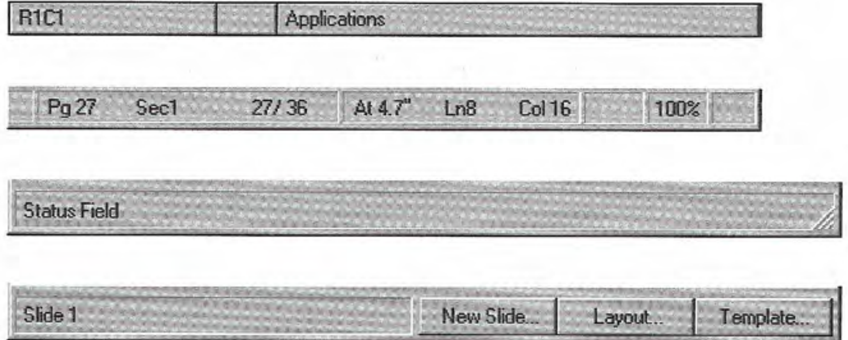


Figure 7.35 Examples of status bars

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.

Petitioner Exhibit 1031, Page 195

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.

Petitioner Exhibit 1031, Page 196

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.

Petitioner Exhibit 1031, Page 197

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

Petitioner Exhibit 1031, Page 198

(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

Petitioner Exhibit 1031, Page 199

(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."

Petitioner Exhibit 1031, Page 200

CHAPTER

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.

Petitioner Exhibit 1031, Page 201

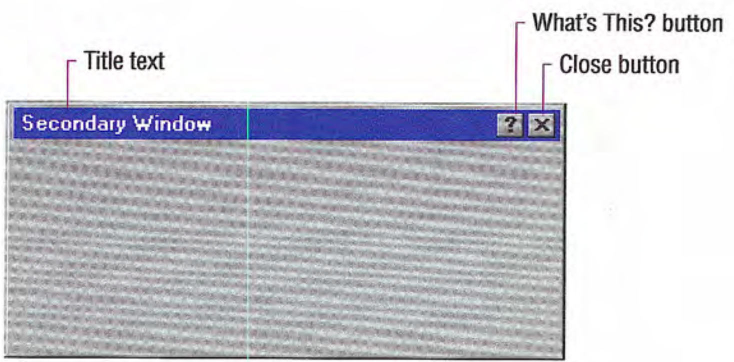


Figure 8.1 A secondary window

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.

Petitioner Exhibit 1031, Page 202

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

Petitioner Exhibit 1031, Page 203

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.

Petitioner Exhibit 1031, Page 204

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.

Petitioner Exhibit 1031, Page 205

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.

Petitioner Exhibit 1031, Page 206

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.

Petitioner Exhibit 1031, Page 207

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.

Petitioner Exhibit 1031, Page 208

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.

Petitioner Exhibit 1031, Page 209

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.

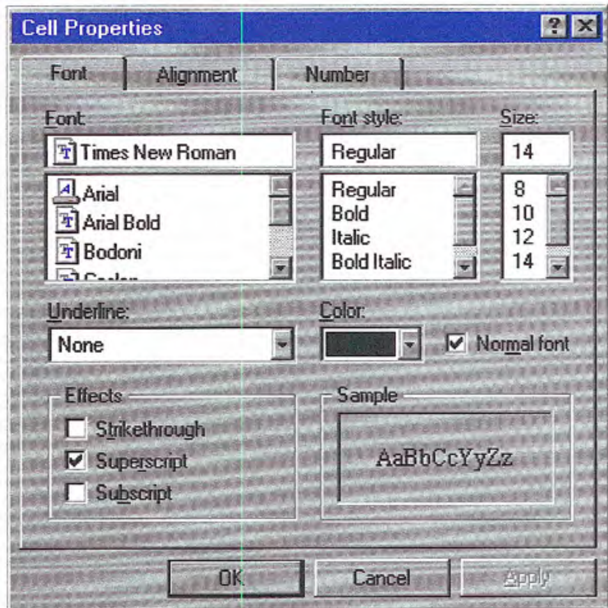


Figure 8.2 A property sheet with tabbed pages

Petitioner Exhibit 1031, Page 210

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.

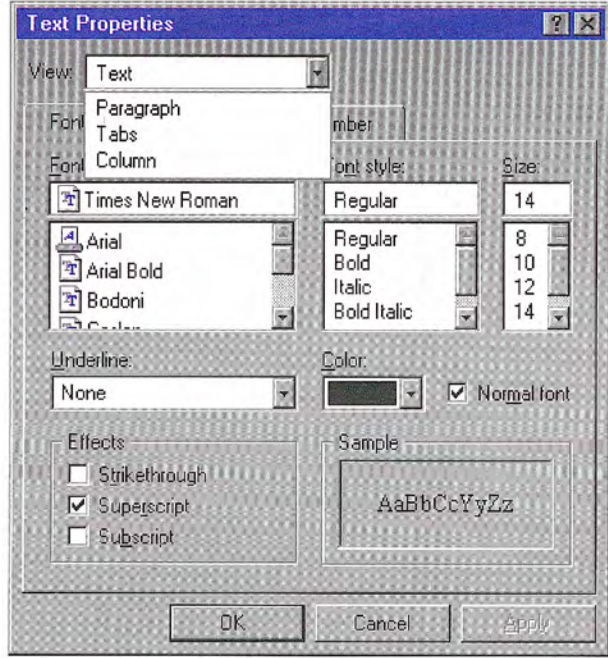


Figure 8.3 A drop-down list for access to hierarchical property sets

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.

Petitioner Exhibit 1031, Page 212

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.

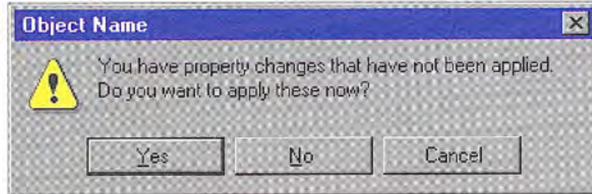


Figure 8.4 Prompting for pending property changes

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.

Petitioner Exhibit 1031, Page 213

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.

Petitioner Exhibit 1031, Page 214

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.

Petitioner Exhibit 1031, Page 215

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.

Petitioner Exhibit 1031, Page 216

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.

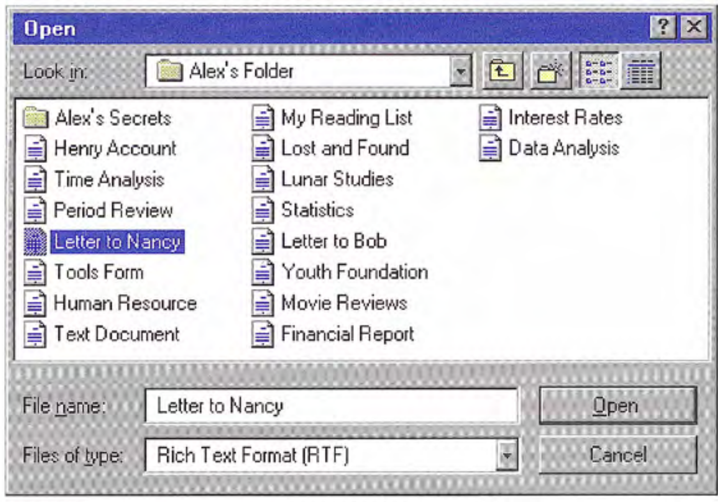


Figure 8.6 The Open dialog box

Petitioner Exhibit 1031, Page 217

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

Petitioner Exhibit 1031, Page 218

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.

Petitioner Exhibit 1031, Page 219

- 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.

Petitioner Exhibit 1031, Page 220

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.

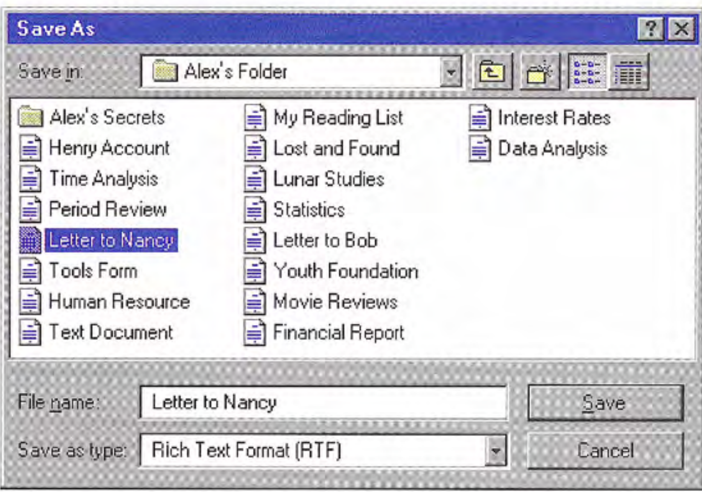


Figure 8.7 The Save As dialog box

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

Petitioner Exhibit 1031, Page 221

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.

Petitioner Exhibit 1031, Page 222

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.

Petitioner Exhibit 1031, Page 223

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.

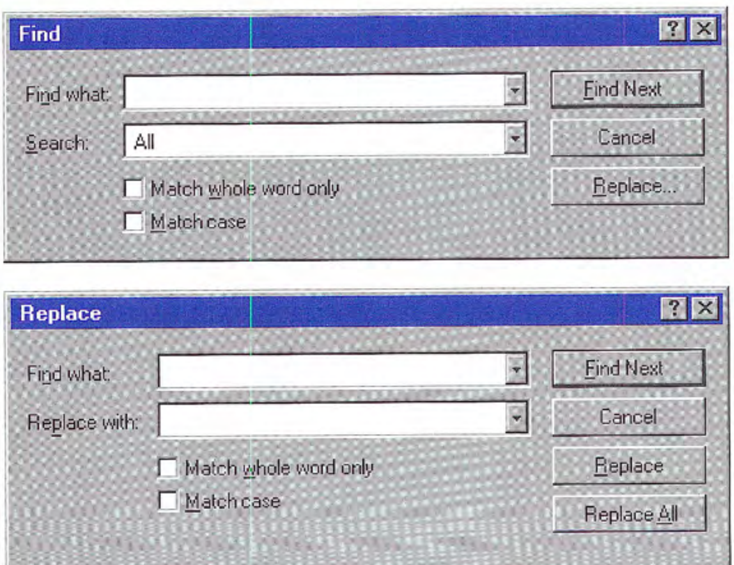


Figure 8.8 The Find and Replace dialog boxes

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.

