How was this music review created? First, a user created a document and typed the text, then moved, copied, or linked content from other documents. Data objects that, when moved or copied, retain their native, full-featured editing and operating capabilities in their new container are called *OLE embedded objects.*

A user can also link information. An *OLE linked object* represents or provides access to another object that is in another location in the same container or in a different, separate container.

Generally, containers support any level of nested OLE embedded and linked objects. For example, a user can embed a chart in a worksheet, which, in turn, can be embedded in a word-processing document. The model for interaction is consistent at each level of nesting.

Creating OLE Embedded and OLE Linked Objects

OLE embedded and linked objects are the result of transferring existing objects or creating new objects of a particular type.

Transferring Objects

Transferring objects into a document follows basic command and direct manipulation interaction methods. The following sections provide additional guidelines for these commands when you use them to create OLE embedded or linked objects.

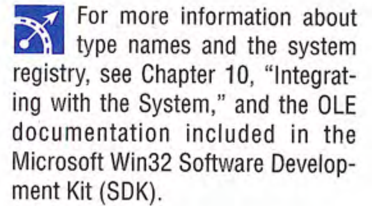
For more information about command transfer and direct manipulation transfer methods, see Chapter 5, "General Interaction Techniques."

The Paste Command

As a general rule, using the Paste command should result in the most complete representation of a transferred object; that is, the object is embedded. However, containers that directly handle the transferred object can accept it optionally as native data instead of embedding it as a separate object, or as a partial or transformed form of the object if that is more appropriate for the destination container.

Petitioner Exhibit 1031, Page 301

Use the format of the Paste command to indicate to the user how a transferred object is incorporated by a container. When the user copies a file object, if the container can embed the object, include the object's filename as a suffix to the Paste command. If the object is only a portion of a file, use the short type name — for example, Paste Worksheet or Paste Recording — as shown in Figure 11.2. A short type name can be derived from information stored in the registry. A Paste command with no name implies that the data will be pasted as native information.



For more information about type names and the system registry, see Chapter 10, "Integrating with the System," and the OLE documentation included in the Microsoft Win32 Software Development Kit (SDK).

Figure 11.2 The Paste command with short type name

The Paste Special Command

Supply the Paste Special command to give the user explicit control over pasting in the data as native information, an OLE embedded object, or an OLE linked object. The Paste Special command displays its associated dialog box, as shown in Figure 11.3. This dialog box includes a list box with the possible formats that the data can assume in the destination container.

Petitioner Exhibit 1031, Page 302

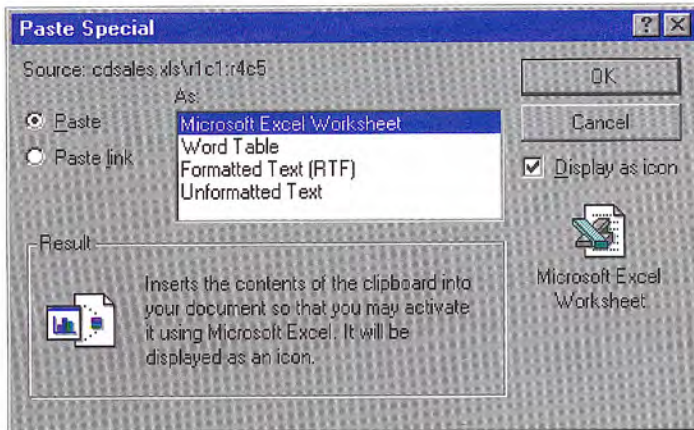
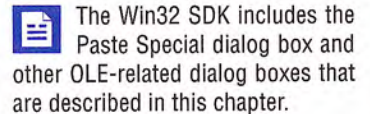


Figure 11.3 The Paste Special dialog box


The Win32 SDK includes the Paste Special dialog box and other OLE-related dialog boxes that are described in this chapter.

In the formats listed in the Paste Special dialog box, include the object's full type name first, followed by other appropriate native data forms. When a linked object has been cut or copied, precede its object type by the word "Linked" in the format list. For example, if the user copies a linked Microsoft Excel worksheet, the Paste Special dialog box shows "Linked Microsoft Excel Worksheet" in the list of format options because it inserts an exact duplicate of the original linked worksheet. Native data formats begin with the destination application's name and can be expressed in the same terms the destination identifies in its own menus. The initially selected format in the list corresponds to the format that the Paste command uses. For example, if the Paste command is displayed as Paste *Object Filename* or Paste *Short Type Name* because the data to be embedded is a file or portion of a file, this is the format that is initially selected in the Paste Special list box.

To support creation of a linked object, the Paste Special dialog box includes a Paste Link option. Figure 11.4 shows this option.

Petitioner Exhibit 1031, Page 303

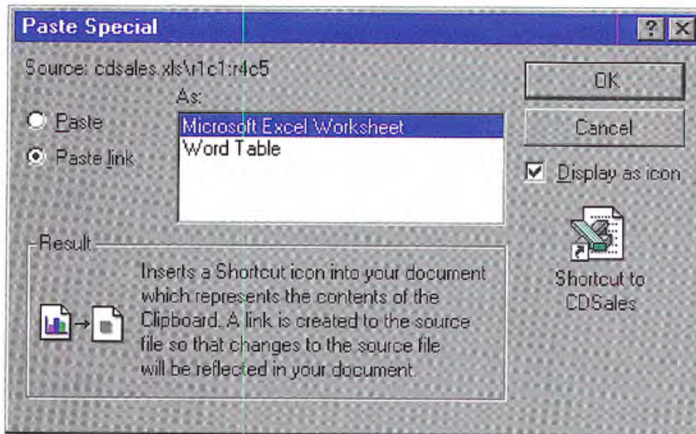


Figure 11.4 Paste Special dialog box with Paste Link option set

A Display As Icon check box allows the user to choose displaying the OLE embedded or linked object as an icon. At the bottom of the dialog box is a section that includes text and pictures that describe the result of the operation. Table 11.1 lists the descriptive text for use in the Paste Special dialog box.

Table 11.1 Descriptive Text for Paste Special Command

Function	Resulting text
Paste as an OLE embedded object.	"Inserts the contents of the Clipboard into your document so you that you may activate it using *CompanyName ApplicationName*."
Paste as an OLE embedded object so that it appears as an icon.	"Inserts the contents of the Clipboard into your document so you that you may activate it using *CompanyName ApplicationName* application. It will be displayed as an icon."
Paste as native data.	"Inserts the contents of the Clipboard into your document as *Native Type Name*. [*Optional additional Help sentence*.]"
Paste as an OLE linked object.	"Inserts a picture of the contents of the Clipboard into your document. Paste Link creates a link to the source file so that changes to the source file will be reflected in your document."

Petitioner Exhibit 1031, Page 304

(Continued)

Function	Resulting text
Paste as an OLE linked object so that it appears as a shortcut icon.	"Inserts a Shortcut icon into your document which represents the contents of the Clipboard. A link is created to the source file so that changes to the source file will be reflected in your document."
Paste as linked native data.	"Inserts the contents of the Clipboard into your document as *Native Type Name*. A link is created to the source file so that changes to the source file will be reflected in your document."

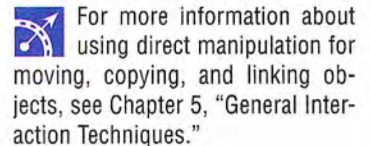
The Paste Link, Paste Shortcut, and Create Shortcut Commands

If linking is a common function in your application, you can optionally include a command that optimizes this process. Use a Paste Link command to support creating a linked object or linked native data. When using the command to create a linked object, include the name of the object preceded by the word "to" — for example, "Paste Link to Latest Sales." Omitting the name implies that the operation results in linked native data.

Use a Paste Shortcut command to support creation of a linked object that appears as a shortcut icon. You can also include a Create Shortcut command that creates a shortcut icon in the container. Apply these commands to containers where icons are commonly used.

Direct Manipulation

You should also support direct manipulation interaction techniques, such as drag and drop, for creating OLE embedded or linked objects. When the user drags a selection into a container, the container application interprets the operation using information supplied by the source, such as the selection's type and format, and by the destination container's own context, such as the container's type and its default transfer operation. For example, dragging a spreadsheet cell selection into a word-processing document can result in an OLE embedded table object. Dragging the same cell selection within the



For more information about using direct manipulation for moving, copying, and linking objects, see Chapter 5, "General Interaction Techniques."

Petitioner Exhibit 1031, Page 305

spreadsheet, however, would likely result in simply transferring the data in the cells. Similarly, the destination container in which the user drops the selection can also determine whether the dragging operation results in an OLE linked object.

For nondefault OLE drag and drop, the container application displays a pop-up menu with appropriate transfer commands at the end of the drag. The choices may include multiple commands that transfer the data in a different format or presentation. For example, as shown in Figure 11.5, a container application could offer the following choices for creating links: Link Here, Link *Short Type Name* Here, and Create Shortcut Here, respectively resulting in a native data link, an OLE linked object displayed as content, and an OLE linked object displayed as an icon. The choices depend on what the container can support.

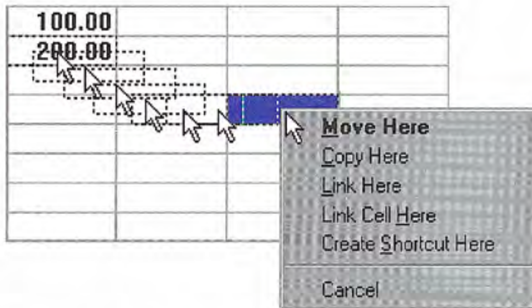


Figure 11.5 Containers can offer different OLE link options

The default appearance of a transferred object also depends on the destination container application. For most types of documents, make the default command one that results in the data or content presentation of the object (or in the case of an OLE linked object, a representation of the content), rather than as an icon. If the user chooses Create Shortcut Here as the transfer operation, display the transferred object as an icon. If the object cannot be displayed as content — for example, because it does not support OLE — always display the object as an icon.

Petitioner Exhibit 1031, Page 306

Transfer of Data to Desktop

The system allows the user to transfer data selection within a file to the desktop or folders providing that the application supports the OLE transfer protocol. For move or copy operations — using the Cut, Copy, and Paste commands or direct manipulation — the transfer operation results in a file icon called a *scrap*. A link operation also results in a shortcut icon that represents a shortcut into a document.

When the user transfers a scrap into a container supported by your application, integrate it as if it were being transferred from its original source. For example, if the user transfers a selected range of cells from a spreadsheet to the desktop, it becomes a scrap. If the user transfers the resulting scrap into a word-processing document, the document should incorporate the scrap as if the cells were transferred directly from the spreadsheet. Similarly, if the user transfers the scrap back into the spreadsheet, the spreadsheet should integrate the cells as if they had been transferred within that spreadsheet. (Typically, internal transfers of native data within a container result in repositioning the data rather than transforming it.)

Inserting New Objects

In addition to transferring objects, you can support user creation of OLE embedded or linked objects by generating a new object based on an existing object or object type and inserting the new object into the target container.

The Insert Object Command

Include an Insert Object command on the menu responsible for creating or importing new objects into a container, such as an Insert menu. If no such menu exists, use the Edit menu. When the user selects this command, display the Insert Object dialog box, as shown in Figure 11.6. This dialog box allows the user to generate new objects based on their object type or an existing file.

Petitioner Exhibit 1031, Page 307

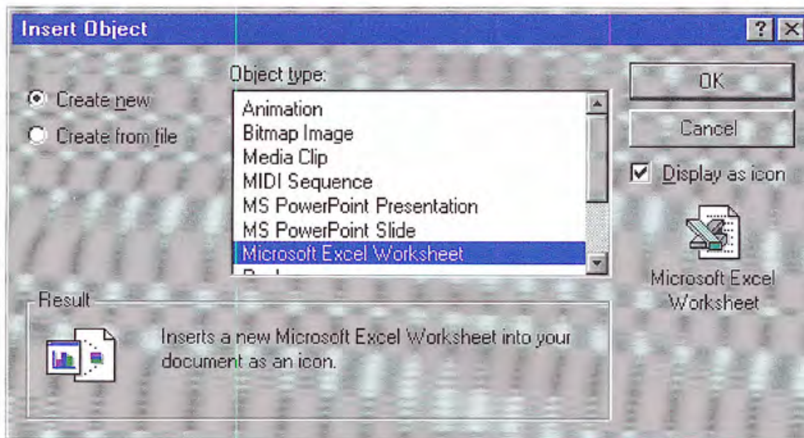
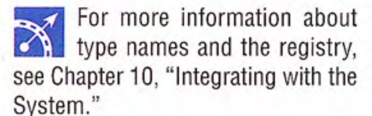


Figure 11.6 The Insert Object dialog box

The type list is composed of the type names of registered types. When the user selects a type from the list box and chooses the OK button, an object of the selected type is created and embedded.


For more information about type names and the registry, see Chapter 10, "Integrating with the System."

The user can also create an OLE embedded or linked object from an existing file, using the Create From File and Link options. When the user sets these options and chooses the OK button, the result is the same as directly copying or linking the selected file.

When the user chooses the Create From File option button, the Object Type list is removed, and a text box and Browse button appear in its place, as shown in Figure 11.7. Ignore any selection formerly displayed in the Object Type list box (shown in Figure 11.6).

Petitioner Exhibit 1031, Page 308

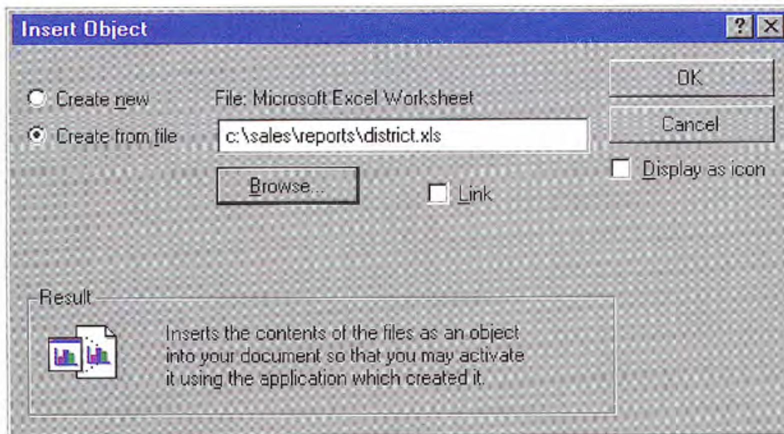


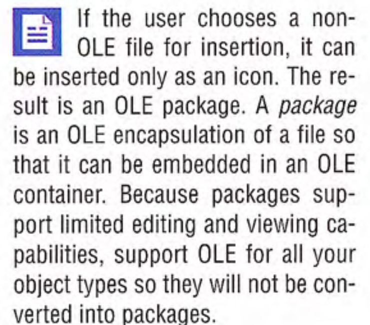
Figure 11.7 Creating an OLE embedded object from an existing file

The text box initially includes the current directory as the selection. The user can edit the current directory path when specifying a file. As an alternative, the Browse button displays an Open dialog box that allows the user to navigate through the file system to select a file. Use the file's type to determine the type of the resulting OLE object.