Petitioner Exhibit 1031, Page 224

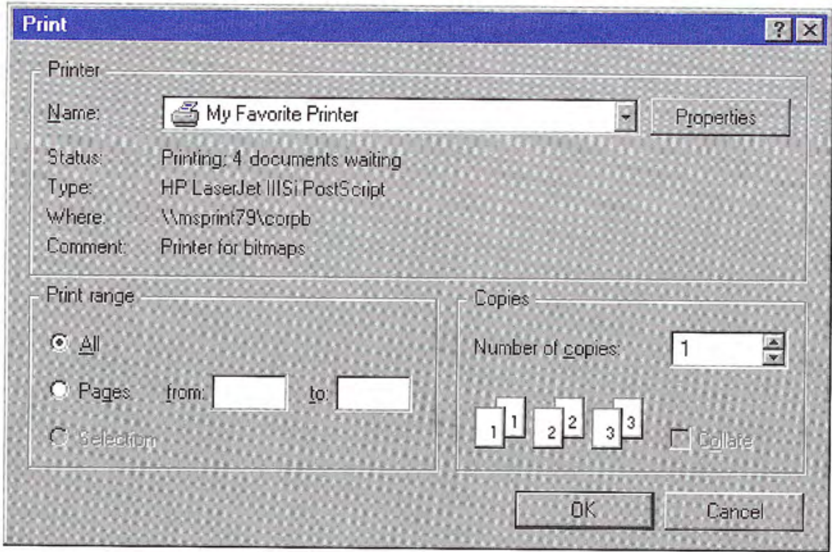


Figure 8.9 The Print dialog box

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.

Petitioner Exhibit 1031, Page 225

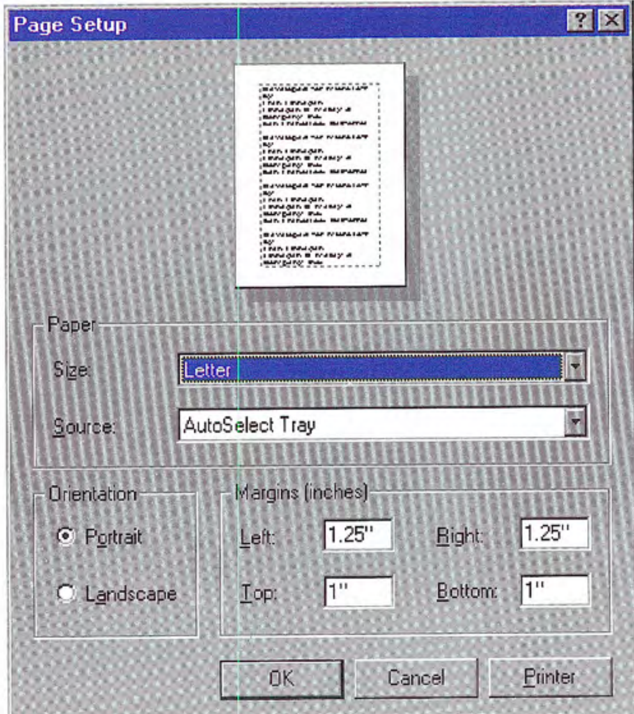


Figure 8.10 Page Setup interface used as a dialog box

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.

Petitioner Exhibit 1031, Page 226

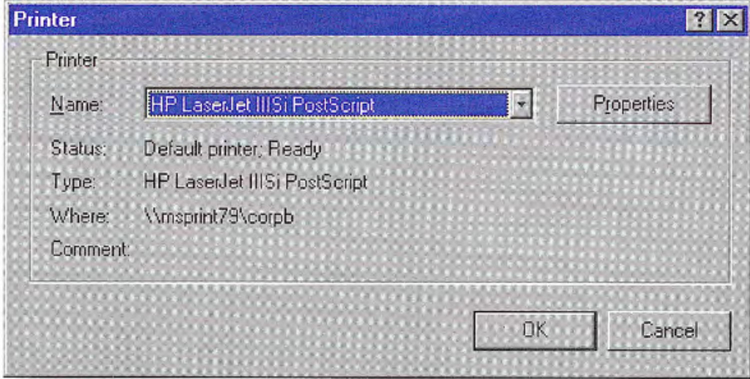


Figure 8.11 The supplemental Printer dialog box

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.

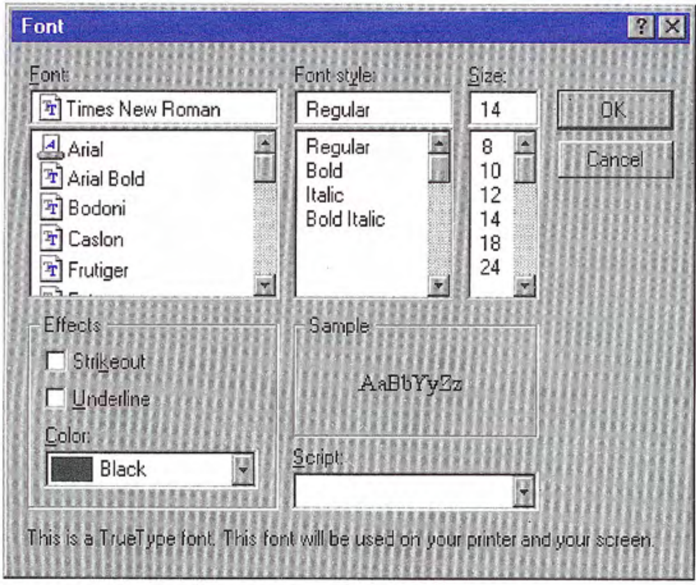


Figure 8.12 The Font dialog box

Petitioner Exhibit 1031, Page 227

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).

Petitioner Exhibit 1031, Page 228

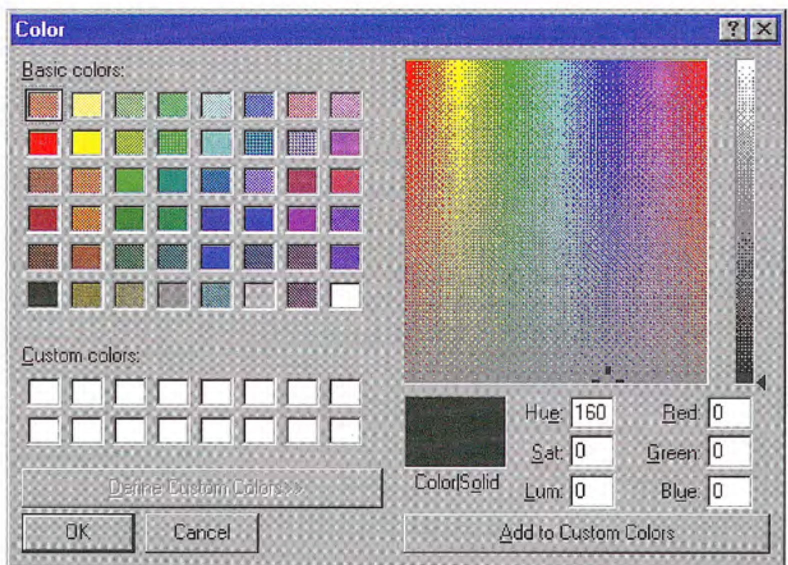


Figure 8.14 The Color dialog box (expanded)

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.

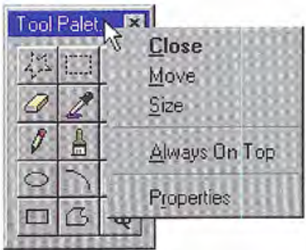


Figure 8.16 A pop-up menu for a palette window

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.

Petitioner Exhibit 1031, Page 230

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

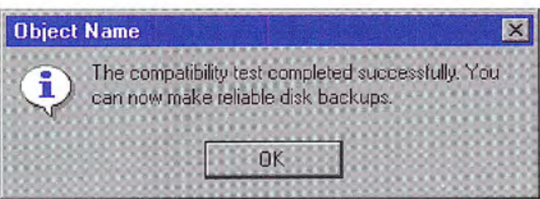
Use the title bar of a message box to appropriately identify the context in which the message is displayed — usually the name of the object. For example, if the message results from editing a document, the title text is the name of that document, optionally followed by the application name. If the message 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.

Petitioner Exhibit 1031, Page 231

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. 
	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?".
	Critical	Informs the user of a serious problem that requires intervention or correction before work can continue.

Petitioner Exhibit 1031, Page 232

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

Petitioner Exhibit 1031, Page 233

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.

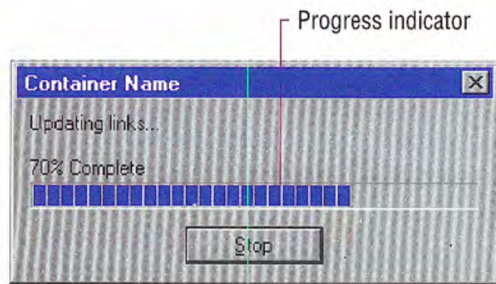


Figure 8.18 A progress message box

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."

Petitioner Exhibit 1031, Page 234

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.

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.

Petitioner Exhibit 1031, Page 235

- 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.

Petitioner Exhibit 1031, Page 236

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."

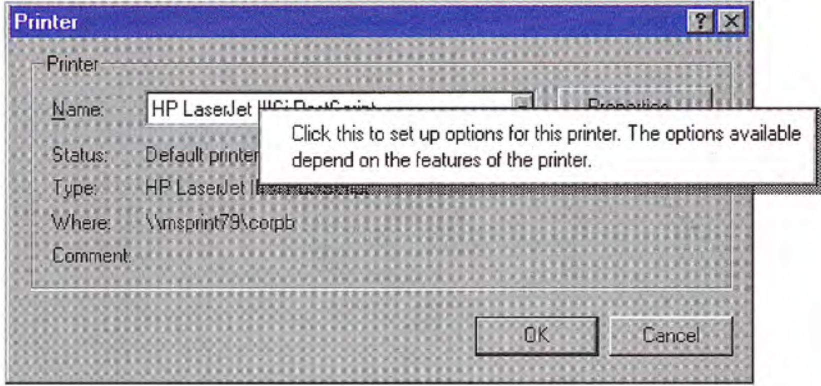


Figure 8.20 A context-sensitive Help pop-up window

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.

Petitioner Exhibit 1031, Page 237

Part III

Design Specifications and Guidelines

Petitioner Exhibit 1031, Page 239

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.

Petitioner Exhibit 1031, Page 242

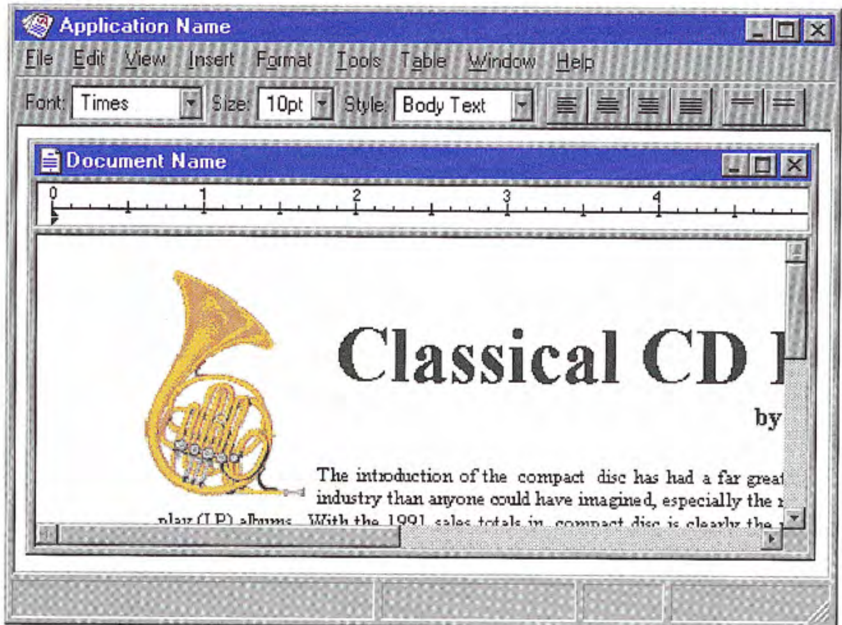


Figure 9.1 An MDI parent and child window

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

Petitioner Exhibit 1031, Page 243

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.

Petitioner Exhibit 1031, Page 244

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.

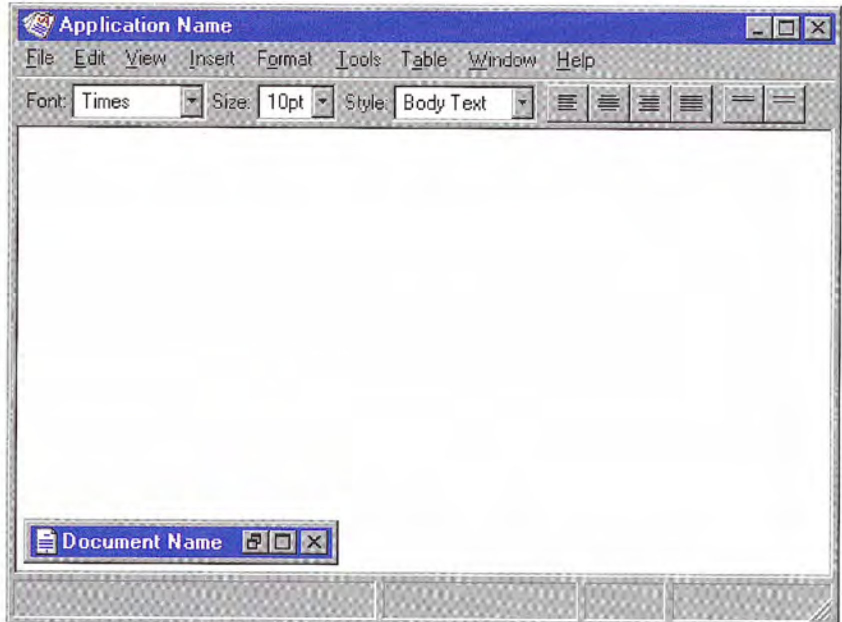


Figure 9.2 A minimized MDI child window

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

Petitioner Exhibit 1031, Page 245

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.

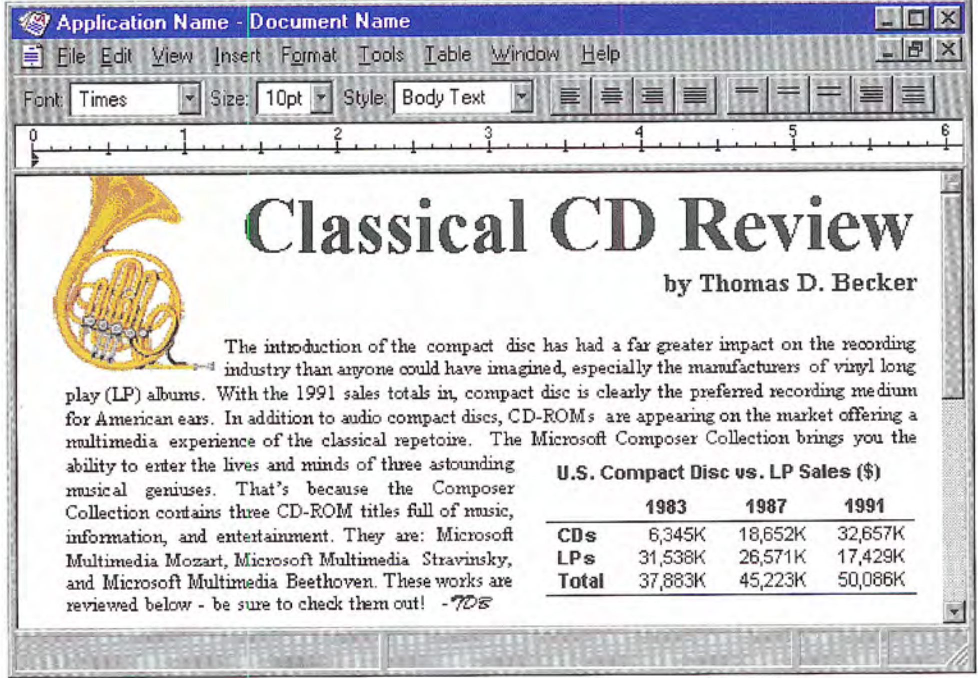


Figure 9.3 A maximized MDI child window

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.

Petitioner Exhibit 1031, Page 246

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.

Petitioner Exhibit 1031, Page 247

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.

Petitioner Exhibit 1031, Page 248

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.

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

Petitioner Exhibit 1031, Page 250

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.

Petitioner Exhibit 1031, Page 251

A Sample Workbook

File Edit View Insert Format Tools Data Window Help

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

TOC | New Client... | Capitol Hill | Fremont | Loan Est...

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.

Petitioner Exhibit 1031, Page 252

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.

Petitioner Exhibit 1031, Page 253

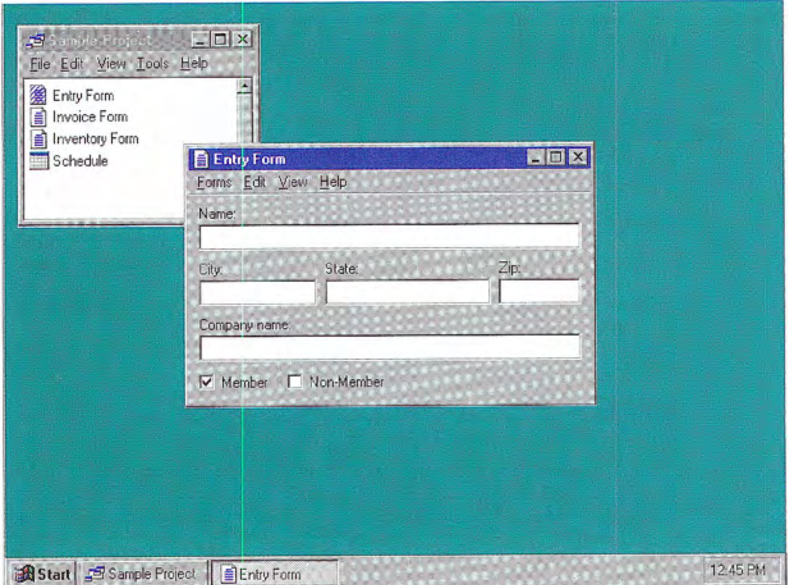


Figure 9.6 Example of a project design

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.

Petitioner Exhibit 1031, Page 254

Selecting a Window Model

Deciding how to present your application's collection of related tasks or processes requires considering a number of design factors: your intended audience and their skill level, the presentation of object or task, effective use of the space on the display, and evolution towards data-centered design.

Presentation of Object or Task

What an object represents and how it is used and relates to other objects influences how you present its view. Simple objects that are self-contained may not require a primary window, or only require a set of menu commands and a property sheet to edit their properties.

An object with user-accessible content in addition to properties, such as a document, only requires a primary window. The single document window interface can be sufficient when the object's primary presentation or use is as a single unit, even when containing different types. Alternative views can easily be supported with controls that allow the user to change the view. Simple, simultaneous views of the same data can even be supported by splitting the window into panes. The system uses the single document window style of interface for most of the components it includes, such as folders.

MDI, workspaces, workbooks, and projects are more effective when the composition of an object requires multiple views or the nature of the user's tasks requires views of multiple objects. These constructs provide a grouping and focus for a set of specific user activities, within the larger environment of the desktop.

MDI is best suited for viewing homogeneous types. The user cannot mix different objects within the same MDI parent windows unless you supply them as part of the application. On the other hand, you can use MDI to support simultaneous views of different objects.

Use a workbook when you want to optimize quick user navigation of multiple views. A workbook simplifies the task by eliminating the management of child windows, but in doing so, it limits the user's ability to see simultaneous views.

Petitioner Exhibit 1031, Page 255

Workspaces and projects provide flexibility for viewing and mixing of objects and their windows. Use a workspace as you would MDI, when you want to clearly segregate the icons and their windows used in a task. Use a project when you do not want to constrain any child windows.

A project provides the greatest flexibility for user placement and arrangement of its windows. It does so, however, at the expense of an increase in complexity because it may be more difficult for a user to differentiate the child window of a project from windows of other applications.

Display Layout

Consider the requirements for layout of information. For very high resolution displays, the use of menu bars, toolbars, and status bars poses little problem for providing adequate display of the information being viewed in a window. Similarly, the appearance of these common interface elements in each window has little impact on the overall presentation. At VGA resolution, however, this can be an issue. The interface components for a set of windows should not so dominate the user's work area that the user cannot easily view or manipulate their data.

MDI, workspaces, workbooks, and projects all allow some interface components to be shared among multiple views. Within shared elements, it must be clear when a particular interface component applies. Although you can automatically switch the content of those components, consider what functions are common across views or child windows and present them in a consistent way to provide for stability in the interface. For example, if multiple views share a Print toolbar button, present that button in a consistent location. If the button's placement constantly shifts when the user switches the view, the user's efficiency in performing the task may decrease. Note that shared interfaces may make user customization of interface components more complex because you need to indicate whether the customization applies to the current context or across all views.

Petitioner Exhibit 1031, Page 256

Regardless of the window model you chose, always consider allowing users to determine which interface components they wish to have displayed. Doing so means that you also need to consider how to make basic functionality available if the user hides a particular component. For example, pop-up menus can often supplement the interface when the user hides the menu bar.

Data-Centered Design

A single document window interface provides the best support for a simple, data-centered design and may be the easiest for users to learn; MDI supports a more conventional application-centered design. It is best suited to multiple views of the same data or contexts where the application does not represent views of user data. You can use workspaces, workbooks, and projects to provide single document window interfaces while preserving some of the management techniques provided by MDI.

Combination of Alternatives

Single document window interfaces, MDIs, workspaces, workbooks, and projects are not exclusive design techniques. It may be advantageous to combine these techniques. For example, documents can be presented within a workspace. You can also design workbooks and projects as objects within a workspace. In similar fashion, a project might contain a workbook as one of its objects.

Petitioner Exhibit 1031, Page 257

Integrating with the System

Users appreciate seamless integration between the system and their applications. This chapter covers information about integrating your software with the system and how to extend its features, including using the registry to store information about your application, installing your application, using appropriate naming conventions, and supporting shell features, such as the taskbar, Control Panel, and Recycle Bin.

This chapter is only intended to provide an overview. Details required for some conventions go beyond the scope of this guide. For information about these conventions, see the documentation included in the Microsoft Win32 Software Development Kit (SDK). In addition, some of these conventions and features may not be supported in all releases. For more information about specific releases, see Appendix D, "Supporting Specific Versions of Windows."