Use the Link check box to support the creation of an OLE linked object to the file specified. The Insert Object dialog box displays this option only when the user chooses the Create From File option. This means that a user cannot insert an OLE linked object when choosing the Create New option button, because linked objects can be created only from existing files.

The Display As Icon check box in the Insert Object dialog box enables the user to specify whether to display the OLE object as an icon. When this option is set, the icon appears beneath the check box. An OLE linked object displayed as an icon is the equivalent of a shortcut icon. It should appear with the link symbol over the icon.

At the bottom of the Insert Object dialog box, text and pictures describe the final outcome of the insertion. Table 11.2 outlines the syntax of descriptive text to use within the Insert Object dialog box.



If the user chooses a non-OLE file for insertion, it can be inserted only as an icon. The result is an OLE package. A *package* is an OLE encapsulation of a file so that it can be embedded in an OLE container. Because packages support limited editing and viewing capabilities, support OLE for all your object types so they will not be converted into packages.

Petitioner Exhibit 1031, Page 309

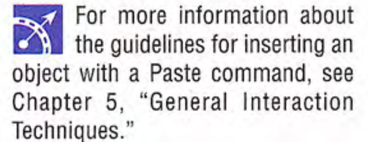
Table 11.2 Descriptive Text for Insert Object Dialog Box

Function	Resulting text
Create a new OLE embedded object based on the selected type.	"Inserts a new *Type Name* into your document."
Create a new OLE embedded object based on the selected type and display it as an icon.	"Inserts a new *Type Name* into your document as an icon."
Create a new OLE embedded object based on a selected file.	"Inserts the contents of the file as an object into your document so that you may activate it using the application which created it."
Create a new OLE embedded object based on a selected file (copies the file) and display it as an icon.	"Inserts the contents of the file as an object into your document so that you may activate it using the application which created it. It will be displayed as an icon."
Create an OLE linked object that is linked to a selected file.	"Inserts a picture of the file contents into your document. The picture will be linked to the file so that changes to the file will be reflected in your document."
Create an OLE linked object that is linked to a selected file and display it as a Shortcut icon.	"Inserts a Shortcut icon into your document which represents the file. The Shortcut icon will be linked to the original file, so that you can quickly open the original from inside your document."

You can also use the context of the current selection in the container to determine the format of the newly created object and the effect of it being inserted into the container. For example, an inserted graph can automatically reflect the data in a selected table. Use the following guidelines to support predictable insertion:

Petitioner Exhibit 1031, Page 310

- If an inserted object is not based on the current selection, follow the same conventions as for a Paste command and add or replace the selection depending on the context. For example, in text or list contexts, where the selection represents a specific insertion location, replace the active selection. For nonordered or Z-ordered contexts, where the selection does not represent an explicit insertion point, add the object, using the destination's context to determine where to place the object.


For more information about the guidelines for inserting an object with a Paste command, see Chapter 5, "General Interaction Techniques."

- If the new object is automatically connected (linked) to the selection (for example, an inserted graph based on selected table data), insert the new object in addition to the selection and make the inserted object the new selection.

After inserting an OLE embedded object, activate it for editing. However, if the user inserts an OLE linked object, do not activate the object.

Other Techniques for Inserting Objects

The Insert Object command provides support for inserting all registered OLE objects. You can include additional commands tailored to provide access to common or frequently used object types. You can implement these as additional menu commands or as toolbar buttons or other controls. These buttons provide the same functionality as the Insert Object dialog box, but perform more efficiently. Figure 11.8 illustrates two examples. The drawing button inserts a new blank drawing object; the graph button creates a new graph that uses the data values from a currently selected table.

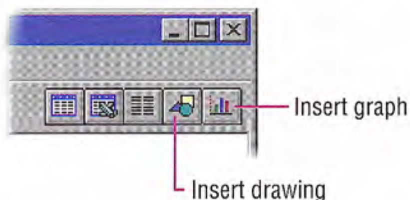


Figure 11.8 Using toolbar buttons for creating new objects

Petitioner Exhibit 1031, Page 311

Displaying Objects

While a container can control whether to display an OLE embedded or linked object in its content or icon presentation, the container requests the object to display itself. In the content presentation, the object may be visually indistinguishable from native objects, as shown in Figure 11.9.

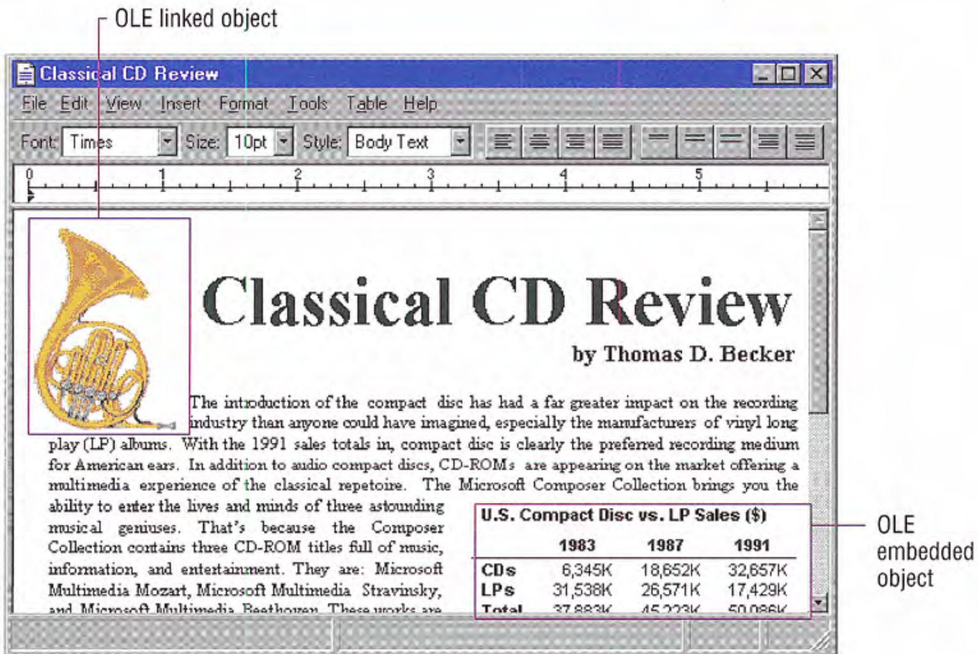
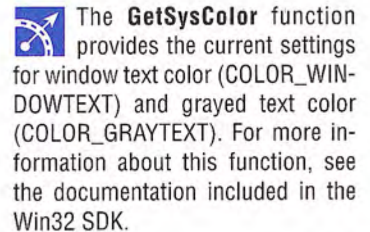


Figure 11.9 A compound document containing OLE objects

You may find it preferable to enable the user to visually identify OLE embedded or linked objects without interacting with them. To do so, you can include a Show Objects command that, when chosen, displays a solid border, one pixel wide, drawn in the window text

Petitioner Exhibit 1031, Page 312

color around the extent of an OLE embedded object and a dotted border around OLE linked objects (shown in Figure 11.10). If the container application cannot guarantee that an OLE linked object is up-to-date with its source because of an unsuccessful automatic update or a manual link, the system should draw a dotted border using the system grayed text color to suggest that the OLE linked object is out of date. The border should be drawn around a container's first-level objects only, not objects nested below this level.



The **GetSysColor** function provides the current settings for window text color (COLOR_WINDOWTEXT) and grayed text color (COLOR_GRAYTEXT). For more information about this function, see the documentation included in the Win32 SDK.

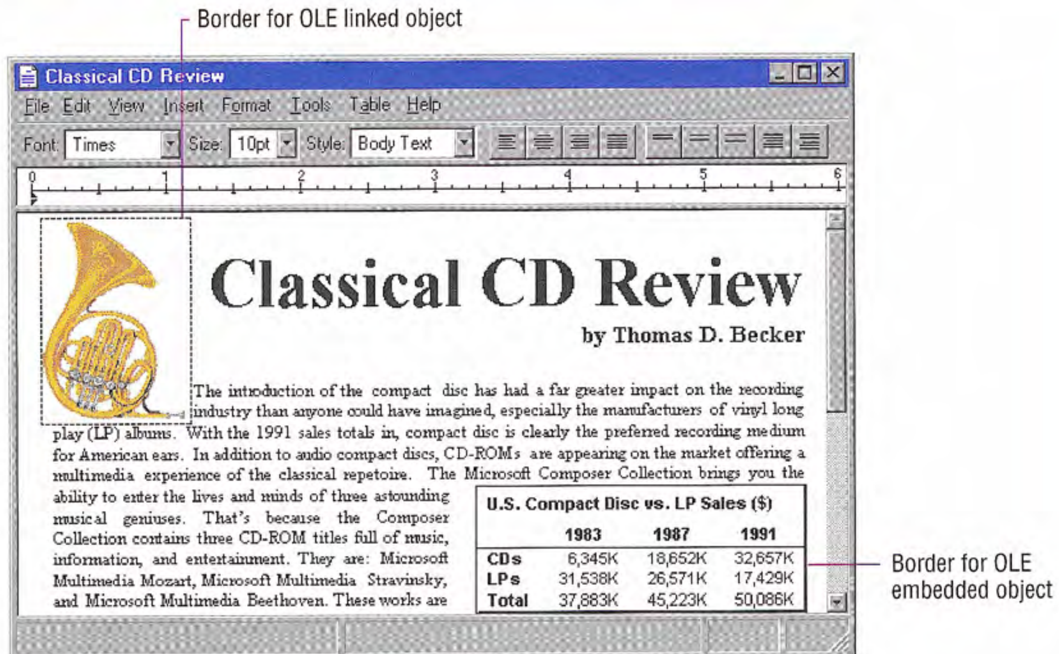


Figure 11.10 Identifying OLE objects using borders

If these border conventions are not adequate to distinguish OLE embedded and linked objects, you can optionally include additional distinctions; however, make them clearly distinct from the appearance for any standard visual states and distinguish OLE embedded from OLE linked objects.

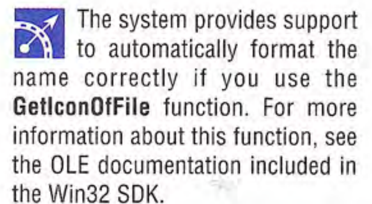
Petitioner Exhibit 1031, Page 313

Whenever the user creates an OLE linked or embedded object with the Display As Icon check box set, display the icon using the icon of its type, unless the user explicitly changes it. A linked icon also includes the shortcut graphic. If an icon is not registered in the registry for the object, use the system-generated icon.

An icon includes a label. When the user creates an OLE embedded object, define the icon's label to be one of the following, based on availability:

- The name of the object, if the object has an existing human-readable name such as a filename without its extension.
- The object's registered short type name (for example, Picture, Worksheet, and so on), if the object does not have a name.
- The object's registered full type name (for example, a bitmap image, a Microsoft Excel Worksheet), if the object has no name or registered short type name.
- "Document" if an object has no name, a short type name, or a registered type name.

When an OLE linked object is displayed as an icon, define the label using the source filename as it appears in the file system, preceded by the words "Shortcut to" — for example, "Shortcut to Annual Report." The path of the source is not included. Avoid displaying the filename extension unless the user chooses the system option to display extensions or the file type is not registered.



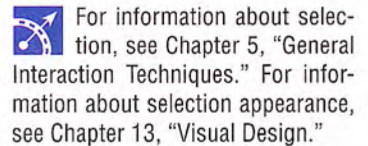
The system provides support to automatically format the name correctly if you use the **GetIconOfFile** function. For more information about this function, see the OLE documentation included in the Win32 SDK.

When the user creates an OLE object linked to only a portion of a document (file), follow the same conventions for labeling the shortcut icon. However, because a container can include multiple links to different portions of the same file, you may want to provide further identification to differentiate linked objects. You can do this by appending a portion of the end of the link path (moniker). For example, you may want to include everything from the end of the path up to the last or next to last occurrence of a link path delimiter. OLE applications should use the exclamation point (!) character for identifying a data range. However, the link path may include other types of delimiters. Be careful when deriving an identifier from the link path to format the additional information using only valid filename characters so that if the user transfers the shortcut icon to a folder or the desktop, the name can still be used.

Petitioner Exhibit 1031, Page 314

Selecting Objects

An OLE embedded or linked object follows the selection behavior and appearance techniques supported by its container; the container application supplies the specific appearance of the object. For example, Figure 11.11 shows how the linked drawing of a horn is handled as part of a contiguous selection in the document.



For information about selection, see Chapter 5, "General Interaction Techniques." For information about selection appearance, see Chapter 13, "Visual Design."

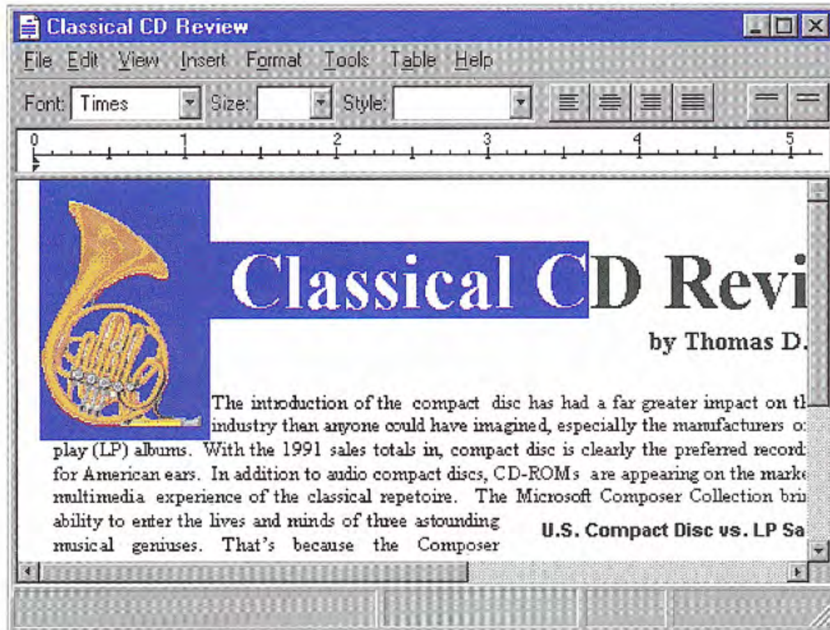


Figure 11.11 An OLE linked object as part of a multiple selection

When the user individually selects the object, display the object with an appropriate selection appearance. For example, for the content view of an object, display it with handles, as shown in Figure 11.12. For OLE linked objects, overlay the content view's lower left corner with the shortcut graphic. In addition, if your application's window includes a status bar that displays messages, display an appropriate description of how to activate the object (see Table 11.3 later in this chapter).