The Registry

Windows provides a special repository called the *registry* that serves as a central configuration database for user-, application-, and computer-specific information. Although the registry is not intended for direct user access, the information placed in it affects your application's user interface. Registered information determines the icons,

commands, and other features displayed for files. The registry also makes it easier to manage and support configuration information used by your application and eliminates redundant information stored in different locations.

The registry is a hierarchical structure. Each node in the tree is called a key. Each key can contain subkeys and data entries called values. Key names cannot include a space, backslash (\), or wildcard character (* or ?). In the **HKEY_CLASSES_ROOT** key, names beginning with a period (**.**) are reserved for special syntax (filename extensions), but you can include a period within a key name. The name of a subkey must be unique with respect to its parent key. Key names are not localized into other languages, although their values may be.

The example registry entries in this chapter represent only the hierarchical relationship of the keys. For more information about the registry and registry file formats, see the documentation included in the Win32 SDK.

A key can have any number of values. A value entry has three parts: the name of the value, its data type, and the value itself. Value entries larger than 2048 bytes should be stored as files with their filenames stored in the registry.

When the user installs your application, register keys for where application data is stored, for filename extensions, icons, shell commands, OLE registration data, and for any special extensions. To register your application's information, you can create a registration file and use the Registry Editor to merge this file into the system registry. You can also use other utilities that support this function, or use the system-supplied registry functions to access or manipulate registry data.

To use memory most efficiently, the system stores only the registry entries that have been installed and that are required for operation. Applications should never fail to write a registry entry because it is not already installed. To ensure this happens, use registry creation functions when adding an entry.

Registering Application State Information

Use the registry to store state information for your application. Typically, the data you store here will be information you may have stored in initialization (.INI) files in previous releases of Windows. Create subkeys under the **Software** subkey in the **HKEY_LOCAL_MACHINE** and **HKEY_CURRENT_USER** keys that include information about your application.

Petitioner Exhibit 1031, Page 260

HKEY_LOCAL_MACHINE
  Software
    CompanyName
      ProductName
        Version
...
HKEY_CURRENT_USER
  Software
    CompanyName
      ProductName
        Version

Use your application's **HKEY_LOCAL_MACHINE** entry as the location to store computer-specific data and the **HKEY_CURRENT_USER** entry to store user-specific data. The latter key allows you to store settings to tailor your application for individual users working with the same computer. Under your application's subkey, you can define your own structure for the information. Although the system still supports initialization files for backward compatibility, use the registry wherever possible to store your application's state information instead.

Use these keys to save your application's state whenever appropriate, such as when the user closes its primary window. In most cases, it is best to restore a window to its previous state when the user reopens it.

When the user shuts down the system with your application's window open, you may optionally store information in the registry so that the application's state is restored when the user starts up Windows. (The system does this for folders.) To have your application's state restored, store your window and application state information under its registry entries when the system notifies your application that it is shutting down. Store the state information in your application's entries under **HKEY_CURRENT_USER** and add a value name–value pair to the **RunOnce** subkey that corresponds to your application. When the user restarts the system, it runs the command line you supply. Once your application runs, you can use the data you stored to restore its state.

Petitioner Exhibit 1031, Page 261

HKEY_CURRENT_USER
Software
Microsoft
Windows
CurrentVersion
RunOnce *application identifier = command line*

If you have multiple instances open, you can include value name entries for each or consolidate them as a single entry and use command-line switches that are most appropriate for your application. For example, you can include entries like the following.

WordPad Document 1 = C:\Program Files\Wordpad.exe Letter to Bill /restore
WordPad Document 2 = C:\Program Files\Wordpad.exe Letter to Paul /restore
Paint = C:\Program Files\Paint.exe Abstract.bmp Cubist.bmp

As long as you provide a valid command-line string that your application can process, you can format the entry in a way that best fits your application.

You can also include a **RunOnce** entry under the **HKEY_LOCAL_MACHINE** key. When using this entry, however, the system runs the application before starting up. You can use this entry for applications that may need to query the user for information that affects how Windows starts. Just remember that any entry here will affect all users of the computer.

RunOnce entries are automatically removed from the registry once the system starts up. Therefore, you need not remove or update the entries, but your application must always save its state when the user shuts down the system. The system also supports a **Run** subkey in both the **HKEY_CURRENT_USER** and **HKEY_LOCAL_MACHINE** keys. The system runs any value name entries under this subkey after the system starts up, but does not remove those entries from the registry. For example, a virus check program can be installed to run automatically after the system starts up. You can also support this functionality by placing a file or shortcut to a file in the Startup folder. The registry stores the location of the Startup folder, as a value in **HKEY_CURRENT_USER \Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders**.

The system's ability to restore an application's state depends on the availability of the application and its data files. If they have been deleted or the user has logged in over the network where the same files are not available, the system may not be able to restore the state.

Petitioner Exhibit 1031, Page 262

Registering Application Path Information

The system supports "per application" paths. If you register a path, Windows sets the PATH environment variable to be the registered path when it starts your application. You set your application's path in the **App Paths** subkey under the **HKEY_LOCAL_MACHINE** key. Create a new key using your application's executable filename as its name. Set this key's Default value to the path of your executable file. The system uses this entry to locate your application if it fails to find it in the current path; for example, if the user chooses the Run command on the Start menu and only includes the filename of the application, or if a shortcut icon doesn't include a path setting. To identify the location of dynamic-link libraries placed in a separate directory, you can also include another value entry called Path and set its value to the path of your dynamic-link libraries.

```
HKEY_LOCAL_MACHINE
    Software
        Microsoft
            Windows
                CurrentVersion
                    App Paths
                        Application Executable Filename = path
                                                          Path = path
```

The system will automatically update the path and default entries if the user moves or renames the application's executable file using the system shell user interface.

Register any system-wide shared dynamic-link libraries in a subkey under a **SharedDLLs** subkey of **HKEY_LOCAL_MACHINE** key. If the file already exists, increment the entry's usage count index. For more information about the usage count index, see the section, "Installation," later in this chapter.

```
HKEY_LOCAL_MACHINE
    Software
        Microsoft
            Windows
                CurrentVersion
                    SharedDLLs filename [= usage count index]
```

Registering File Extensions

If your application creates and maintains files, register entries for the file types that you expose directly to users and that you want users to be able to easily differentiate. For every file type you register, include at least two entries: a filename-extension key entry and an application (class) identification key entry.

If you do not register an extension for a file type, it will be displayed with the system's generic file object icon, as shown in Figure 10.1, and its extension will always be displayed. In addition, the user will not be able to double-click the file to open it. (Open With will be the icon's default command.)

Figure 10.1 System-generated icons for unregistered types

The Filename Extension Key

The filename extension entry maps a filename extension to an application identifier. To register an extension, create a subkey in the **HKEY_CLASSES_ROOT** key using the three-letter extension (including a period) and set its value to an application identifier.

HKEY_CLASSES_ROOT
    .ext = *ApplicationIdentifier*

For the value of the application identifier (also known as programmatic identifier or Prog ID), use a string that uniquely identifies a given class. This string is used internally by the system and is not exposed directly to users (unless explicitly exported with a special registry utility); therefore, you need not localize this entry.

Petitioner Exhibit 1031, Page 264

Avoid assigning multiple extensions to the same application identifier. To ensure that each file type can be distinguished by the user, define each extension such that each has a unique application identifier. If you have utility files that the user does not interact with directly, you should still register an extension (and icon) for them, preferably the same extension so that they can be identified. In addition, mark them with the hidden file attribute.

The system provides no arbitration for applications that use the same extensions. So define unique identifiers and check the registry to avoid writing over and replacing existing extension entries, a practice which may seriously affect the user's existing files. More specifically, avoid registering an extension that conflicts or redefines the common filename extensions used by the system. Examples of these extensions are shown in Table 10.1.

Table 10.1 Common Filename Extensions Supported by Windows

Extension	Type description
386	Windows virtual device driver
3GR	Screen grabber for MS-DOS–based applications
ACM	Audio compression manager driver
ADF	Administration configuration files
ANI	Animated pointer
AVI	Video clip
AWD	FAX viewer document
AWP	FAX key viewer
AWS	FAX signature viewer
BAK	Backed-up file
BAT	MS-DOS batch file
BFC	Briefcase
BIN	Binary data file
BMP	Picture (Windows bitmap)
CAB	Windows Setup file
CAL	Windows Calendar file
CDA	CD audio track
CFG	Configuration file

Petitioner Exhibit 1031, Page 265

(Continued)

Extension	Type description
CNT	Help contents
COM	MS-DOS – based application
CPD	FAX cover page
CPE	FAX cover page
CPI	International code page
CPL	Control Panel extension
CRD	Windows Cardfile document
CSV	Command-separated data file
CUR	Cursor (pointer)
DAT	System data file
DCX	FAX viewer document
DLL	Application extension (dynamic-link library)
DOC	WordPad document
DOS	MS-DOS file (also extension for NDIS2 net card and protocol drivers)
DRV	Device driver
EXE	Application
FND	Saved search
FON	Font file
FOT	Shortcut to font
GR3	Windows 3.0 screen grabber
GRP	Program group file
HLP	Help file
HT	HyperTerminal™ file
ICM	ICM profile
ICO	Icon
IDF	MIDI instrument definition
INF	Setup information
INI	Initialization file (configuration settings)

Petitioner Exhibit 1031, Page 266

(Continued)

Extension	Type description
KBD	Keyboard layout
LGO	Windows logo driver
LIB	Static-link library
LNK	Shortcut
LOG	Log file
MCI	MCI command set
MDB	File viewer extension
MID	MIDI sequence
MIF	MIDI instrument file
MMF	Microsoft Mail message file
MMM	Animation
MPD	Mini-port driver
MSG	Microsoft® Exchange mail document
MSN	Microsoft Network home base
NLS	Natural language services driver
PAB	Microsoft Exchange personal address book
PCX	Bitmap picture (PCX format)
PDR	Port driver
PF	ICM profile
PIF	Shortcut to MS-DOS–based application
PPD	PostScript® printer description file
PRT	Printer formatted file (result of Print to File option)
PST	Microsoft Exchange personal information store
PWL	Password list
QIC	Backup set for Microsoft Backup
REC	Windows Recorder file
REG	Application registration file
RLE	Picture (RLE format)
RMI	MIDI sequence

Petitioner Exhibit 1031, Page 267

(Continued)

Extension	Type description
RTF	Document (rich-text format)
SCR	Screen saver
SET	File set for Microsoft Backup
SHB	Shortcut into a document
SHS	Scrap
SPD	PostScript printer description file
SWP	Virtual memory storage
SYS	System file
TIF	Picture (TIFF® format)
TMP	Temporary file
TRN	Translation file
TSP	Windows telephony service provider
TTF	TrueType® font
TXT	Text document
VBX	Microsoft Visual Basic® control file
VER	Version description file
VXD	Virtual device driver
WAV	Sound wave
WPC	WordPad file converter
WRI	Windows Write document

It is a good idea to investigate extensions commonly used by popular applications so you can avoid creating a new extension that might conflict with them, unless you intend to replace or superset the functionality of those applications.