Petitioner Exhibit 1031, Page 315

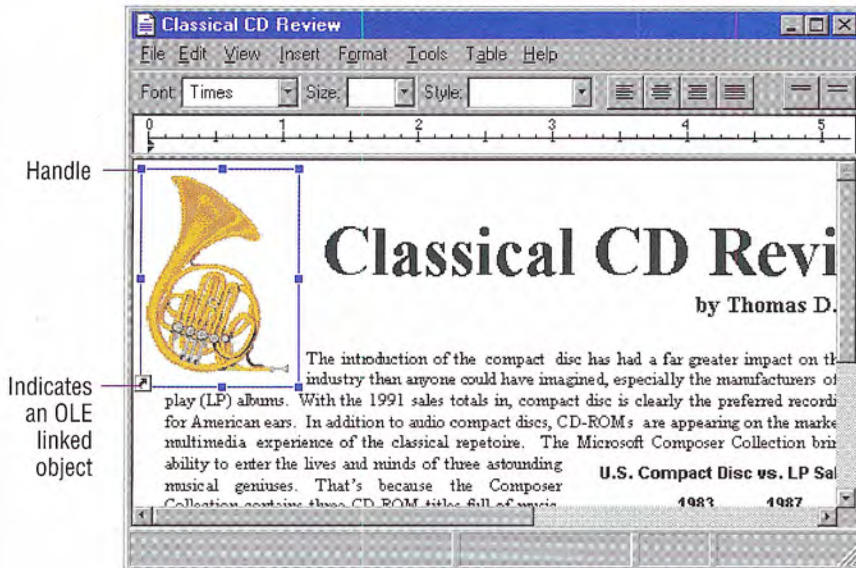


Figure 11.12 An individually selected OLE linked object

When the object is displayed as an icon, use the same selection appearance as for selected icons in folders and on the desktop, as shown in Figure 11.13.

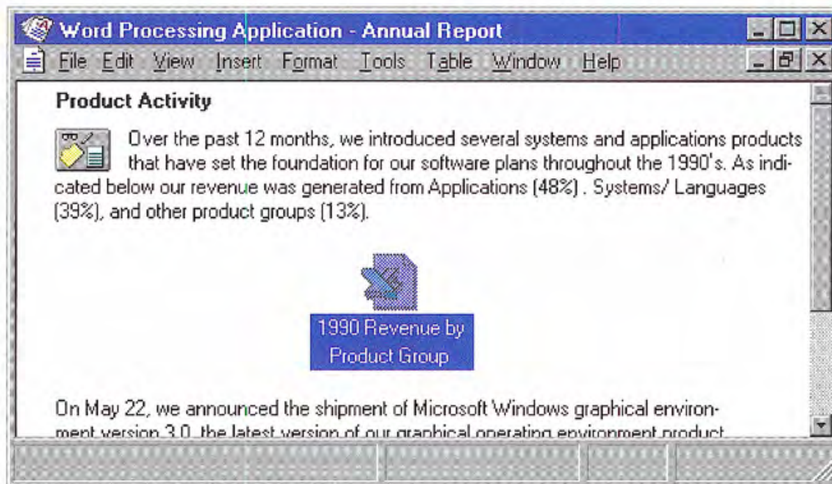
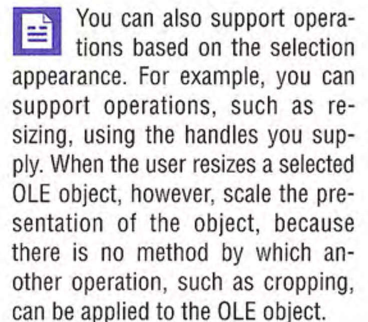


Figure 11.13 A selected OLE object displayed as an icon

Accessing Commands for Selected Objects

A container application always displays the commands that can be applied to its objects. When the user selects an OLE embedded or linked object as part of the selection of native data in a container, enable commands that apply to the selection as a whole. When the user individually selects the object, enable only commands that apply specifically to the object. The container application retrieves these commands from what has been registered by the object's type in the registry and displays these commands in the menus that are supplied for the object. If your application includes a menu bar, include the selected object's commands on a submenu of the Edit menu, or as a separate menu on the menu bar. Use the name of the object as the text for the menu item. If you use the short type name as the name of the object, add the word "Object." For an OLE linked object, use the short type name, preceded by the word "Linked." Figure 11.14 shows these variations.



You can also support operations based on the selection appearance. For example, you can support operations, such as resizing, using the handles you supply. When the user resizes a selected OLE object, however, scale the presentation of the object, because there is no method by which another operation, such as cropping, can be applied to the OLE object.

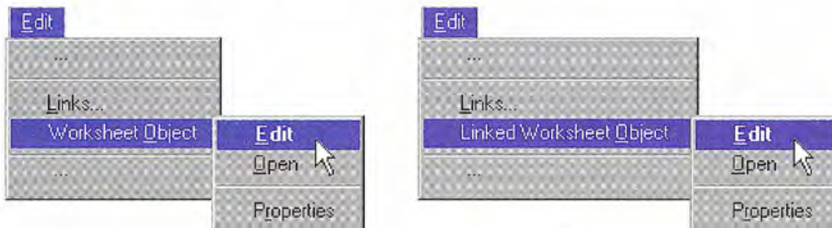


Figure 11.14 Drop-down menus for selected OLE object

Define the first letter of the word "Object", or its localized equivalent, as the access character for keyboard users. When no object is selected, display the command with just the text, "Object", and disable it.

A container application should also provide a pop-up menu for a selected OLE object (shown in Figure 11.15), displayed using the standard interaction techniques for pop-up menus (clicking with mouse button 2). Include on this menu the commands that apply to the object as a whole as a unit of content, such as transfer commands and the object's registered commands. In the pop-up menu, display the object's registered commands as individual menu items rather than in a cascading menu. It is not necessary to include the object's name or the word "Object" as part of the menu item text.

Petitioner Exhibit 1031, Page 317

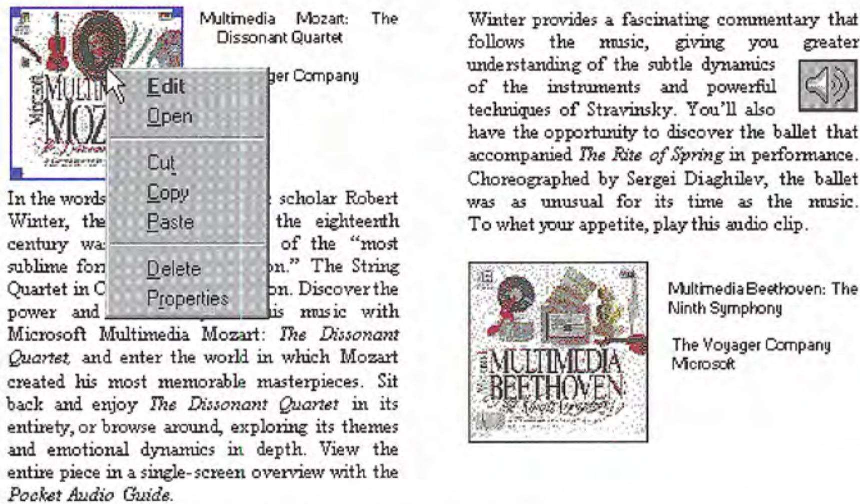


Figure 11.15 Pop-up menu for an OLE embedded picture

In the drop-down menu and the pop-up menu, include a Properties command. You can also include commands that depend on the state of the object. For example, a media object that uses Play and Rewind as operations disables Rewind when the object is at the beginning of the media object.

If an object's type is not registered, you still supply any commands that can be appropriately applied to the object as content, such as transfer commands, alignment commands, and an Edit and Properties command. When the user chooses the Edit command, display the system-supplied message box, as shown in Figure 11.41. This message box provides access to a dialog box that enables the user to choose from a list of applications that can operate on the type or convert the object's type.

Petitioner Exhibit 1031, Page 318

Activating Objects

Although selecting an object provides access to commands applicable to the object as a whole, it does not provide access to editing the content of the object. The object must be activated in order to provide user interaction with the internal content of the object. There are two basic models for activating objects: outside-in activation and inside-out activation.

Outside-in Activation

Outside-in activation requires that the user choose an explicit activation command. Clicking, or some other selection operation, performed on an object that is already selected simply reselects that object and does not constitute an explicit action. The user activates the object by using a particular command such as Edit or Play, usually the object's default command. Shortcut actions that correspond to these commands, such as double-clicking or pressing a shortcut key, can also activate the object. Most OLE container applications employ this model because it allows the user to easily select objects and reduces the risk of inadvertently activating an object whose underlying code may take a significant amount of time to load and dismiss.

When supporting outside-in activation, display the standard pointer (northwest arrow) over an outside-in activated object within your container when the object is selected, but inactive. This indicates to the user that the outside-in object behaves as a single, opaque object. When the user activates the object, the object's application displays the appropriate pointer for its content. Use the registry to determine the object's activation command.

Inside-out Activation

With *inside-out activation*, interaction with an object is direct; that is, the object is activated as the user moves the pointer over the extent of the object. From the user's perspective, inside-out objects are indistinguishable from native data because the content of the object

Petitioner Exhibit 1031, Page 319

is directly interactive and no additional action is necessary. Use this method for the design of objects that benefit from direct interaction, or when activating the object has little effect on performance or use of system resources.

Inside-out activation requires closer cooperation between the container and the object. For example, when the user begins a selection within an inside-out object, the container must clear its own selection so that the behavior is consistent with normal selection interaction. An object supporting inside-out activation controls the appearance of the pointer as it moves over its extent and responds immediately to input. Therefore, to select the object as a whole, the user selects the border, or some other handle, provided by the object or its container. For example, the container application can support selection techniques, such as region selection that select the object.

Although the default behavior for an OLE embedded object is outside-in activation, you can store information in the registry that indicates that an object's type (application class) is capable of inside-out activation (OLEMISC_INSIDEOUT) and prefers inside-out behavior (OLEMISC_ACTIVATEWHENVISIBLE). You can set these values in a **MiscStatus** subkey, under the **CLSID** subkey of the **HKEY_CLASSES_ROOT** key.

For more information about how to access OLEMISC_ INSIDEOUT and OLEMISC_ACTIVATEWHENVISIBLE and the **IOleObject::GetMiscStatus** function, see the OLE documentation included in the Win32 SDK.

Container Control of Activation

The container application determines how to activate its component objects: either it allows the inside-out objects to handle events directly or it intercedes and only activates them upon an explicit action. This is true regardless of the capability or preference setting of the object. That is, even though an object may register inside-out activation, it can be treated by a particular container as outside-in. Use an activation style for your container that is most appropriate for its specific use and is in keeping with its own native style of activation so that objects can be easily assimilated.

Petitioner Exhibit 1031, Page 320

Regardless of the activation capability of the object, a container should always activate its content objects of the same type consistently. Otherwise, the unpredictability of the interface is likely to impair its usability. Following are four potential container activation methods and when to use them.

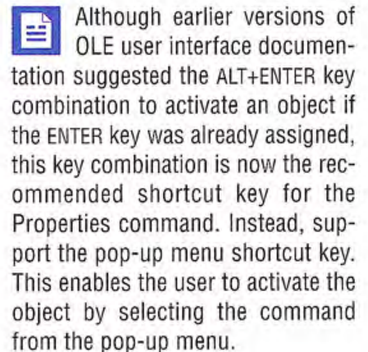
Activation method	When to use
Outside-in throughout	This is the most common design for containers that often embed large OLE objects and deal with them as whole units. Because many available OLE objects are not yet inside-out capable, most compound document editors support outside-in throughout to preserve uniformity.
Inside-out throughout	Ultimately, OLE containers will blend embedded objects with native data so seamlessly that the distinction dissolves. Inside-out throughout containers will become more feasible as increasing numbers of OLE objects support inside-out activation.
Outside-in plus inside-out preferred objects	Some containers may use an outside-in model for large, foreign embeddings but also include some inside-out preferred objects as though they were native objects (by supporting OLEMISC_ACTIVATE-WHENVISIBLE). For example, an OLE document might present form control objects as inside-out native data while activating larger spreadsheet and chart objects as outside-in.
Switch between inside-out throughout and outside-in throughout	Visual programming and forms layout design applications may include design and run modes. In this type of environment, a container typically holds an object that is capable of inside-out activation (if not preferable) and alternates between outside-in throughout when designing and inside-out throughout when running.

Petitioner Exhibit 1031, Page 321

OLE Visual Editing of OLE Embedded Objects

One of the most common uses for activating an object is editing its content in its current location. Supporting this type of in-place interaction is called *OLE visual editing*, because the user can edit the object within the visual context of its container.

Unless the container and the object both support inside-out activation, the user activates an embedded object for visual editing by selecting the object and choosing its Edit command, either from a drop-down or pop-up menu. You can also support shortcut techniques. For example, by making Edit the object's default operation, the user can double-click to activate the object for editing. Similarly, you can support pressing the ENTER key as a shortcut for activating the object.



Although earlier versions of OLE user interface documentation suggested the ALT+ENTER key combination to activate an object if the ENTER key was already assigned, this key combination is now the recommended shortcut key for the Properties command. Instead, support the pop-up menu shortcut key. This enables the user to activate the object by selecting the command from the pop-up menu.

When the user activates an OLE embedded object for visual editing, the user interface for its content becomes available and blended into its container application's interface. The object can display its frame *adornments*, such as row or column headers, handles, or scroll bars, outside the extent of the object and temporarily cover neighboring material. The object's application can also change the menu interface, which can range from adding menu items to existing drop-down menus to replacing entire drop-down menus. The object can also add toolbars, status bars, supplemental palette windows, and provide pop-up menus for selected content.

The degree of interface blending varies based on the nature of the OLE embedded object. Some OLE embedded objects may require extensive support and consequently result in dramatic changes to the container application's interface. Finer grain objects that emulate the native components of a container may have little or no need to make changes in the container's user interface. The container always determines the degree to which an OLE embedded object's interface can be blended with its own, regardless of the capability or preference of the OLE embedded object. A container application that provides its own interface for an OLE embedded object can suppress an OLE embedded object's own interface. Figure 11.16 shows how the interface might appear when its embedded worksheet is active.