Petitioner Exhibit 1031, Page 268

The Application Identifier Key

The second registry entry you create for a file type is its class-definition (Prog ID) key. Using the same string as the application identifier you used for the extension's value, create a key, and assign a type name as the value of the key.

HKEY_CLASSES_ROOT
    .ext = *ApplicationIdentifier*
    ApplicationIdentifier = *Type Name*

Under this key, you specify shell and OLE properties of the class. Provide this entry even if you do not have any extra information to place under this key; doing so provides a label for users to identify the file type. In addition, you use this entry to register the icon for the file type.

Define the type name (also known as the MainUserTypeName) as the human-readable form of its application identifier or class name. It should convey to the user the object's name, behavior, or capability. A type name can include all of the following elements:

1. *Company Name*
   Communicates product identity.

2. *Application Name*
   Indicates which application is responsible for activating a data object.

3. *Data Type*
   Indicates the basic category of the object (for example, drawing, spreadsheet, or sound). Limit the number of characters to a maximum of 15.

4. *Version*
   When there are multiple versions of the same basic type, for upgrading purposes, you may want to include a version number to distinguish types.

When defining your type name, use title capitalization. The name can include up to a maximum of 40 characters. Use one of the following three recommended forms:

Petitioner Exhibit 1031, Page 269

1. *Company Name Application Name* [*Version*] *Data Type*
   For example, Microsoft Excel Worksheet.

2. *Company Name-Application Name* [*Version*] *Data Type*
   For cases when the company name and application are the same — for example, ExampleWare 2.0 Document.

3. *Company Name Application Name* [*Version*]
   When the application sufficiently describes the data type — for example, Microsoft Graph.

These type names provide the user with a precise language for referring to objects. Because object type names appear throughout the interface, the user becomes conscious of an object's type and its associated behavior. However, because of their length, you may also want to include a short type name. A *short type name* is the data type portion of the full type name. Applications that support OLE always include a short type name entry in the registry. Use the short type name in drop-down and pop-up menus. For example, a Microsoft® Excel Worksheet is simply referred to as a "Worksheet" in menus.

To provide a short type name, add an **AuxUserType** subkey under the application's registered **CLSID** subkey (which is under the **CLSID** key).

For more information about registering type names and other information you should include under the **CLSID** key, see the OLE documentation included in the Win32 SDK.

```
HKEY_CLASSES_ROOT
    .ext = ApplicationIdentifier
...
    ApplicationIdentifier = Type Name
        CLSID = {CLSID identifier}
...
    CLSID
        {CLSID identifier}
            AuxUserType
                2 = Short Type Name
```

If a short type name is not available for an object because the string was not registered, use the full type name instead. All controls that display the full type name must allocate enough space for 40 characters in width. By comparison, controls need only accommodate 15 characters when using the short type name.

Petitioner Exhibit 1031, Page 270

Supporting Creation

The system supports the creation of new objects in system containers, such as folders and the desktop. Register information for each file type that you want the system to include. The registered type will appear on the New command that the system includes on menus for the desktop, folders, and the Open and Save As common dialog boxes. This provides a more data-centered design because the user can create a new object without having to locate and run the associated application.

To register a file type for inclusion, create a subkey using the Application Identifier under the extension's subkey in **HKEY_CLASSES_ROOT**. Under it, also create the **ShellNew** subkey.

HKEY_CLASSES_ROOT
    .ext = *ApplicationIdentifier*
        ApplicationIdentifier
            ShellNew *Value Name* = *Value*

Assign a value entry to the **ShellNew** subkey with one of the four methods for creating a file with this extension.

Value name	Value	Result
NullFile	" "	Creates a new file of this type as a null (empty) file.
Data	*binary data*	Creates a new file containing the binary data.
FileName	*path*	Creates a new file by copying the specified file.
Command	*filename*	Carries out the command. Use this to run your own application code to create a new file (for example, run a wizard).

Petitioner Exhibit 1031, Page 271

The system also will automatically provide a unique filename for the new file using the type name you register.

When using a Command value, place your application file (that creates the new file) in the directory that the system uses to store these files. To determine the path for that directory, check the setting for the Templates value in the **Shell Folders** subkey found in **HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer**. Then you need only register the filename for the command.

Registering Icons

The system uses the registry to determine which icon to display for a specific file. You register an icon for every data file type that your application supports and that you want the user to be able to distinguish easily. Create a **DefaultIcon** subkey entry under the application identifier subkey you created and define its value as the filename containing the icon. Typically, you use the application's executable filename and the index of the icon within the file. The index value corresponds to the icon resource within the file. A positive number represents the icon's position in the file. A negative number corresponds to the inverse of the resource ID number of the icon. The icon for your application should always be the first icon resource in your executable file. The system always uses the first icon resource to represent executable files. This means the index value for your data files will be a number greater than 0.

HKEY_CLASSES_ROOT
    ApplicationIdentifier = *Type Name*
        DefaultIcon = *path* [,*index*]

Instead of registering the application's executable file, you can register the name of a dynamic link library file (.DLL), an icon file (.ICO), or bitmap file (.BMP) to supply your data file icons. If an icon does not exist or is not registered, the system supplies an icon derived from the icon of the file type's registered application. If no icon is available for the application, the system supplies a generic icon. These icons do not make your files uniquely identifiable, so design and register icons for both your application and its data file types. Include the following sizes: 16 x 16 pixel (16 color), 32 x 32 pixel (16 color), and 48 x 48 pixel (256 color).

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

Petitioner Exhibit 1031, Page 272

Registering Commands

Many of the commands found on icons, including Send To, Cut, Copy, Paste, Create Shortcut, Delete, Rename, and Properties, are provided by their container — that is, their containing folder or the desktop. But you must provide support for the icon's primary commands, also referred to as verbs, such as Open, Edit, Play, and Print. You can also register additional commands that apply to your file types, such as a What's This? command and even commands for other file types.

To add these commands, in the **HKEY_CLASSES_ROOT** key, you register a **shell** subkey and a subkey for each verb, and a **command** subkey for each menu command name.

HKEY_CLASSES_ROOT
  ApplicationIdentifier = *Type Name*
    shell [= *default verb* [,*verb2* [,..]]
      verb [= *Menu Command Name*]
        command = *pathname* [*parameters*]

You can also register a DDE command string for a DDE command.

HKEY_CLASSES_ROOT
  ApplicationIdentifier = *Type Name*
    shell [= *default verb* [,*verb2* [,..]]
      verb [= *Menu Command Name*]
        ddeexec = *DDE command string*
          Application = *DDE Application Name*
          Topic = *DDE topic name*

A verb is a language-independent name of the command. Applications may use it to invoke a specific command programmatically. The system defines Open, Print, Find, and Explore as standard verbs and automatically provides menu command names and appropriate access key assignments, localized in each international version of Windows. When you supply verbs other than these, provide menu command names localized for the specific version of Windows on which the application is installed. To assign a menu command name for a verb, make it the default value of the verb subkey.

Petitioner Exhibit 1031, Page 273

The menu command names corresponding to the verbs for a file type are displayed to the user, either on a folder's File drop-down menu or pop-up menu for a file's icon. These appear at the top of the menu. You define the order of the menu commands by ordering the verbs in the value of the **shell** key. The first verb becomes the default command in the menu.

By default, capitalization follows how you enter format the menu command name value of the verb subkey. Although the system automatically capitalizes the standard commands (Open, Print, Explore, and Find), you can use the value of the menu command name to format the capitalization differently. Similarly, you use the menu command name value to set the access key for the menu command following normal menu conventions, prefixing the character in the name with an ampersand (&). Otherwise, the system sets the first letter of the command as the access key for that command.

To support user execution of a verb, provide the path for the application or a DDE command string. You can include command-line switches. For paths, include a %1 parameter. This parameter is an operational placeholder for whatever file the user selects.

For example, to register an Analyze command for an application that manages stock market information, the registry entries might look like the following.

```
HKEY_CLASSES_ROOT
    stockfile = Stock Data
        shell = analyze
            analyze = &Analyze
                command = C:\Program Files\Stock Analysis\Stock.exe /A
```

You may have different values for each command. You may assign one application to carry out the Open command and another to carry out the Print command, or use the same application for all commands.

Petitioner Exhibit 1031, Page 274

Enabling Printing

If your file types are printable, include a Print verb entry in the **shell** subkey under **HKEY_CLASSES_ROOT**, following the conventions described in the previous section. This will display the Print command on the pop-up menu for the icon and on the File menu of the folder in which the icon resides when the user selects the icon. When the user chooses the Print command, the system uses the registry entry to determine what application to run to print the file.

Also register a Print To registry entry for the file types your application supports. This entry enables dragging and dropping of a file onto a printer icon. Although a Print To command is not displayed on any menu, the printer includes Print Here as the default command on the pop-up menu displayed when the user drag and drops a file on the printer using button 2.

In both cases, print the file, preferably, without opening the application's primary window. One way to do this is to provide a command-line switch that runs the application for handling the printing operation only (for example, WordPad.exe /p). In addition, display some form of user feedback that indicates whether a printing process has been initiated and, if so, its progress. For example, this feedback could be a modeless message box that displays, "Printing page *m* of *n* on *printer name*" and a Cancel button. You may also include a progress indicator control.

Registering OLE

Applications that support OLE use the registry as the primary means of defining class types, operations, and properties for data types supported by those applications. You store OLE registration information in the **HKEY_CLASSES_ROOT** key in subkeys under the **CLSID** subkey and in the class description's (Prog ID) subkey.

For more information about the specific registration entries for OLE, see the OLE documentation included in the Win32 SDK.

Petitioner Exhibit 1031, Page 275

Registering Shell Extensions

Your application can extend the functionality of the operational environment provided by the system, also known as the shell, in a number of ways. A shell extension enhances the system by providing additional ways to manipulate file objects, by simplifying the task of browsing through the file system, or by giving the user easier access to tools that manipulate objects in the file system.

Support for shell extensions may depend on the version of Windows installed. For more information about specific releases, see Appendix D, "Supporting Specific Versions of Windows."

Every shell extension requires a *handler*, special application code (32-bit OLE InProc server implemented as a dynamic-link library) that implements subordinate functions. The types of handlers you can provide include:

- Pop-up (context) menu handlers: these add menu items to the pop-up menu for a particular file type.
- Drag handlers: these allow you to support the OLE data transfer conventions for drag and drop operations of a specific file type.
- Drop handlers: these allow you to carry out some action when the user drops objects on a specific type of file.
- Nondefault drag and drop handlers: these are pop-up menu handlers that the system calls when the user drags and drops an object by using mouse button 2.
- Icon handlers: these can be used to add per-instance icons for file objects or to supply icons for all files of a specific type.
- Property sheet handlers: these add pages to a property sheet that the shell displays for a file object. The pages can be specific to a class of files or to a particular file object.
- Copy-hook handlers: these are called when a folder or printer object is about to be moved, copied, deleted, or renamed by the user. The handler can be used to allow or prevent the operation.

Petitioner Exhibit 1031, Page 276

You register the handler for a shell extension in the **HKEY_CLASSES_ROOT** key. The **CLSID** subkey contains a list of class identifier key values such as {00030000-0000-0000-C000-000000000046}. Each class identifier must also be a globally unique identifier.

For more information about creating handlers and class identifiers, see the OLE documentation included in the Win32 SDK.

You must also create a **shellex** subkey under the application's class identification entry in the **HKEY_CLASSES_ROOT** key.

```
HKEY_CLASSES_ROOT
    ApplicationIdentifier = Type Name
        Shell [ = default verb [,verb2 [,..]]
            ...
            shellex
                HandlerType
                    {CLSID identifier} = Handler Name
                ...
                HandlerType = {CLSID identifier}
```

The shell also uses several other special keys, such as *, **Folder**, **Drives**, and **Printers**, under **HKEY_CLASSES_ROOT**. You can use these keys to register extensions for system-supplied objects. For example, you may use the * key to register handlers that the shell calls whenever it creates a pop-up menu or property sheet for a file object, as in the following example.

```
HKEY_CLASSES_ROOT
    * = *
        shellex
            ContextMenuHandlers
                {00000000-1111-2222-3333-0000000001}
            PropertySheetHandlers = SummaryInfo
                {00000000-1111-2222-3333-0000000002}
            IconHandler = {00000000-1111-2222-3333-000000003}
```

The shell would use these handlers to add to the pop-up menus and property sheets of every file object. (The entries are intended only as examples, not literal entries.)

Petitioner Exhibit 1031, Page 277

A pop-up menu handler may add commands to the pop-up menu of a file type, but it may not delete or modify existing menu commands. You can register multiple pop-up menu handlers for a file type. The order of the subkey entries determines the order of the items in the context menu. Handler-supplied menu items always follow registered command names.

Keep in mind that if you want to include a command on the pop-up menu of every file of a particular type, you do not need to create and register a pop-up menu handler. You can just use the normal means of registering commands for that type. Create a pop-up menu handler only when you want to provide a command only under specific conditions, such as the length of the file or its timestamp.

When registering an icon handler for providing per-instance icons for a file type, set the value for the **DefaultIcon** key to %1. This denotes that each file instance of this type can have a different icon.

Supporting the Quick View Command

The system includes support for fast, read-only views of many file types when the user chooses the Quick View command from the file object's menu. This allows the user to view files without opening the application.

For more information about supporting the Quick View command and creating file viewers, see the documentation included in the Win32 SDK.

If your file type is not supported, you can install a file parser that translates your file type into a format the system file viewer can read. Although this approach allows you to easily support viewers for your data file types, it limits the interaction options for your file types to those provided by the system. Alternatively, you can create your own file viewer, using the system-supplied interfaces. You can also register a file viewer for a file type already registered.

You can also support the Quick View command for data objects stored within your application's interface, either by supplying a specific viewer for your data types or by writing the data to a temporary file and then executing a file viewer and passing the temporary file as a parameter.

Petitioner Exhibit 1031, Page 278

Registering Sound Events

Your application can register specific events to which the user can assign sound files so that when those events are triggered, the assigned sound file is played. To register a sound event, create a key under the **HKEY_CURRENT_USER** key.

HKEY_CURRENT_USER
  AppEvents
    Event Labels
      EventName = *Event Name*

Set the value for EventName to a human-readable name.

Registering a sound event only makes it available in Control Panel so the user can assign a sound file. Your application must provide the code to process that event.

Installation

The following sections provide guidelines for installing your application's files. Applying these guidelines will help you reduce the clutter of irrelevant files when the user browses for a file. In addition, you'll reduce the redundancy of common files and make it easier for the user to update applications or the system software.

Copying Files

When the user installs your software, avoid copying files into the Windows directory (folder) or its System subdirectory. Doing so clutters the directory and may degrade system performance. Instead, create a single directory, preferably using the application's name, in the Program Files directory (or the location that the user chooses). In this directory, place the executable file. For example, if a program is named My Application, create a My Application subdirectory and place My Application.exe in that directory.

Petitioner Exhibit 1031, Page 279

To locate the Program Files directory, check the ProgramFilesDir value in the **CurrentVersion** subkey under **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows**. The actual directory may not literally be named Program Files. For example, in international versions of Microsoft Windows, the directory name is appropriately localized. For networks that do not support the Windows long filename conventions, MS-DOS names may be used instead.

In your application's directory, create a subdirectory named System and place all support files that the user does not directly access in it, such as dynamic-link libraries and Help files. For example, place a support file called My Application.dll in the subdirectory Program Files\My Application\System. Hide the support files and your application's System directory and register its location using a Path value in the **App Paths** subkey under **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion**. Although you may place support files in the same directory as your application, placing them in a subdirectory helps avoid confusing the user and makes files easier to manage.

Applications can share common support files to reduce the amount of disk space consumed by duplication. If some non-user-accessed files of your application are shared as systemwide components (such as Visual Basic's Vbrun300.dll), place them in the System subdirectory of the directory where the user installs Windows. The process for installing shared files includes these logical steps:

1. Before copying the file, verify that it is not already present.
2. If the file is already present, compare its date and size to determine whether it is the same version as the one you are installing. If it is, increment the usage count in its corresponding registry entry.
3. If the file you are installing is not more recent, do not overwrite the existing version.
4. If the file is not present, copy it to the directory.

The system provides support services in Ver.dll for assisting you to do version verification. For more information about this utility, see the documentation included in the Win32 SDK.

If you store a new file in the System directory installed by Windows, register a corresponding entry in the **SharedDLL** subkey under the **HKEY_LOCAL_MACHINE** key.

Petitioner Exhibit 1031, Page 280

If a file is shared, but only among your applications, create a subdirectory using your application's name in the Common Files subdirectory of the Program Files subdirectory and place the file there. To locate the Common Files directory, check the Common-FilesDir value in the **CurrentVersion** subkey of **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows**. Alternatively, for "suite" style when multiple applications are bundled together, you can create a suite subdirectory in Program Files, where you place your executable files, and within that a System subdirectory with the support files shared only within the suite. In either case, register the path using the **Path** subkey under the **App Paths** subkey

When installing an updated version of the shared file, ensure that it is upwardly compatible before replacing the existing file. Alternatively, you can create a separate entry with a different filename (for example, Vbrun301.dll).

Name your executable file, dynamic-link libraries, and any other files that the user does not directly use, but that may be shared on a network, using conventional MS-DOS (8.3) names rather than long filenames. This will provide better support for users operating in environments where these files may need to be installed on network services that do not support the Windows long filename conventions.

Windows no longer requires Autoexec.bat and Config.sys files. Ensure that your application also does not require these files. Consider converting any MS-DOS device drivers to Windows virtual device drivers. The system supports dynamic loading of this type of device drivers, unlike MS-DOS device drivers which need to be loaded through Config.sys when starting the system. Similarly, because the registry allows you to register your application paths, your application does not require path information in Autoexec.bat.

In addition, do not make entries in Win.ini. Storing information in this file can make it difficult for the user to update or move your application. Also, avoid maintaining your application's own initialization file. Instead, use the registry. The registry provides conventions for storing most application and user settings. The registry provides greater flexibility allowing you to store information on a per machine or per user basis. It also supports accessing this infomation across a network.

Petitioner Exhibit 1031, Page 281

Make certain you register the types supported by your application and the icons for these types along with your application's icons. In addition, register other application information, such as information required to enable printing.

Providing Access to Your Application

To provide easy user access to your application, place a shortcut icon to the application in the Programs folder. You can determine the path for this folder in **Shell Folders** subkey under **HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer**. This adds the entry to the submenu of the Programs menu of the Start button. Avoid adding entries for every application you might include in your software; this quickly overloads the menu. Optionally, you can allow the user to choose which icons to place in the menu. Avoid using a folder as your entry in the Programs menu, because this creates a multilevel hierarchy. Including a single entry makes it easier and simpler for a user to access your application.

You can create a "program group" entry in the Programs folder using the Windows 3.1 dynamic date exchange application programming interface (API). However, it is not recommended for applications installed with Microsoft Windows 95 and later releases, configured with the new shell user interface.

Also consider the layout of files you provide with your application. Folders in Windows 95 and later releases provide much greater flexibility for file organization than did the Windows Program Manager. In addition to the recommended structure for your main executable file and its support files, you may want to create special folders for documents, templates, conversion tools, or other files that the user accesses directly.

Designing Your Installation Program

Your installation program should offer the user different installation options such as:

- Typical Setup: installation that proceeds with the common defaults set, copying only the most common files. Make this the default setup option.

- Compact Setup: installation of the minimum files necessary to operate your application. This option is best for situations where disk space must be conserved — for example, on laptop computers. You can optionally add a Portable setup option for additional functionality designed especially for configurations on laptops, portables, and portables used with docking stations.

Petitioner Exhibit 1031, Page 282

- Custom Setup: installation for the experienced user. This option allows the user to choose where to copy files and which options or features to include. This can include options or components not available for compact or typical setup.

- CD-ROM Setup: installation from a CD-ROM. This option allows users to select what files to install from the CD and allows them to run the remaining files directly from the CD.

- Silent Setup: installation using a command-line switch. This allows your setup program to run with a batch file

In addition to these setup options, your installation program should be a well-designed, Windows-based application and follow the conventions detailed in this guide and in the following guidelines:

- Supply a common response to every option so that the user can step through the installation process by confirming the default settings (that is, by pressing the ENTER key).

- Tell users how much disk space they will need before proceeding with installation. In the custom setup option, adjust the figure as the user chooses to include or exclude certain options. If there is not sufficient disk space, let the user know, but also give the user the option to override.

- Offer the user the option to quit the installation before it is finished. Keep a log of the files copied and the settings made so the canceled installation can be cleaned up easily.

- Ask the user to insert a disk only once during the installation. Lay out your files on disk so that the user does not have to reinsert the same disk multiple times.

- Provide a visual prompt and an audio cue when the user needs to insert the next disk.

- Support installation from any location. Do not assume that installation must be done from a logical MS-DOS drive (such as drive A). Design your installation program to support any valid universal naming convention (UNC) path.

- Provide a progress indicator message box to inform the user how far they are through the installation process.

Petitioner Exhibit 1031, Page 283

If you are creating your own installation program, consider using the wizard control. Using this control and following the guidelines for wizards will result in a consistent interface for users.

For more information about designing wizards, see Chapter 12, "User Assistance."

Naming your installation program Setup.exe or Install.exe (or localized equivalent) will allow the system to recognize the file. Place the file in the root directory of the disk the user inserts. This allows the system to automatically run your installation program when the user chooses the Install button in the Add/Remove Programs utility in Control Panel.

Installing Fonts

When installing fonts with your application on a local system, determine whether the font is already present. If it is, rename your font file — for example, by appending a number to the end of its filename. After copying a font file, register the font in the **Fonts** subkey.