Petitioner Exhibit 1031, Page 322

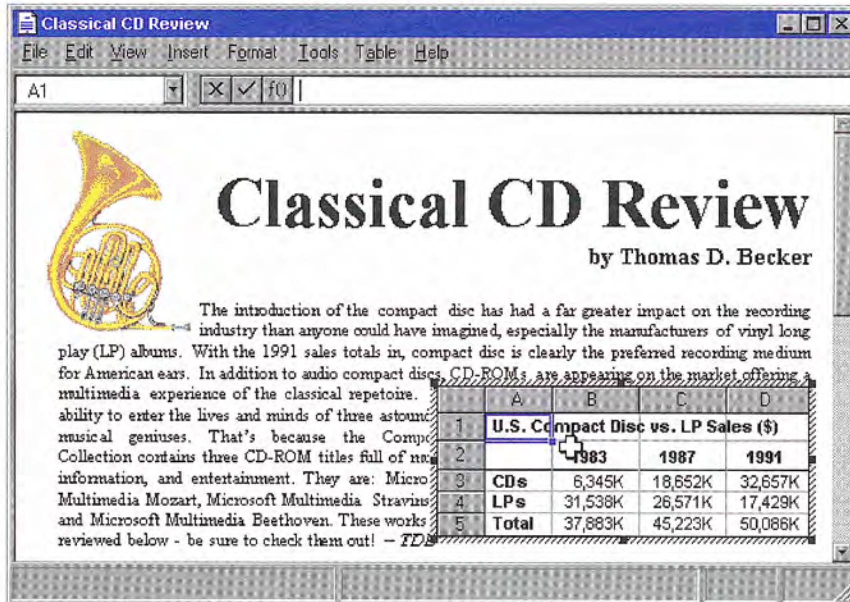


Figure 11.16 An embedded worksheet activated for OLE visual editing

When the user activates an OLE embedded object, avoid changing the view and position of the rest of the content in the window. Although it may seem reasonable to scroll the window and thereby preserve the content's position, doing so can disturb the user's focus, because the active object shifts down to accommodate a new toolbar and shifts back up when it is deactivated. An exception may be when the activation exposes an area in which the container has nothing to display. However, even in this situation, you may wish to render a visible region or filled area that corresponds to the background area outside the visible edge of the container so that activation keeps the presentation stable.

Activation does not affect the title bar. Always display the top-level container's name. For example, when the worksheet shown in Figure 11.16 is activated, the title bar continues to display the name of the document in which the worksheet is embedded and not the name of the worksheet. You can provide access to the name of the worksheet by supporting property sheets for your OLE embedded objects.

Petitioner Exhibit 1031, Page 323

A container can contain multiply nested OLE embedded objects. However, only a single level is active at any one time. Figure 11.17 shows a document containing an active embedded worksheet with an embedded graph of its own. Clicking on the graph merely selects it as an object within the worksheet.

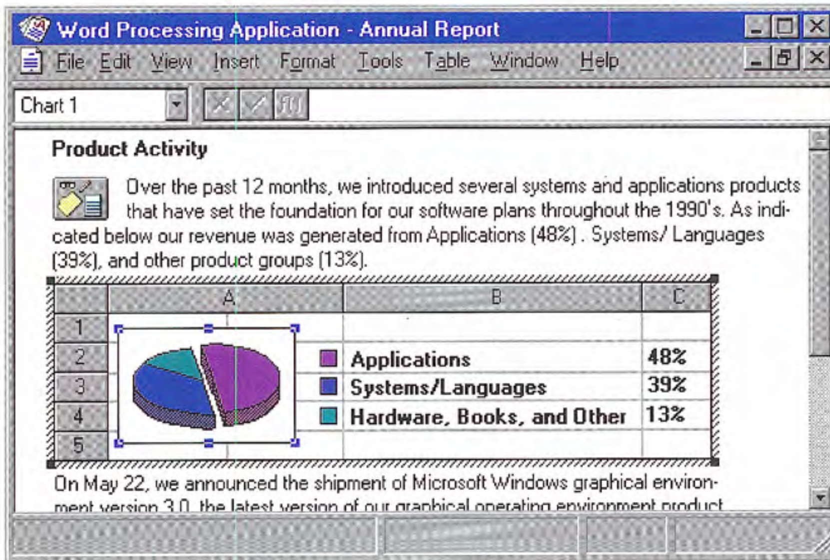


Figure 11.17 A selected graph within an active worksheet

Activating the embedded graph, for example, by choosing the graph's Edit command, activates the object for OLE visual editing, displaying the graph's menus in the document's menu bar. This is shown in Figure 11.18. At any given time, only the interface for the currently active object and the topmost container are presented; intervening parent objects do not remain visibly active.

Petitioner Exhibit 1031, Page 324

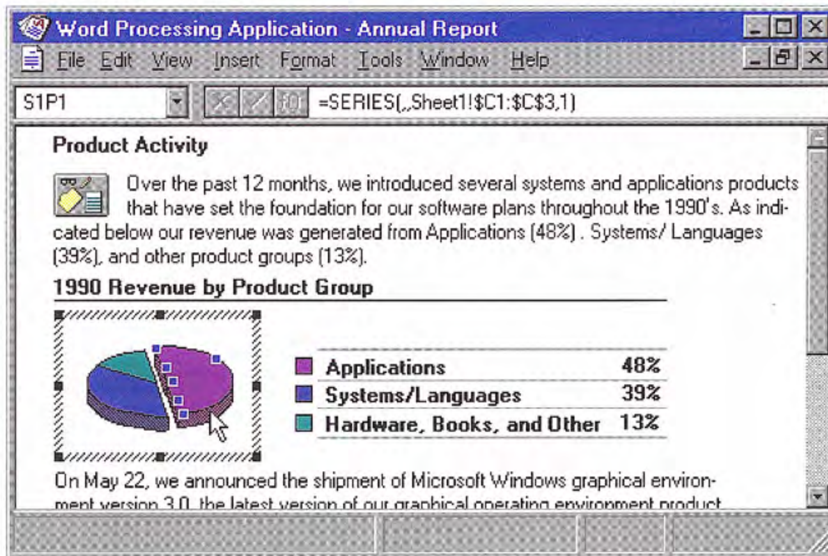


Figure 11.18 An active graph within a worksheet

An OLE embedded object should support OLE visual editing at any view magnification level because its container's view can be scaled arbitrarily. If an object cannot accommodate OLE visual editing in its container's current view scale, or if its container does not support OLE visual editing, open the object into a separate window for editing. For more information about opening OLE embedded objects, see the section, "Opening Embedded Objects," later in this chapter.

For any user interaction outside the extent of an active object, such as when the user selects or activates another object in the container, deactivate the current object and give the focus to the new object. This is also true for an object that is nested in the currently active object. An OLE embedded object application should also support user deactivation when the user presses the ESC key, after which it becomes the selected object of its container. If the object uses the ESC key at all times for its internal operation, the SHIFT+ESC key should deactivate the object.

Petitioner Exhibit 1031, Page 325

Edits made to an active object become part of the container immediately and automatically, just like edits made to native data. Consequently, do not display an "Update changes?" message box when the object is deactivated. Remember that the user can abandon changes to the entire container, embedded or otherwise, if the topmost container includes an explicit command that prompts the user to save or discard changes to the container's file.

OLE embedded objects participate in the undo stack of the window in which they are activated. For more information about embedded objects and the undo stack, see the section, "Undo Operations for Active and Open Objects," later in this chapter.

While Edit is the most common command for activating an OLE embedded object for OLE visual editing, other commands can also create such activation. For example, when the user carries out a Play command on a video clip, you can display a set of commands that allow the user to control the clip (Rewind, Stop, and Fast Forward). In this case, the Play command provides a form of OLE visual editing.

The Active Hatched Border

If a container allows an OLE embedded object's user interface to change its user interface, then the active object's application displays a hatched border around itself to show the extent of the OLE visual editing context (shown in Figure 11.19). For example, if an active object places its menus in the topmost container's menu bar, display the active hatched border. The object's request to display its menus in the container's menu bar must be granted by the container application. If the object's menus do not appear in the menu bar (because the object did not require menus or the container refused its request for menu display), or the object is otherwise accommodated by the container's user interface, you need not display the hatched border. The hatched pattern is made up of 45-degree diagonal lines.

Petitioner Exhibit 1031, Page 326

	A	B	C	D
1	U.S. Compact Disc vs. LP Sales ($)			
2		1983	1987	1991
3	CDs	6,345K	18,652K	32,657K
4	LPs	31,538K	26,571K	17,429K
5	Total	37,883K	45,223K	50,086K

Figure 11.19 Hatched border around active OLE embedded objects

The active object takes on the appearance that is best suited for its own editing; for example, the object may display frame adornments, table gridlines, handles, and other editing aids. Because the hatched border is part of the object's territory, the active object defines the pointer that appears when the user moves the mouse over the border.

Clicking in the hatched pattern (and not on the handles) is interpreted by the object as clicking just inside the edge of the border of the active object. The hatched area is effectively a hot zone that prevents inadvertent deactivations and makes it easier to select the content of the embedded object.

Menu Integration

As the user activates different objects, different commands need to be accessed in the window's user interface. The following classification of menus — primary container menu, workspace menu, and active object menus — separates the interface based on menu groupings. This classification enhances the usability of the interface by defining the interface changes as the user activates or deactivates different objects.

Petitioner Exhibit 1031, Page 327

Primary Container Menu

The topmost or primary container viewed in a primary window controls the work area of that window. If the primary container includes a menu bar, it supplies at least one menu that includes commands that apply to the primary container as an entire unit. For example, for document file objects, use a File menu for this purpose, as shown in Figure 11.20. This menu includes document and file level commands such as Open, Save, and Print. Always display the primary container menu in the menu bar at all times, regardless of which object is active.

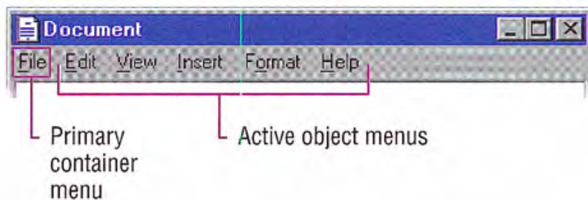


Figure 11.20 OLE visual editing menu layout

Workspace Menu

An MDI-style application also includes a workspace menu (typically labeled "Window") on its menu bar that provides commands for managing the document windows displayed within it, as shown in Figure 11.21. Like the primary container menu, the workspace menu should always be displayed, independent of object activation.

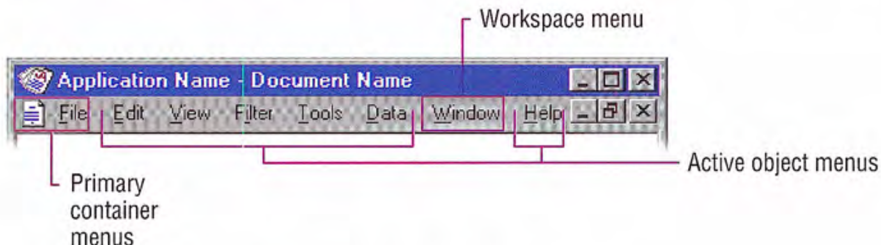


Figure 11.21 OLE visual editing menu layout for MDI

Petitioner Exhibit 1031, Page 328

Active Object Menus

Active objects can define menus that appear on the primary container's menu bar that operate on their content. Place commands for moving, deleting, searching and replacing, creating new items, applying tools, styles, and Help on these menus. As the name suggests, active object commands are executed by the currently active object and apply only within the extent of that object. If no embedded objects are active, but the window is active, the primary container should be considered the active object.

An active object's menus typically occupy the majority of the menu bar. Organize these menus following the same order and grouping that you display when the user opens the object into its own window. Avoid naming your active object menus File or Window, because primary containers often use those titles. Objects that use direct manipulation as their sole interface need not provide active object menus or alter the menu bar when activated.

The active object can display a View menu. However, when the object is active, include only commands that apply to the object. If the object's container requires its document or window-level "viewing" commands to be available while an object is active, place them on a menu that represents the primary container window's pop-up menu and on the Window menu — if present.

When designing the interface of selected objects within an active object, follow the same guidelines as that of a primary container and one of its selected OLE embedded objects; that is, the active object displays the commands of the selected object (as registered in the registry) either as submenus of its menus or as separate menus.

An active object also has the responsibility for defining and displaying pop-up menus for its content, with commands appropriate to apply to any selection within it. Figure 11.22 shows an example of a pop-up menu for a selection within an active bitmap image.

Petitioner Exhibit 1031, Page 329

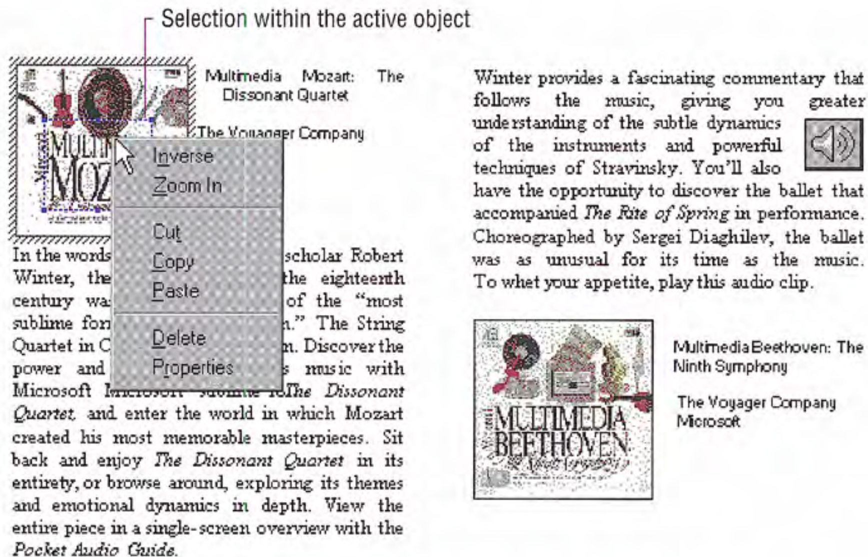


Figure 11.22 Example of pop-up menu for a selection in an active object

Keyboard Interface Integration

In addition to integrating the menus, you must also integrate the access keys and shortcut keys used in these menus.

Access Keys

The access keys assigned to the primary container's menu, an active object's menus, and MDI workspace menus should be unique. Following are guidelines for defining access keys for integrating these menu names:

- Use the first letter of the menu of the primary container as its access key character. Typically, this is "F" for File. Use "W" for a workspace's Window menu. Localized versions should use the appropriate equivalent.