Installing Your Application on a Network

If you create a client-server application so that multiple users access it from a network server, create separate installation programs: an installation program that allows the network administrator to prepare the server component of the application, and a client installation program that installs the client component files and sets up the settings to connect to the server. Design your client software so that an administrator can deploy it over the network and have it automatically configure itself when the user starts it.

For more information about designing client-server applications, see Chapter 14, "Special Design Considerations." Additional information can also be found in the documentation included in Win32 SDK.

Because Windows may itself be configured to be shared on a server, do not assume that your installation program can store information in the main Windows directory on the server. In addition, shared application files should not be stored in the "home" directory provided for the user.

Design your installation program to support UNC paths. Also, use UNC paths for any shortcut icons you install in the Start Menu folder.

Petitioner Exhibit 1031, Page 284

Uninstalling Your Application

The user may need to remove your application to recover disk space or to move the application to another location. To facilitate this, provide an uninstall program with your application that removes its files and settings. Remember to remove registry entries and shortcuts your application may have placed in the Start menu hierarchy. However, be careful when removing your application's directory structure not to delete any user files (unless you confirm their removal with the user).

Your uninstall program should follow the conventions detailed in this guide and in the following guidelines:

- Display a window that provides the user with information about the progress of the uninstall process. You can also provide an option to allow the program to uninstall "silently" — that is, without displaying any information so that it can be used in batch files.
- Display clear and helpful messages for any errors your uninstall program encounters during the uninstall process.
- When uninstalling an application, decrement the usage count in the registry for any shared component — for example, a dynamic-link library. If the result is zero, give the user the option to delete the shared component with the warning that other applications may use this file and will not work if it is missing.

Registering your uninstall program will display your application in the list of the Uninstall page of the Add/Remove Program utility included with Windows. To register your uninstall program, add entries for your application to the **Uninstall** subkey.

```
HKEY_LOCAL_MACHINE
    Software
        Microsoft
            Windows
                CurrentVersion
                    Uninstall
                        ApplicationName DisplayName = Application Name
                                        UninstallString = path [ switches ]
```

Petitioner Exhibit 1031, Page 285

Both the **DisplayName** and **UninstallString** values must be supplied and be complete for your uninstall program to appear in the Add/Remove Program utility. The path you supply to **Uninstall-String** must be the complete command line used to carry out your uninstall program. The command line you supply should carry out the uninstall program directly rather than from a batch file or subprocess.

Supporting AutoPlay

Windows supports the ability to automatically run a file when the user inserts removable media that support insertion notification, such as CD-ROM, PCMCIA hard disks, or flash ROM cards. To support this feature, include a file named Autorun.inf in the root directory of the removable media. In this file, include the filename of the file to run, using the following syntax.

```
[autorun]
open = filename
```

Unless you specify a path, the system looks for the file in the root of the inserted media. If you want to run a file located in a subdir-ectory, include a path relative to the root; include that path with the file as in the following example.

```
open = My Directory\My File.exe
```

Running the file from a subdirectory does not change the current directory setting. The command-line string you supply can also in-clude parameters or switches.

Because the autoplay feature is intended to provide automatic operation, design the file you specify in the Autorun.inf file to pro-vide visual feedback quickly to confirm the successful insertion of the media. Consider using a startup up window with a graphic or animated sequence. If the process you are automating requires a long load time or requires user input, offer the user the option to cancel the process.

Petitioner Exhibit 1031, Page 286

Although you can use this feature to install an application, avoid writing files to the user's local disk without the user's confirmation. Even when you get the user's confirmation, minimize the file storage requirements, particularly for CD-ROM games or educational applications. Consuming a large amount of local file space defeats some of the benefits of the turnkey operation that the autoplay feature provides. Also, because a network administrator or the user can disable this feature, avoid depending on it for any required operations.

You can define the icon that the system displays for the media by including an entry in the Autorun.inf file that includes the filename (and optionally the path) including the icon using the following form.

icon = *filename*

The filename can specify an icon, a bitmap, an executable, or a dynamic-link library file. If the file contains more than one icon resource, specify the resource with a number after the filename — for example, My File.exe, 1. The numbering follows the same conventions as the registry. The default path for the file will be relative to the Autorun.inf file. If you want to specify an absolute path for an icon, use the following form.

defaulticon = *path*

The system automatically provides a pop-up menu for the icon and includes AutoPlay as the default command on that menu, so that double-clicking the icon will run the Open = line. You can include additional commands on the menu for the icon by adding entries for them in the Autorun.inf file, using the following form.

shell\verb\command = *filename*
shell\verb = *Menu Item Name*

To define an access key assignment for the command, precede the character with an ampersand (&). For example, to add the command Read Me First to the menu of the icon, include the following in the Autorun.inf file.

shell\readme\command = Notepad.exe My Directory\Readme.txt
shell\readme = Read &Me First

Petitioner Exhibit 1031, Page 287

Although AutoPlay is typically the default menu item, you can define a different command to be the default by including the following line.

```
shell = verb
```

When the user double-clicks on the icon, the command associated with this entry will be carried out.

System Naming Conventions

Windows provides support for filenames up to 255 characters long. Use the long filename when displaying the name of a file. Avoid displaying the filename extension unless the user chooses the option to display extensions or when the file type is not registered.

The system automatically formats a filename correctly if you use the **SHGetFileInfo** or **GetFileTitle** function. For more information about these functions, see the documentation included in the Win32 SDK.

Because the system uses three-letter extensions to describe a file type, do not use extensions to distinguish different forms of the same file type. For example, if your application has a function that automatically backs up a file, name the backup file Backup of *filename.ext* (using its existing extension) or some reasonable equivalent, not *filename*.bak. The latter implies a change of the file's type. Similarly, do not use a Windows filename extension unless your file fits the type description.

Long filenames can include any character, except the following.

\ / : * ? < > | “

When your application automatically supplies a filename, use a name that communicates information about its creation. For example, files created by a particular application should use either the application-supplied type name or the short type name as a proposed name — for example, worksheet or document. When that file exists already in the target directory, add a number to the end of the proposed name — for example, Document (2). When adding numbers to the end of a proposed filename, use the first number of an ordinal sequence that does not conflict with an existing name in that directory.

Petitioner Exhibit 1031, Page 288

When saving a file, make certain you preserve the creation date of the file. For simple applications that open and save a file, this happens automatically. However, more sophisticated applications may create temporary files, delete the original file, and rename the temporary file to the original filename. In this case, the application needs to copy the creation date as well from the old file to the new, using the standard system functions. Certain system file management functionality may depend on the correct creation date.

When you create a filename, the system automatically creates an MS-DOS filename (alias) for a file. The system displays both the long filename and the MS-DOS filename in the property sheet for the file.

When a file is copied, use the words "Copy of" as part of the generated filename — for example, "Copy of Sample" for a file named "Sample." If the prefix "Copy of" is already assigned to a file, include a number in parentheses — for example, "Copy (2) of Sample". You can apply the same naming scheme to links, except the prefix is "Link to" or "Shortcut to."

It is also important to support UNC paths for identifying the location of files and folders. UNC paths and filenames have the following form.

*Server**Share**Directory**Filename.ext*

Using UNC names enables the user to directly browse the network and open files without having to make explicit network connections.

Wherever possible, display the full name of a file (without the extension). The number of characters you'll be able to display depends somewhat on the font used and the context in which the name is displayed. In any case, supply enough characters such that the user can reasonably distinguish between names. Take into account common prefixes such as "Copy of" or "Shortcut to". If you don't display the full name, indicate that it has been truncated by appending an ellipsis to the end of the name.

Petitioner Exhibit 1031, Page 289

You can use an ellipsis to abbreviate path names, in a displayable, but noneditable situation. In this case, include at least the first two entries of the beginning and the end of the path, using ellipses as notation for the names in between, as in the following example.

\\My Server\My Share\...\My Folder\My File

When using an icon to represent a network resource, label the icon with the name of the resource. If you need to show the network context rather than using a UNC path, label the resource using the following format.

Resource Name on *Computer Name*

Taskbar Integration

The system provides support for integrating your application's interface with the taskbar. The following sections provide information on some of the capabilities and appropriate guidelines.

Taskbar Window Buttons

When an application creates a primary window, the system automatically adds a taskbar button for that window and removes it when that window closes. For some specialized types of applications that run in the background, a primary window may not be necessary. In such cases, make certain you provide reasonable support for controlling the application using the commands available on the application's icon; it should not appear as an entry on the taskbar, however. Similarly, the secondary windows of an application should also not appear as a taskbar button.

The taskbar window buttons support drag and drop, but not in the conventional way. When the user drags an object over a taskbar window button, the system automatically restores the window. The user can then drop the object in the window.

Petitioner Exhibit 1031, Page 290

Status Notification

The system allows you to add status or notification information to the taskbar. Because the taskbar is a shared resource, add information to it that is of a global nature only or that needs monitoring by the user while working with other applications.

The **Shell_NotifyIcon** function provides support for adding a status item in the taskbar. For more information about this function, see the documentation included in the Win32 SDK.

Present status notification information in the form of a graphic supplied by your application, as shown in Figure 10.2.

Figure 10.2 Status indicator in the taskbar

When adding a status indicator to the taskbar, also support the following interactions:

- Provide a pop-up window that displays further information or controls for the object represented by the status indicator when the user clicks with button 1. For example, the audio (speaker) status indicator displays a volume control. Use a pop-up window to supply for further information rather than a dialog box, because the user can dismiss the window by clicking elsewhere. Position the pop-up window near the status indicator so the user can navigate to it quickly and easily. Avoid displaying other types of secondary windows because they require explicit user interaction to dismiss them. If there is no information or control that applies, do not display anything.

- Display a pop-up menu for the object represented by the status indicator when the user clicks on the status indicator with button 2. On this menu, include commands that bring up property sheets or other windows related to the status indicator. For example, the audio status indicator provides commands that display the audio properties as well as the Volume Control mixer application.

- Carry out the default command defined in the pop-up menu for the status indicator when the user double-clicks.

Petitioner Exhibit 1031, Page 291

- Display a tooltip that indicates what the status indicator represents. For example, this could include the name of the indicator, a value, or both.

- Provide the user an option to not display the status indicator, preferably in the property sheet for the object displaying the status indicator. This allows the user to determine which indicator to include in this shared space. You may need to provide an alternate means of conveying this status information when the user turns off the status indicator.

Message Notification

When your application's window is inactive but must display a message, rather than displaying a message box on top of the currently active window and switching the input focus, flash your application's title bar and taskbar window button to notify the user of the pending message. This avoids interfering with the user's current activity but lets the user know a message is waiting. When the user activates your application's window, the application can display a message box.

The **FlashWindow** function supports flashing your title bar and taskbar window button. For more information about this function, see the documentation included in the Win32 SDK.

Use the system setting for the cursor blink rate for your flash rate. This allows the user to control the flash rate to a comfortable frequency.

The **GetCaretBlinkTime** function provides access to the current cursor blink rate setting. For more information about this function, see the documentation included in the Win32 SDK.

Rather than flashing the button continually, you can flash the window button only a limited number of times (for example, three), then leave the button in the highlighted state, as shown in Figure 10.3. This lets the user know there is still a pending message.