- Use characters other than those assigned to the primary container and workspace menus for the menu titles of active OLE embedded objects. (If an OLE embedded object has previously existed as a standalone document, its corresponding application avoids these characters already.)

Petitioner Exhibit 1031, Page 330

- Define unique access keys for an object's registered commands and avoid characters that are potential access keys for common container-supplied commands, such as Cut, Copy, Paste, Delete, and Properties.

Despite these guidelines, if the same access character is used more than once, pressing an ALT+*letter* combination cycles through each command, selecting the next match each time it is pressed. To carry out the command, the user must press the ENTER key when it is selected. This is standard system behavior for menus.

Shortcut Keys

For primary containers and active objects, follow the shortcut key guidelines covered in this guide. In addition, avoid defining shortcut keys for active objects that are likely to be assigned to the container. For example, include the standard editing and transfer (Cut, Copy, and Paste) shortcut keys, but avoid File menu or system-assigned shortcut keys. There is no provision for registering shortcut keys for a selected object's commands.

For more information about defining shortcut keys, see Chapter 4, "Input Basics," and Appendix B, "Keyboard Interface Summary."

If a container and an active object share a common shortcut key, the active object captures the event. That is, if the user activates an OLE embedded object, its application code directly processes the shortcut key. If the active object does not process the key event, it is available to the container, which has the option to process it or not. This applies to any level of nested OLE embedded objects. If there is duplication between shortcut keys, the user can always direct the key based on where the active focus is by activating that object. To direct a shortcut key to the container, the user deactivates an OLE embedded object — for example, by selecting in the container — but outside the OLE embedded object. Activation, not selection, of an OLE embedded object allows it to receive the keyboard events. The exception is inside-out activation, where activation results from selection.

Petitioner Exhibit 1031, Page 331

Toolbars, Frame Adornments, and Palette Windows

Integrating drop-down and pop-up menus is straightforward because they are confined within a particular area and follow standard conventions. Toolbars, frame adornments (as shown in Figure 11.23), and palette windows can be constructed less predictably, so it is best to follow a replacement strategy when integrating these elements for active objects. That is, toolbars, frame adornments, and palette windows are displayed and removed as entire sets rather than integrated at the individual control level — just like menu titles on the menu bar.

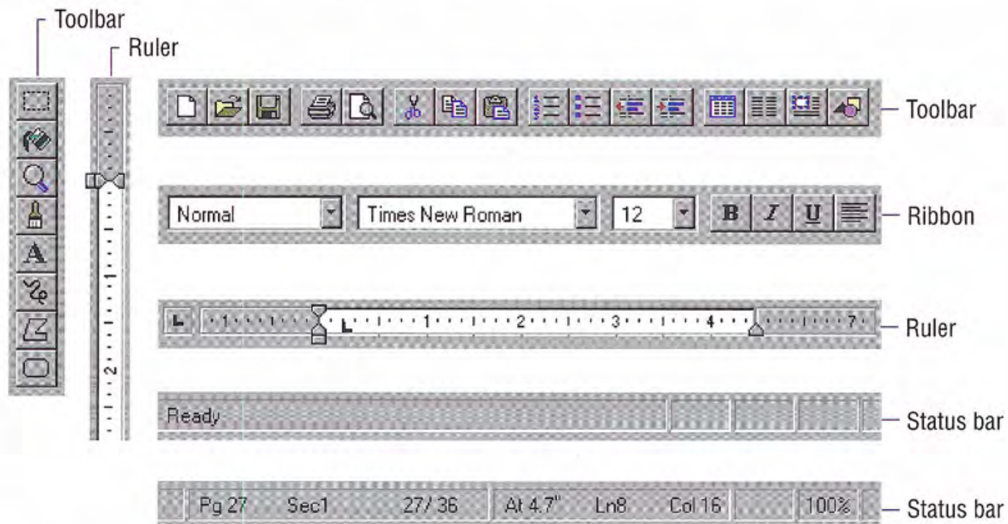


Figure 11.23 Examples of toolbars, status bars, and frame adornments

When the user activates an object, the object application requests a specific area from its container in which to post its tools. The container application determines whether to:

- Replace its tool (or tools) with the tools of the object, if the requested space is already occupied by a container tool.
- Add the object's tool (or tools), if a container tool does not occupy the requested space.
- Refuse to display the tool (or tools) at all. This is the least desirable method.

Petitioner Exhibit 1031, Page 332

Toolbars, frame adornments, and palette windows are all basically the same interfaces — they differ primarily in their location and the degree of shared control between container and object. There are four locations in the interface where these types of controls reside, and you determine their location by their scope. Figure 11.24 shows possible positions for interface controls.

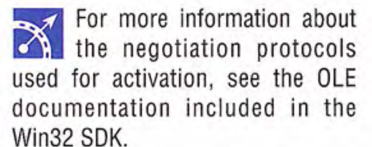
Location	Description
Object frame	Place object-specific controls, such as a table header or a local coordinate ruler, directly adjacent to the object itself for tightly coupled interaction between the object and its interface. An object (such as a spreadsheet) can include scrollbars if its content extends beyond the boundaries of its frame.
Pane frame	Locate view-specific controls at the pane level. Rulers and viewing tools are common examples.
Document (primary container) window frame	Attach tools that apply to the entire document (or documents in the case of an MDI window) just inside any edge of its primary window frame. Popular examples include ribbons, drawing tools, and status lines.
Windowed	Display tools in a palette window — this allows the user to place them as desired. A palette window typically floats above the primary window and any other windows of which it is part.

Petitioner Exhibit 1031, Page 333

Figure 11.24 Possible locations for interface controls

When determining where to locate a tool area, avoid situations that cause the view to shift up and down as different-sized tool areas are displayed or removed as the user activates different objects. This can be disruptive to the user's task.

Because container tool areas can remain visible while an object is active, they are available to the user simply by interacting with them — this can reactivate the container application. The container determines whether to activate or leave the object active. If toolbar buttons of an active object represent a primary container or workspace commands, such as Save, Print, or Open, disable them.



For more information about the negotiation protocols used for activation, see the OLE documentation included in the Win32 SDK.

Petitioner Exhibit 1031, Page 334

As the user resizes or scrolls its container's area, an active object and its toolbar or frame adornments placed on the object frame are clipped, as is all container content. These interface control areas lie in the same plane as the object. Even when the object is clipped, the user can still edit the visible part of the object in place and while the visible frame adornments are operational.

Some container applications scroll at certain increments that may prevent portions of an OLE embedded object from being visually edited. For example, consider a large picture embedded in a worksheet cell. The worksheet scrolls vertically in complete row increments; the top of the pane is always aligned with the top edge of a row. If the embedded picture is too large to fit within the pane at one time, its bottom portion is clipped and consequently never viewed or edited in place. In cases like this, the user can open the picture into its own window for editing.

Window panes clip frame adornments of nested embedded objects, but not by the extent of any parent object. Objects at the very edge of their container's extent or boundary potentially display adornments that extend beyond the bounds of the container's defined area. In this case, if the container displays items that extend beyond the edge, display all the adornments; otherwise, clip the adornments at the edge of the container. Do not temporarily move the object within its container just to accommodate the appearance of an active embedded object's adornments. A pane-level control can potentially be clipped by the primary (or parent, in the case of MDI) window frame, and a primary window adornment or control is clipped by other primary windows.

Opening OLE Embedded Objects

The previous sections have focused on OLE visual editing — editing an OLE embedded object in place; that is, its current location is within its container. Alternatively, the user can also open embedded objects into their own window. This gives the user the opportunity of seeing more of the object or seeing the object in a different view state. Support this operation by registering an Open command for the object. When the user chooses the Open command of an object, it opens it into a separate window for editing, as shown in Figure 11.25.

Petitioner Exhibit 1031, Page 335

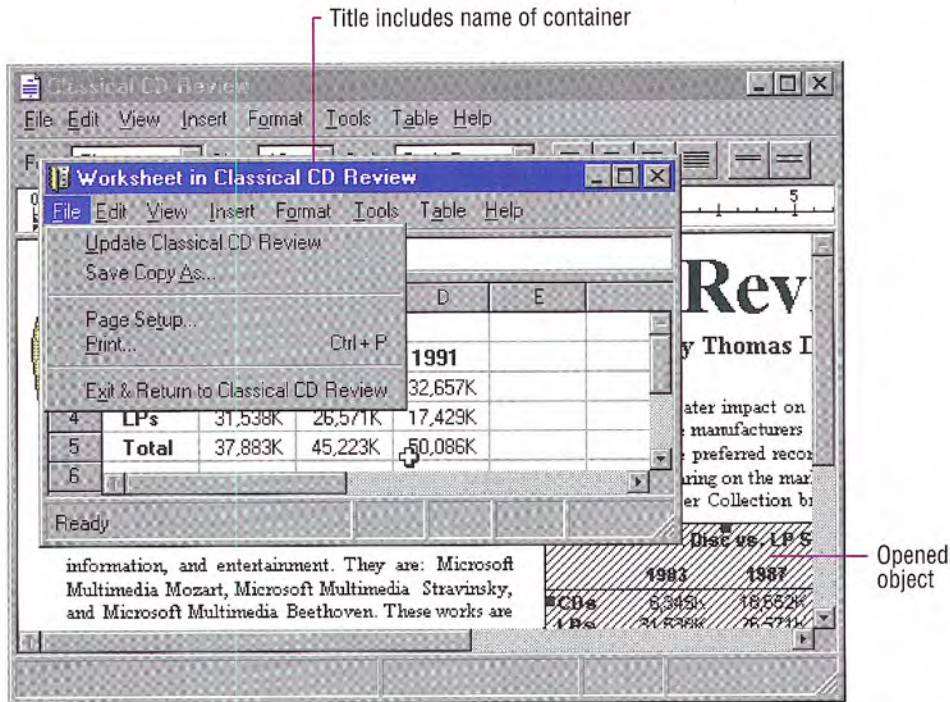


Figure 11.25 An opened OLE embedded worksheet

After opening an object, the container displays it masked with an "open" hatched (lines at a 45-degree angle) pattern that indicates the object is open in another window, as shown in Figure 11.26.

U.S. Compact Disc vs. LP Sales ($)			
	1983	1987	1991
CDs	6,345K	18,652K	32,657K
LPs	31,538K	26,571K	17,429K
Total	37,883K	45,223K	50,086K

Figure 11.26 An opened OLE embedded object

Format the title text for the open object's window as "*Object Name* in *Container Name*" (for example, "Sales Worksheet in Classical CD Review"). Including the container's name emphasizes that the object in the container and the object in the open window are considered the same object.

This convention for the title bar text applies only when the user opens an OLE embedded object. When the user activates an OLE embedded object for visual editing, do not change the title bar text.

Petitioner Exhibit 1031, Page 336

An open OLE embedded object represents an alternate window onto the same object within the container as opposed to a separate application that updates changes to the container document. Therefore, reflect edits immediately and automatically in the object in the window of its container. There is no need to display an update confirmation message upon exiting the open window. Nevertheless, you can still include an Update *Container Filename* command in the window of the open object to allow the user to request an update explicitly. This is useful if you cannot support frequent "real-time" image updates because of operational performance. In addition, when the user closes an open object's window, automatically update its presentation in the container's window. Provide a Close & Return To *Container Filename* or Exit & Return To *Container Filename* on the File menu replacing the Close or Exit command, as shown in Figure 11.25.

You can also include Import File and similar commands in the window of the open object. Treat importing a file into the window of the open embedded object the same as any change to the object.

If it has file operations, such as New or Open, remove these in the resulting window or replace them with commands, such as Import, to avoid severing the object's connection with its container. The objective is to present a consistent conceptual model; the object in the opened window is always the same as the one in the container. You can replace the Save As command with a Save Copy As command that displays the Save As dialog box, but unlike Save As, Save Copy As does not make the copied file the active file.

When the user opens an object, it is the selected object in the container; however, the user can change the selection in the container afterwards. Like any selected OLE embedded object, the container supplies the appropriate selection appearance together with the open appearance, as shown in Figure 11.27.

Handle

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

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

Figure 11.27 A selected open OLE embedded object

Petitioner Exhibit 1031, Page 337

The selected and open appearances apply only to the object's appearance on the display. If the user chooses to print the container while an OLE embedded object is open or active, use the presentation form of objects; neither the open nor active hatched pattern should appear in the printed document because neither pattern is part of the content.

While an OLE embedded object is open, it is still a functioning member of its container. It can still be selected or unselected, and can respond to appropriate container commands. At any time, the user may open any number of OLE embedded objects. When the user closes its container window, deactivate and close the windows for any open OLE embedded objects.

Editing an OLE Linked Object

An OLE linked object can be stored in a particular location, moved or copied, and has its own properties. Container actions can be applied to the OLE linked object as a unit of content. So an OLE container supplies commands, such as Cut, Copy, Delete, and Properties, and interface elements such as handles, drop-down and pop-up menu items, and property sheets, for the OLE linked objects it contains.

The container also provides access to the commands that activate the OLE linked object, including the commands that provide access to content represented by the OLE linked object. These commands are the same as those that have been registered for the link source's type. Because an OLE linked object represents and provides access to another object that resides elsewhere, editing an OLE linked object always takes the user back to the link source. Therefore, the command used to edit an OLE linked object is the same as the command of its linked source object. For example, the menu of a linked object can include both Open and Edit if its link source is an OLE embedded object. The Open command opens the embedded object, just as carrying out the command on the OLE embedded object does. The Edit command opens the container window of the OLE embedded object and activates the object for OLE visual editing.

Petitioner Exhibit 1031, Page 338

Figure 11.28 shows the result of opening a linked bitmap image of a horn. The image appears in its own window for editing. Note that changes made to the horn are reflected not only in its host container, the "Classical CD Review" document, but in every other document that contains an OLE linked object linked to that same portion of the "Horns" document. This illustrates both the power and the potential danger of using links in documents.

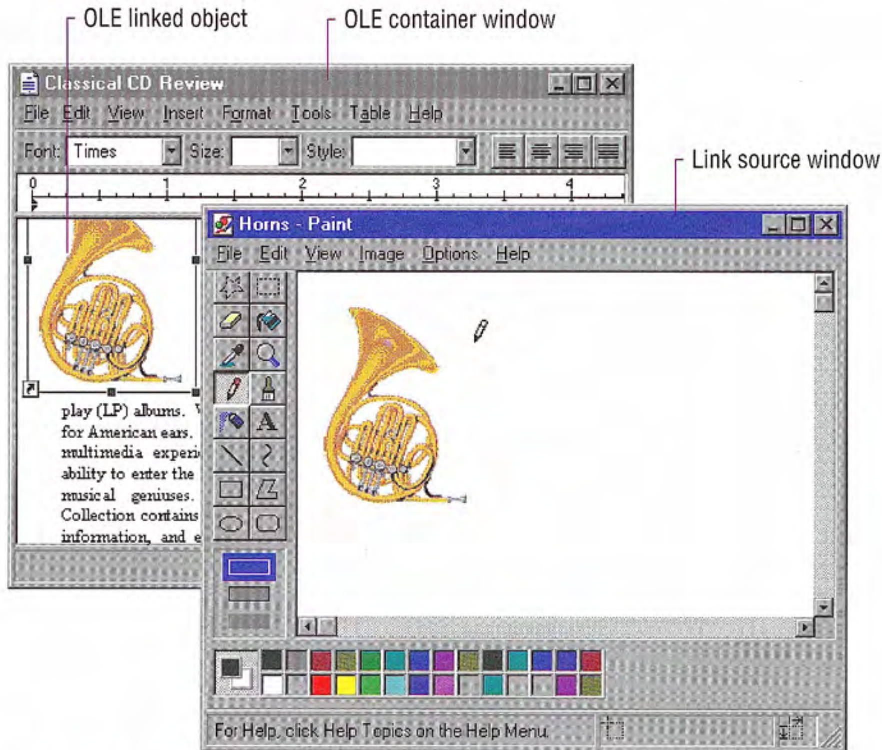


Figure 11.28 Editing a link source

At first glance, editing an OLE linked object seems to appear similar to an opened OLE embedded object; a separate primary window opens displaying the data. However, the container of an OLE linked object does not render the link representation using the open hatched pattern because the link source does not reside at this location. The

OLE linked object is not the real object, only a stand-in that enables the source to be visually present in other locations. Editing the linked object is functionally identical to opening the link source. Similarly, the title bar text of the link source's window does not use the convention as an open OLE embedded object because the link source is an independent object. Therefore, the windows operate and close independently of each other. If the link source's window is already visible, the OLE linked object notifies the link source to activate, bringing the existing window to the top of the Z order.

Note that the container of the OLE linked object does display messages related to opening the link source. For example, the container displays a message if the link source cannot be accessed.

Automatic and Manual Updating

When the user creates an OLE link, by default it is an automatic link; that is, whenever the source data changes, the link's visual representation changes without requiring any additional information from the user. Therefore, do not display an "Update Automatic Links Now?" message box. If the update takes a significant time to complete, you can display a message box indicating the progress of the update.

If users wish to exercise control over when links are updated, they can set the linked object's update property to manual. Doing so requires that the user choose an explicit command to update the link representation. The link can also be updated as a part of the link container's "update fields" or "recalc" action or other command that implies updating the presentation in the container's window.

Petitioner Exhibit 1031, Page 340

Operations and Links

The operations available for an OLE linked object are supplied by its container and its source. When the user chooses a command supplied by its container, the container application handles the operation. For example, the container processes commands such as Cut, Copy, or Properties. When the user chooses a command supplied (registered) by its source, the operation is conceptually passed back to the linked source object for processing. In this sense, activating an OLE linked object activates its source object.

In certain cases, the linked object exhibits the result of an operation; in other cases, the linked source object can be brought to the top of the Z order to handle the operation. For example, carrying out commands, such as Play or Rewind, on a link to a sound recording appear to operate on the linked object in place. However, if the user chooses a command to alter the link's representation of its source's content (such as Edit or Open), the link source is exposed and responds to the operation instead of the linked object itself.

A link can play a sound in place, but cannot support editing in place. For a link source to properly respond to editing operations, fully activate the source object (with all of its containing objects and its container). For example, when the user double-clicks a linked object whose default operation is Edit, the source (or its container) opens, displaying the linked source object ready for editing. If the source is already open, the window displaying the source becomes active. This follows the standard convention for activating a window already open; that is, the window comes to the top of the Z order. You can adjust the view in the window, scrolling or changing focus within the window, as necessary, to present the source object for easy user interaction. The linked source window and linked object window operate and close independently of each other.

If a link source is contained within a read-only document, display a message box advising the user that edits cannot be saved to the source file.

Petitioner Exhibit 1031, Page 341

Types and Links

An OLE linked object includes a cached copy of its source's type at the time of the last update. When the type of a linked source object changes, all links derived from that source object contain the old type and operations until either an update occurs or the linked source is activated. Because out-of-date links can potentially display obsolete operations to the user, a mismatch can occur. When the user chooses a command for an OLE linked object, the linked object compares the cached type with the current type of the linked source. If they are the same, the OLE linked object forwards the operation on to the source. If they are different, the linked object informs its container. In response, the container can either:

- Carry out the new type's operation, if the operation issued from the old link is syntactically identical to one of the operations registered for the source's new type.

- Display a message box, if the issued operation is no longer supported by the link source's new type (as shown in Figure 11.44).

In either case, the OLE linked object adopts the source's new type, and subsequently the container displays the new type's operations in the OLE linked object's menu.

Link Management

An OLE linked object includes properties such as: the name of its source, its source's type, and the link's updating basis, which is either automatic or manual. An OLE linked object also has a set of commands related to these properties. It is the responsibility of the container of the linked object to provide the user access to these commands and properties. To support this, an OLE container provides a property sheet for all of its OLE objects. You can optionally also include a Links dialog box for viewing and altering the properties of several links simultaneously.

Petitioner Exhibit 1031, Page 342

Accessing Properties of OLE Objects

Like other types of objects, OLE embedded and linked objects have properties. The container of an OLE object is responsible for providing the user interface for access to the object's properties. The following sections describe how to provide user access to the properties of OLE objects.

The Properties Command

Design OLE containers to include a Properties command and property sheets for any OLE objects it contains. If the container application already includes a Properties command for its own native data, you can also use it to support selected OLE embedded or linked objects. Otherwise, add the command to the drop-down and pop-up menu you provide for accessing the other commands for the object, preceded by a menu separator, as shown in Figure 11.29.

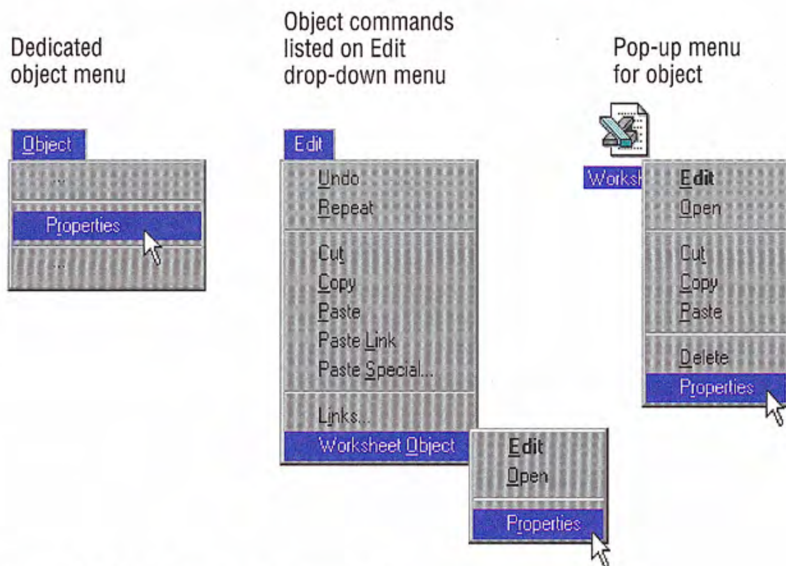


Figure 11.29 The Properties command

Petitioner Exhibit 1031, Page 343

When the user chooses the Properties command, the container displays a property sheet containing all the salient properties and values, organized by category, for the selected object. Figure 11.30 shows examples property sheet pages for an OLE object.

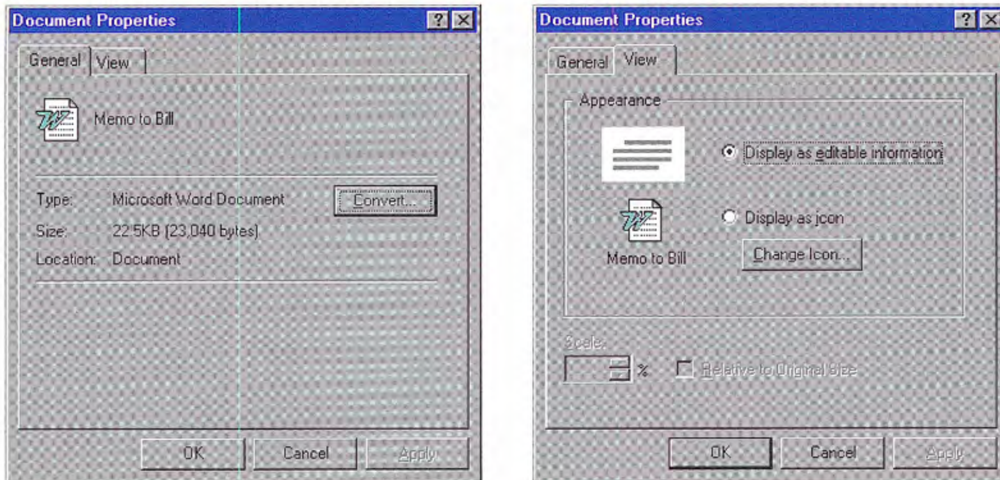


Figure 11.30 OLE embedded object property sheet

Follow the format the system uses for property sheets and the conventions outlined in this guide. Use the short type name in the title bar; for an OLE linked object, precede the name with the word "Linked," as in "Linked Worksheet." Include a General property page displaying the icon, name, type, size, and location of the object. Also include a Convert command button to provide access to the type conversion dialog box. On a View page, display properties associated with the view and presentation of the OLE object within the container. These include scaling or position properties and whether to display the object in its content presentation or as an icon. The Display As Icon field includes a Change Icon command button that allows the user to customize the icon presentation of the object. The Change Icon dialog box is shown in Figure 11.31.

Petitioner Exhibit 1031, Page 344

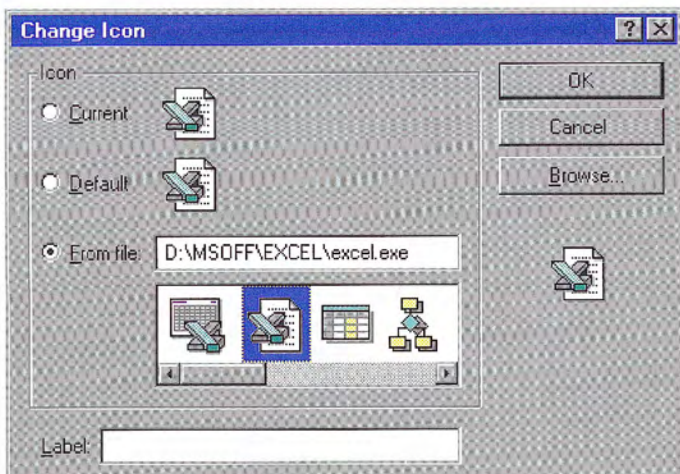


Figure 11.31 The Change Icon dialog box

For OLE linked objects, also include a Link page in its property sheet containing the essential link parameters and commands, as shown in Figure 11.32.

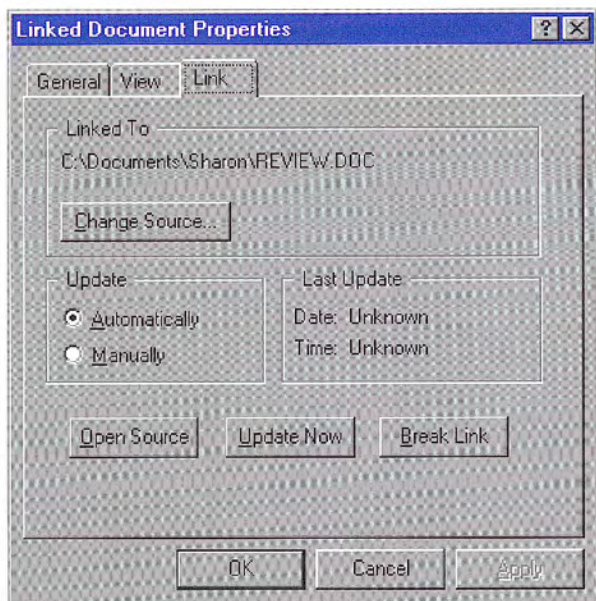


Figure 11.32 The Link page for the property sheet of an OLE linked object

For the typical OLE link, include the source name, the Update setting (automatic or manual), the Last Update timestamp, and command buttons that provide the following link operations:

- Break Link effectively disconnects the selected link.
- Update Now forces the selected link to connect to its sources and retrieve the latest information.
- Open Source opens the link source for the selected link.
- Change Source invokes a dialog box similar to the common Open dialog box to allow the user to respecify the link source.

The Links Command

In addition to property sheets, OLE containers can optionally include a Links command that provides access to a dialog box for displaying and managing multiple links. Figure 11.33 shows the Links dialog box. The list box in the dialog box displays the links in the container. Each line in the list contains the link source's name, the link source's object type (short type name), and whether the link updates automatically or manually. If a link source cannot be found, "Unavailable" appears in the update status column.

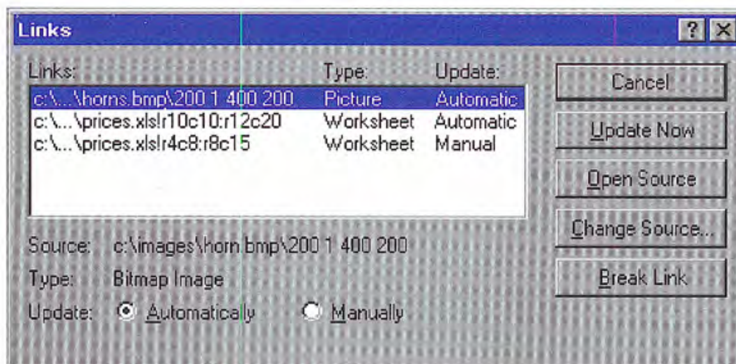


Figure 11.33 The Links dialog box

If the user chooses the Links command when the current selection includes a linked object (or objects), display that link (or links) as selected in the Links dialog box and scroll the list to display the first selected link at the top of the list box.

Petitioner Exhibit 1031, Page 346

Allow 15 characters for the short type name field, and enough space for Automatic and Manual to appear completely. As the user selects each link in the list, its type, name, and updating basis appear in their entirety at the bottom of the dialog box. The dialog box also includes link management command buttons included in the Link page of OLE linked object property sheets: Break Link, Update Now, Open Source, and Change Source.

Define the Open Source button to be the default command button when the input focus is within the list of links. Support double-clicking an item in the list as a shortcut for opening that link source.