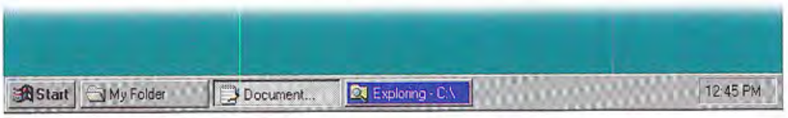


Figure 10.3 Flashing a taskbar button to notify a user of a pending message

This cooperative means of notification is preferable unless a message relates to the system integrity of the user's data, in which case your application may immediately display a system modal message box. In such cases, flush the input queue so that the user does not inadvertently select a choice in that message box.

Petitioner Exhibit 1031, Page 292

Application Desktop Toolbars

The system supports applications supplying their own desktop toolbars, also referred to as access bars or appbars, that operate similarly to the Windows taskbar. These may be docked to the edges of a screen and provide access to controls, such as buttons, for specific functions.

The system supports the same auto-hide behavior for application desktop toolbars as it does for the taskbar. This allows the desktop toolbar to only be visible when the user moves the pointer to the edge of the screen. The system also provides the "always on top" behavior used by the taskbar. When the user sets this property, the taskbar always appears on top (in the Z order) of any windows and also acts as a boundary for windows set to maximize to the display screen size.

Desktop toolbars can also be undocked and displayed as a palette window or redocked at a different edge of the screen. In the undocked, displayed as a palette window state, the toolbar no longer constrains other windows. However, if it supports the Always on Top property, it remains on top of other application windows.

For more information on the recommended behavior for undocking and redocking toolbars, see Chapter 7, "Menus, Controls, and Toolbars."

Before designing a desktop toolbar, consider whether your application's tasks really require one. Remember that a desktop toolbar will potentially affect the visible area for all applications. Only provide one for frequently used interfaces that can be applied across applications and always design it to be an optional interface, allowing the user to close it or otherwise configure it not to appear. You may also want to consider removing it when a specific application or applications are closed.

When creating your own desktop toolbar, model its behavior on the taskbar. Consider using the system's notification of when the taskbar's auto-hide or Always on Top property changes to apply a desktop toolbar you provide. If this does not fit your design, be certain to provide your own property sheet for setting these attributes for your desktop toolbar. Note that the system only supports auto-hide functionality for one desktop toolbar on each edge of the display. In addition, always provide a pop-up menu to access commands that apply to your desktop toolbar, such as Close, Move, Size, and Properties (but not the commands included on the desktop toolbar).

Petitioner Exhibit 1031, Page 293

You can choose to display a desktop toolbar when the user runs a specific application, or by creating a separate application and including a shortcut icon to it in the system's Startup folder. Preferably set the initial size and position of your desktop toolbar so that it does not interfere with other desktop toolbars or the taskbar. However, the system does support multiple desktop toolbars to be docked along the same edge of the display screen. When docking on the same edge as the taskbar, the system places the taskbar on the outermost edge.

Your desktop toolbar can include any type of control. A desktop toolbar can also be a drag and drop target. Follow the recommendations outlined in this guide for supporting appropriate interaction.

Full-Screen Display

Although the taskbar and application desktop toolbars normally constrain or clip windows displayed on the screen, you can define a window to the full extent of the display screen. Because this is not the typical form of interaction, only consider using full-screen display for very special circumstances, such as a slide presentation, and only when the user explicitly chooses a command for this purpose. Make certain you provide an easy way for the user to return to normal display viewing. For example, you can display an on-screen button when the user moves the pointer that restores the display when the user clicks it. In addition, keyboard interfaces, like ALT+TAB and ESC, should automatically restore the display.

Remember that desktop toolbars, including the taskbar, should support auto-hide options that allow the user to configure them to reduce their visual impact on the screen. Consider whether this auto-hide capability may be sufficient before designing your application to require a full-screen presentation. Advising the user to close or hide desktop toolbars may provide you with sufficient space without having to use the full display screen.

Petitioner Exhibit 1031, Page 294

Recycle Bin Integration

The Recycle Bin provides a repository for deleted files. If your application includes a facility for deleting files, support the Recycle Bin interface. You can also support deletion to the Recycle Bin for nonfile objects by first formatting the deleted data as a file by writing it to a temporary file and then calling the system functions that support the Recycle Bin.

The **SHFileOperation** function supports deletion using the Recycle Bin interface. For more information about this function, see the documentation included in the Win32 SDK.

Control Panel Integration

The Windows Control Panel includes special objects that let users configure aspects of the system. Your application can add Control Panel objects or add property pages to the property sheets of existing Control Panel objects.

Adding Control Panel Objects

You can create your own Control Panel objects. Most Control Panel objects supply only a single secondary window, typically a property sheet. Define your Control Panel object to represent a concrete object rather than an abstract idea.

Every Control Panel object is a dynamic-link library. To ensure that the dynamic-link library can be automatically loaded by the system, set the file's extension to .CPL and install it in the Windows System directory.

The system automatically caches information about Control Panel objects in order to provide quick user access, provided that the Control Panel object supports the correct system interfaces. For more information about developing Control Panel objects, see the documentation included in the Win32 SDK.

Adding to the Passwords Object

The Passwords object in Control Panel supplies a property sheet that allows the user to set security options and manage passwords for all password-protected services in the system. The Passwords object also allows you to add the name of a password-protected service to the object's list of services and use the Windows login password for all password-protected services in the system.

Petitioner Exhibit 1031, Page 295

When you add your service to the Passwords object, the name of the service appears in the Select Password dialog box that appears when the user chooses Change Other Passwords. The user can then change the password for the service by selecting the name and filling in the resulting dialog box. The name of your service also appears in the Change Windows Password dialog box; the name appears with a check box next to it. By setting the check box option, the user chooses to keep the password for the service identical to the Windows login password. Similarly, the user can disassociate the service from the Windows login password by toggling the check box setting off.

To add your service to the Passwords object, register your service under the **HKEY_LOCAL_MACHINE** key.

For more information about registering your password service, see the documentation included in the Win32 SDK.

```
HKEY_LOCAL_MACHINE
    System
        CurrentControlSet
            Control
                PwdProvider
                    Provider Name Value Name = Value
```

You can also add a page to the property sheet of the Passwords object to support other security-related services that the user can set as property values. Add a property page if your application provides security-related functionality beyond simple activation and changing of passwords. To add a property page, follow the conventions for adding shell extensions.

Petitioner Exhibit 1031, Page 296

Plug and Play Support

Plug and Play is a feature of Windows that, with little or no user intervention, automatically installs and configures drivers when their corresponding hardware peripherals are plugged into a PC. This feature applies to peripherals designed according to the Plug and Play specification. Supporting and appropriately adapting to Plug and Play hardware change can make your application easier to use. Following are some examples of supporting Plug and Play:

- Resizing your windows and toolbars relevant to screen size changes.
- Prompting users to shut down and save their data when the system issues a low power warning.
- Warning users about open network files when undocking their computers.
- Saving and closing files appropriately when users eject or remove removable media or storage devices or when network connections are broken.

System Settings and Notification

The system provides standard metrics and settings for user interface aspects, such as colors, fonts, border width, and drag rectangle (used to detect the start of a drag operation). The system also notifies running applications when its settings change. When your application starts up, query the system to set your application's user interface to match the system parameters to ensure visual and operational consistency. Also, design your application to adjust itself appropriately when the system notifies it of changes to these settings.

The **GetSystemMetrics**, **GetSysColor**, and **SystemParametersInfo** functions and the WM_SETTINGSCHANGE message are important to consider when supporting standard system settings. For more information about these system interfaces, see the documentation included in the Win32 SDK.

Petitioner Exhibit 1031, Page 297

Modeless Interaction

When designing your application, try to ensure that it is as interactive and nonmodal as possible. Here are some suggested ways of doing this:

- Use modeless secondary windows wherever possible.
- Segment processes, like printing, so you do not need to load the entire application to perform the operation.
- Make long processes run in the background, keeping the foreground interactive. For example, when something is printing, it should be possible to minimize the window even if the document cannot be altered. The multitasking support of Windows provides for defining separate processes, or *threads*, in the background.

For more information about threads, see the documentation included in the Win32 SDK.

Petitioner Exhibit 1031, Page 298

CHAPTER

Working with OLE Embedded and OLE Linked Objects

Microsoft OLE provides a set of system interfaces that enables users to combine objects supported by different applications. This chapter outlines guidelines for the interface for OLE embedded and OLE linked objects; you can apply many of these guidelines to any implementation of containers and their components.

The Interaction Model

As data becomes the major focus of interface design, its content is what occupies the user's attention, not the application managing it. In such a design, data is not limited to its native creation and editing environment; that is, the user is not limited to creating or editing data only within its associated application window. Instead, data can be transferred to other types of containers while maintaining its viewing and editing capability in the new container. Compound documents are a common example and illustration of the interaction between containers and their components, but they are not the only expression of this kind of object relationship that OLE can support.

Figure 11.1 shows an example of a compound document. The document includes word-processing text, tabular data from a spreadsheet, a sound recording, and pictures created in other applications.

Petitioner Exhibit 1031, Page 299

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 repertoire. 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

Multimedia Mozart: The Dissonant Quartet

The Voyager Company
Microsoft

In the words of author and music scholar Robert Winter, the string quartet in the eighteenth century was regarded as one of the "most sublime forms of communication." The String Quartet in C Major is no exception. Discover the power and the beauty of this music with Microsoft Multimedia Mozart: *The Dissonant Quartet,* and enter the world in which Mozart created his most memorable masterpieces. Sit back and enjoy *The Dissonant Quartet* in its entirety, or browse around, exploring its themes and emotional dynamics in depth. View the entire piece in a single-screen overview with the *Pocket Audio Guide.*

Multimedia Stravinsky: The Rite of Spring

The Voyager Company
Microsoft

Multimedia Stravinsky: *The Rite of Spring* offers you an in-depth look at this controversial composition. Author Robert Winter provides a fascinating commentary that follows the music, giving you greater understanding of the subtle dynamics of the instruments and powerful techniques of Stravinsky. You'll also have the opportunity to discover the ballet that accompanied *The Rite of Spring* in performance. Choreographed by Sergei Diaghilev, the ballet was as unusual for its time as the music. To whet your appetite, play this audio clip.

Multimedia Beethoven: The Ninth Symphony

The Voyager Company
Microsoft

Multimedia Beethoven: *The Ninth Symphony* is one of a series of engaging, informative, and interactive musical explorations from Microsoft. It enables you to examine Beethoven's world and life, and explore the form and beauty of one of his foremost compositions. You can compare musical themes, hear selected orchestral instruments, and see the symphonic score come alive. Multimedia Beethoven: *The Ninth Symphony* is an extraordinary opportunity to learn while you listen to one of the world's musical treasures. Explore this inspiring work at your own pace in *A Close Reading.* As you listen to a superb performance of Beethoven's

The Audiophile Journal, June 1994 12

Figure 11.1 A compound document

Petitioner Exhibit 1031, Page 300