Clicking the Change Source button displays a version of the Open dialog box that allows the user to change the source of a link by selecting a file or typing a filename. If the user enters a source name that does not exist and chooses the default button, a message box is displayed with the following message, as shown in Figure 11.34.

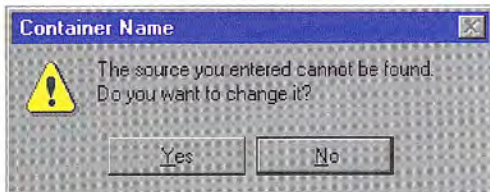


Figure 11.34 A message box for an invalid source

If the user chooses Yes, display the Change Source dialog box to correct the string. If the user chooses No, store the unparsed display name of the link source until the user links successfully to a newly created object that satisfies the dangling reference. The container application can also choose to allow the user to connect only to valid links.

If the user changes a link source or its directory, and other linked objects in the same container are connected to the same original link source, the container may offer the user the option to make the changes for the other references. To support this option, use the message box, as shown in Figure 11.35.

Petitioner Exhibit 1031, Page 347

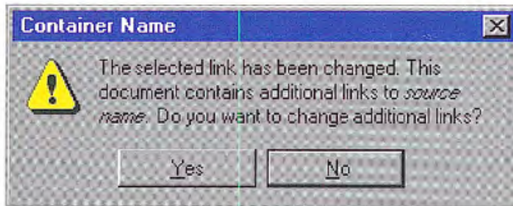


Figure 11.35 Changing additional links with the same source

Converting Types

Users may want to convert an object's type, so they can edit the object with a different application. To support the user's converting an OLE object from its current type to another registered type, provide a Convert dialog box, as shown in Figure 11.36. The user accesses the Convert dialog box by including a Convert button beside the Type field in an object's property sheet.

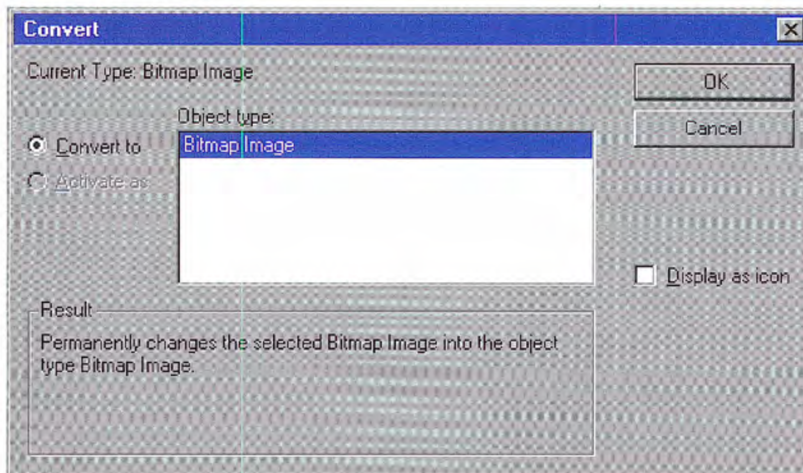
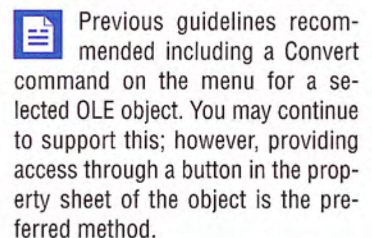


Figure 11.36 The Convert dialog box

This dialog box displays the current type of the object and a list box with all possible conversions. This list is composed of all types registered as capable of reading the selected object's format, but this does not necessarily guarantee the possibility of reverse conversion. If the user selects a new type from the list and chooses the OK button, the selected object is converted immediately to the new type. If the object is open, the container closes it before beginning the conversion.


Previous guidelines recommended including a Convert command on the menu for a selected OLE object. You may continue to support this; however, providing access through a button in the property sheet of the object is the preferred method.

Petitioner Exhibit 1031, Page 348

Make sure the application that supplies the conversion does so with minimal impact in the user interface. That is, avoid displaying the application's primary window, but do provide a progress indicator message box with appropriate controls so that the user can monitor or interrupt the conversion process.

If the conversion of the type could result in any lost data or information, the application you use to support the type conversion should display a warning message box indicating that data will be lost and request confirmation by the user before continuing. Make the message as specific as possible about the nature of the information that might be lost; for example, "Text properties will not be preserved." If the conversion will result in no data loss, the warning message is not necessary.

In addition to converting a type, the Convert dialog box offers the user the option to change the type association for the object by choosing the Activate As option. When the user chooses this option, selects a type from the list, and chooses the OK button, the object's type is now treated as the new type. This differs from type conversion in that the object's type remains the same, but its activation command is now handled by a different application. It also differs in that converting a type only affects the object that is converted. Changing the activation association of a single object of the type, changes it for all OLE embedded objects of that type. For example, converting a rich-text format document to a text document only affects the converted document. However, if the user chooses the Activate As option to change the association for the rich-text format object so they will be activated as a text object (that is, by the same application registered for editing text objects), all OLE embedded rich-text format objects will be also be activated this way.

At the bottom of the Convert dialog box, text describes the outcome of the choices the user selects. Table 11.3 outlines the syntax of the descriptive text to use within the Convert dialog box.

Petitioner Exhibit 1031, Page 349

Table 11.3 Descriptive Text for Convert Dialog Box

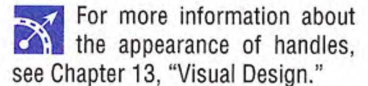
Function	Resulting text
Convert the selected object's type to a new type.	"Permanently changes the selected *Existing Type Name* object to a *New Type Name* object."
Convert the selected object's type to a new type and display the object as an icon.	"Permanently changes the selected *Existing Type Name* object to a *New Type Name* object. The object will be displayed as an icon."
No type change (the selected type is the same as its existing type).	"The *New Type Name* you selected is the same as the type of the selected object, so its type will not be converted."
Change the activation association for the selected object's type.	"Every *Existing Type Name* object will be activated as a *New Type Name* object, but not be converted to the new type."
Change the activation association for the selected object's type and display the object as an icon.	"Every *Existing Type Name* object will be activated as a *New Type Name* object, but converted to the new type. The selected object will be displayed as an icon.

Disable the Convert option for a linked object because conversion for a link must occur on the link source. Also disable Activate As option if no types are registered for alternative activation. If the user can neither convert nor change the activation association, disable the Convert command that displays this dialog box.

Petitioner Exhibit 1031, Page 350

Using Handles

A container displays handles for an OLE embedded or linked object when the object is selected individually. When an object is selected and not active, only the scaling of the object (its cached metafile) can be supported. If a container uses handles for indicating selection but does not support scaling of the image, use the hollow form of handles.



For more information about the appearance of handles, see Chapter 13, "Visual Design."

When an OLE embedded object is activated for OLE visual editing, it displays its own handles. Display the handles within the active hatched pattern, as shown in Figure 11.37.

	A	B	C	D
1	U.S. Compact Disc vs. LP Sales ($)			
2		1983	1987	1991
3	CDs	6,345K	18,652K	32,657K
4	LPs	31,538K	26,571K	17,429K
5	Total	37,883K	45,223K	50,086K

Figure 11.37 An active OLE embedded object with handles

The interpretation of dragging the handle is defined by the OLE embedded object's application. The recommended operation is cropping, where you expose more or less of the OLE embedded object's content and adjust the viewport. If cropping is inappropriate or unsupportable, use an operation that better fits the context of the object or simply support scaling of the object. If no operation is meaningful, but handles are required to indicate selection while activated, use the hollow handle appearance.

Petitioner Exhibit 1031, Page 351

Undo Operations for Active and Open Objects

Because different objects (that is, different underlying applications) take control of a window during OLE visual editing, managing commands like Undo or Redo present a question: how are the actions performed within an edited OLE embedded object reconciled with actions performed on the native data of the container with the Undo command? The recommended undo model is a single undo stack per open window — that is, all actions that can be reversed, whether generated by OLE embedded objects or their container, accumulate on the same undo state sequence. Therefore, choosing Undo from either the container's menus or an active object's menus reverses the last undoable action performed in that open window, regardless of whether it occurred inside or outside the OLE embedded object. If the container has the focus and the last action in the window occurred within an OLE embedded object, when the user chooses Undo, activate the embedded object, reverse the action, and leave the embedded object active.

The same rule applies to open objects — that is, objects that have been opened into their own window. Because each open window manages a single stack of undoable states, actions performed in an open object are local to that object's window and consequently must be undone from there; actions performed in the open object (even if they create updates in the container) do not contribute to the undo state of the container.

Carrying out a registered command of a selected, but inactive, object (or using a shortcut equivalent) is not a reversible action; therefore, it does not add to a container's undo stack. For example, if the user opens an object, this action cannot be undone from its container. The resulting window must be closed directly to remove it.

Figure 11.38 shows two windows: container Window A, which has an active OLE embedded object, and an open embedded object in Window B. Between the two windows, nine actions have been performed in the order and at the location indicated by the numbers. The resulting undo stacks are displayed beneath the windows.

Petitioner Exhibit 1031, Page 352

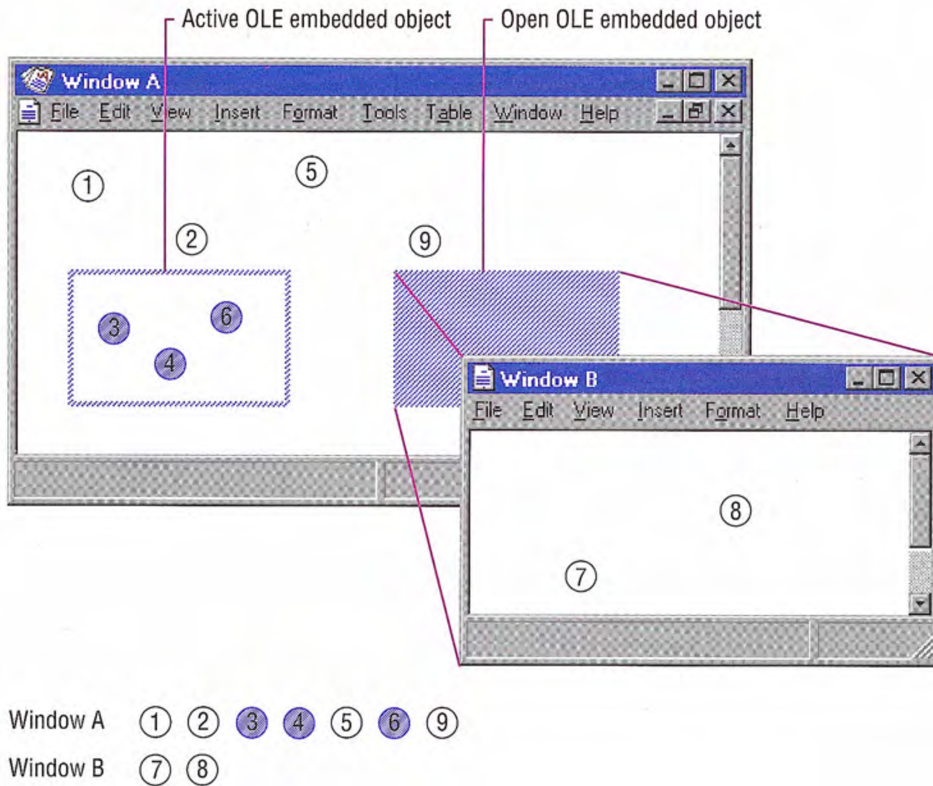


Figure 11.38 Undo stacks for active and open OLE embedded objects

The sequence of undo states shown in Figure 11.38 does not necessarily imply an *n*-level undo. It is merely a timeline of actions that can be undone at 0, 1, or more levels, depending on what the container-object cooperation supports.

The active object actions and native data actions within Window A have been serialized into the same stack, while the actions in Window B have accumulated onto its own separate stack.

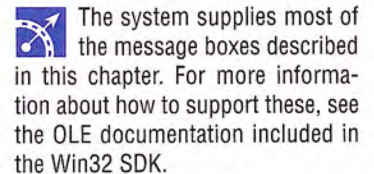
The actions discussed so far apply to a single window, not to actions that span multiple windows, such as OLE drag and drop. For a single action that spans multiple windows, the ideal design allows the user to undo the action from the last window involved. This is because, in

Petitioner Exhibit 1031, Page 353

most cases, the user focuses on that window when desiring to reverse the action. So if the user drags and drops an item from Window A into Window B, the action appends to Window B's undo thread, and undoing it undoes the entire OLE drag and drop operation. Unfortunately, the system does not support multiple window undo coordination. So for a multiple window action, create independent undo actions in each window involved in the action.

Displaying Messages

This section includes recommendations about other messages to display for OLE interaction using message boxes and status line messages. Use the following messages in addition to those described earlier in this chapter.


The system supplies most of the message boxes described in this chapter. For more information about how to support these, see the OLE documentation included in the Win32 SDK.

Object Application Messages

Display the following messages to notify the user about situations where an OLE object's application is not accessible.

Object's Application Cannot Run Standalone

Some OLE objects are designed to be used only as components within a container and have no value in being opened directly. If the user attempts to open or run an OLE object's application that cannot run as a standalone application, display the message box shown in Figure 11.39.

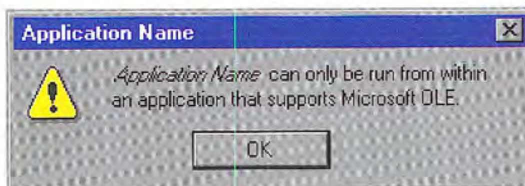


Figure 11.39 Object's application cannot be run standalone message

Petitioner Exhibit 1031, Page 354

Object's Application Busy

An object's application can be running, but busy for several reasons. For example, it can be busy printing, waiting for user input to a modal message box, or the application has stopped responding to the system. If the object's application is busy, display the message box shown in Figure 11.40.

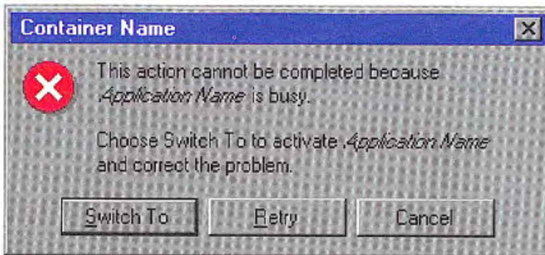


Figure 11.40 Object's application is busy message

Object's Application Unavailable

If the user attempts to activate an object and the container cannot locate the requested object's application, for example, because the object's type is not registered or because the network server where the application resides is unavailable, display the message box shown in Figure 11.41.

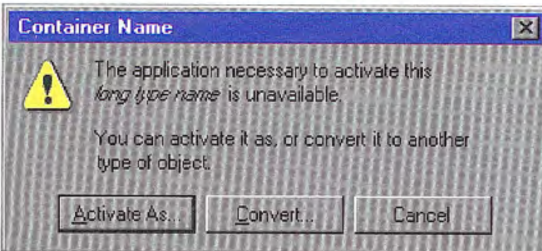


Figure 11.41 Object's application is unavailable message

Choosing the Activate As button displays the Convert dialog box preset with the Activate As option and a list of current types the user can use to associate with activating the object. Choosing the Convert button displays the Convert dialog box with the Convert option set, and the list of types the user can choose to change the type of the object. Ideally, an application that registers the type should be able to read and write that format without any loss of information. If it

cannot preserve the information of the original type, the application handling the type emulation displays a message box warning the user about what information it cannot preserve and optionally allows the user to convert the object's type.

If a container supports inside-out activation for an object, display this message when the user tries to interact with that object, not when its container is opened. This avoids the display of the message to the user who only intends to view the content.

OLE Linked Object Messages

Display the following messages to notify the user about situations related to interaction with OLE linked objects.

Link Source Files Unavailable

When a container requests an update for its OLE linked objects, either because the user chooses an explicit Update command or as the result of another action such as a Recalc operation that forces an update, if the link source files for some OLE links are unavailable to provide the update, display the message box shown in Figure 11.42.

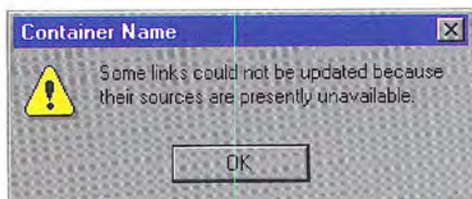


Figure 11.42 Link source files are unavailable message

When the user chooses the OK button, close the dialog box without updating the links.

Optionally, if you want to support the user changing the source, you supply your own message box that also includes a Properties button, a Links button, or both in the message box. Choosing the Properties button displays the property sheet for the link (see Figure 11.32) with "Unavailable" in the Update field. The user can then use the

Petitioner Exhibit 1031, Page 356

Change Source button to search for the file or choose other commands related to the link. When the user chooses this Links button, display your Links dialog box, following the same conventions as for the property sheet.

Similarly, if the user issues a command to an OLE linked object with an unavailable source, display the warning message shown in Figure 11.43.

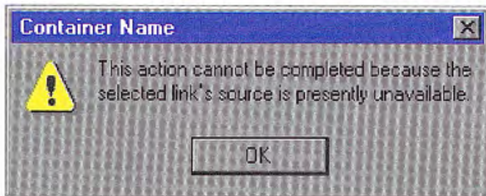


Figure 11.43 Selected link source is unavailable message

You can also supply your own message if you want to provide a Properties or Links button that enables the user to change the source. Display the OLE linked object's update status as "Unavailable."

Link Source Type Changed

If a link source's type has changed, but it is not yet reflected for an OLE linked object, and the user chooses a command that does not support the new type, display the message box shown in Figure 11.44.

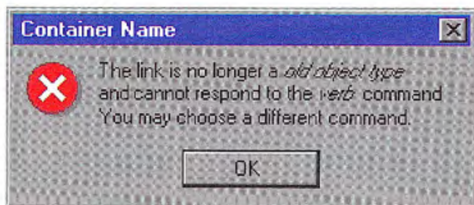


Figure 11.44 Link source's type has changed message

Petitioner Exhibit 1031, Page 357

Link Updating

While links are updating, display the progress indicator message box shown in Figure 11.45. The Stop button interrupts the update process and prevents any updating of additional links.

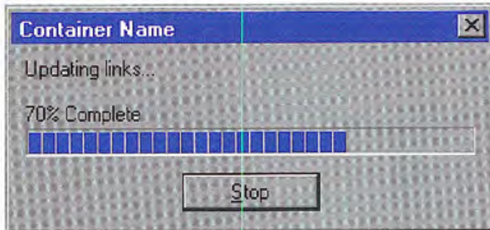


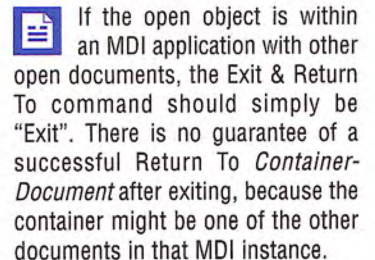
Figure 11.45 Progress indicator while links update message

Status Line Messages

Table 11.4 lists suggested status line messages for commands on the primary container menu (commonly the File menu) of an opened object.

Table 11.4 Primary Container Menu Status Line Messages

Command	Status line message
Update *Container-Document*	Updates the appearance of this *Type Name* in *Container-Document.*
Close & Return to *Container-Document*	Closes *Object Name* and returns to *Container-Document.*
Save Copy As	Saves a copy of *Type Name* in a separate file.
Exit & Return to *Container-Document*	Exits *Object Application* and returns to *Container-Document.*



If the open object is within an MDI application with other open documents, the Exit & Return To command should simply be "Exit". There is no guarantee of a successful Return To *Container-Document* after exiting, because the container might be one of the other documents in that MDI instance.

Table 11.5 lists the recommended status line messages for the Edit menu of containers of OLE embedded and linked objects.

Petitioner Exhibit 1031, Page 358

Table 11.5 Edit Menu Status Line Messages

Command	Status line message
Paste *Object Name*[1]	Inserts the content of the Clipboard as *Object Name*.
Paste Special	Inserts the content of the Clipboard with format options.
Paste Link [to *Object Name*[1]]	Inserts a link to *Object Name*.
Paste Shortcut [to *Object Name*[1]]	Inserts a shortcut icon to *Object Name*.
Insert Object	Inserts a new object.
[Linked] *Object Name*[1] [Object] ▶	Applies the following commands to *Object Name*.
[Linked] *Object Name*[1] [Object] ▶ *Command*	Varies based on command.
[Linked] *Object Name*[1] [Object] ▶ Properties	Allows properties of *Object Name* to be viewed or modified.
Links	Allows links to be viewed, updated, opened, or removed.

[1]*Object Name* may be either the object's short type name or its filename.

Table 11.6 lists other related status messages.

Table 11.6 Other Status Line Messages

Command	Status line message
Show Objects	Displays the borders around objects (toggle).
Select Object (when the user selects an object)	Double-click or press ENTER to *Default Command Object Name*.

The default command stored in the registry contains an ampersand character (&) and the access key indicator; these must be stripped out before the verb is displayed on the status line.

Petitioner Exhibit 1031, Page 359

CHAPTER

User Assistance

Online user assistance is an important part of a product's design and can be supported in a variety of ways, from automatic display of information based on context to commands that require explicit user selection. Its content can be composed of contextual, procedural, explanatory, reference, or tutorial information. But user assistance should always be simple, efficient, and relevant so that a user can obtain it without becoming lost in the interface. This chapter provides a description of the system support to create common online forms of user assistance support and guidelines for implementation. For more information about authoring Help files, see the documentation included in the Microsoft Win32 Software Development Kit (SDK).

Contextual User Assistance

A contextual form of user assistance provides information about a particular object and its context. It answers questions such as "What is this?" and "Why would I use it?" This section covers some of the basic ways to support contextual user assistance in your application.

Context-Sensitive Help

The What's This? command supports a user obtaining contextual information about any object on the screen, including controls in property sheets and dialog boxes. This form of contextual user

assistance is referred to as *context-sensitive Help*. As shown in Figure 12.1, you can support user access to this command by including:

- A What's This? command from the Help drop-down menu of a primary window.
- A What's This? button on a toolbar.
- A What's This? button on the title bar of a secondary window.
- A What's This? command included on the pop-up menu for the specific object.

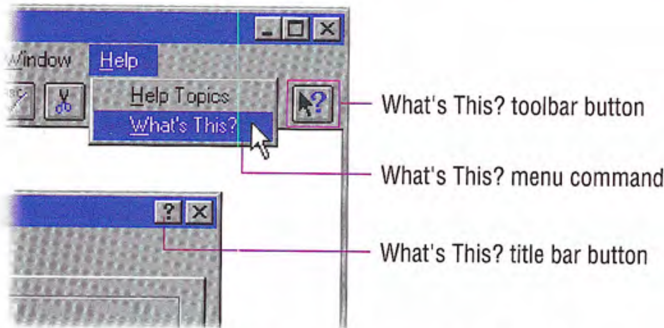


Figure 12.1 Different methods of accessing What's This?

Design your application so that when the user chooses the What's This? command from the Help drop-down menu or clicks a What's This? button, the system is set to a temporary mode. Change the pointer's shape to reflect this mode change, as shown in Figure 12.2. The SHIFT+F1 combination is the shortcut key for this mode.

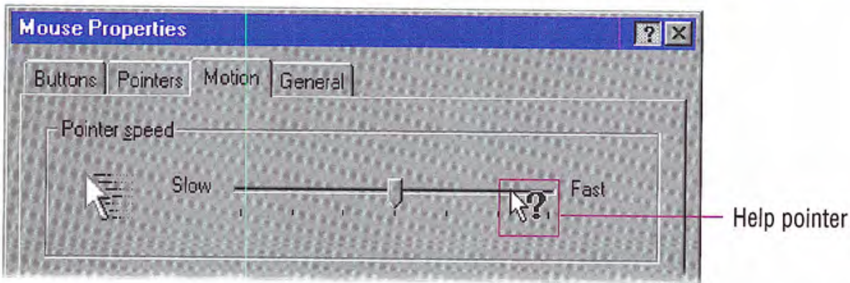


Figure 12.2 A context-sensitive Help pointer

Petitioner Exhibit 1031, Page 362

Display the context-sensitive Help pointer only over the window that provides context-sensitive Help; that is, only over the active window from which the What's This? command was chosen.

In this mode, when the user clicks an object with mouse button 1 (for pens, tapping), display a context-sensitive Help pop-up window for that object. The context-sensitive Help window provides a brief explanation about the object and how to use it, as shown in Figure 12.3. Once the context-sensitive Help window is displayed, return the pointer and pointer operation to its usual state.

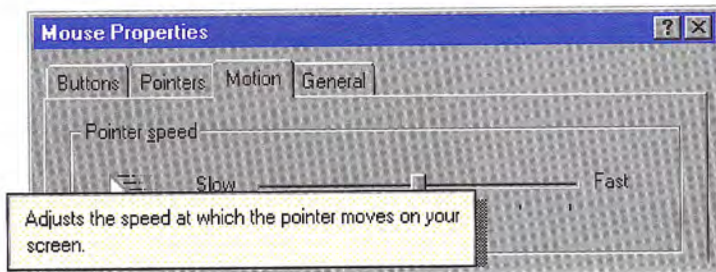


Figure 12.3 A pop-up window for context-sensitive Help

If the user presses a shortcut key that applies to a window that is in contextual Help mode, you can display a contextual Help pop-up window for the command associated with that shortcut key.

However, there are some exceptions to this interaction. First, if the user chooses a menu title, either in the menu bar or a cascading menu, maintain the mode until the user chooses a menu item and then display the context-sensitive Help window. Second, if the user clicks the item with mouse button 2 and the object supports a pop-up menu, maintain the mode until the user chooses a menu item or cancels the menu. If the object does not support a pop-up menu, the interaction should be the same as clicking it with mouse button 1. Finally, if the chosen object or location does not support context-sensitive Help or is otherwise an inappropriate target for context-sensitive Help, cancel the context-sensitive Help mode.

If the user chooses the What's This? command a second time, clicks outside the window, or presses the ESC key, cancel the context-sensitive Help mode. Restore the pointer to its usual image and operation in that context.

Petitioner Exhibit 1031, Page 363

When the user chooses the What's This? command from a pop-up menu (as shown in Figure 12.4), the interaction is slightly different. Because the user has identified the object by clicking mouse button 2, there is no need for entering the context-sensitive Help mode. Instead, immediately display the context-sensitive Help pop-up window for that object.

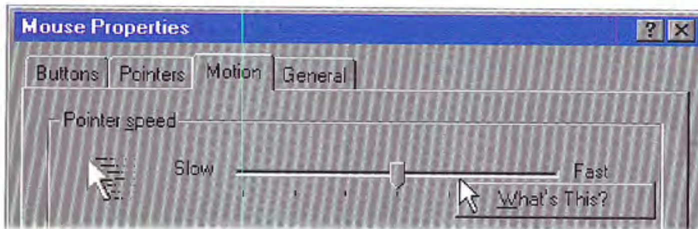


Figure 12.4 A pop-up menu for a control

The F1 key is the shortcut key for this form of interaction; that is, pressing F1 displays a context-sensitive Help window for the object that has the input focus.

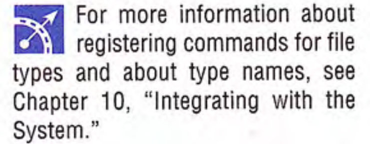
Guidelines for Writing Context-Sensitive Help

When authoring context-sensitive Help information, you are answering the question "What is this?" Indicate the action associated with the item. In English versions, begin the description with a verb; for example, "Adjusts the speed of your mouse," or "Provides a place for you to type in a name for your document." For command buttons, you may use an imperative form — for example, "Click this to close the window." When describing a function or object, use words that explain the function or object in common terms instead of technical terminology or jargon. For example, instead of "Undoes the last action," say "Reverses the last action."

In the explanation, you might want to include "why" information. You can also include "how to" information, but if the procedure requires multiple steps, consider supporting this information using task-oriented Help. Keep your information brief, but as complete as possible so that the Help window is easy and quick to read.

Petitioner Exhibit 1031, Page 364

As an option, you can provide context-sensitive Help information for your supported file types by registering a What's This? command for the type, as shown in Figure 12.5. This allows the user to choose the "What's This?" command from the file icon's pop-up menu to get information about an icon representing that type. When defining this Help information, include the type name and a brief description of its function, using the previously described guidelines.



For more information about registering commands for file types and about type names, see Chapter 10, "Integrating with the System."

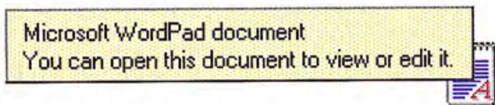


Figure 12.5 Context-sensitive Help information for an icon

Tooltips

Another form of contextual user assistance are tooltips. *Tooltips* are small pop-up windows that display the name of a control when the control has no text label. The most common use of tooltips is for toolbar buttons that have graphic labels, as shown in Figure 12.6, but they can be used for any control.

Figure 12.6 A tooltip for a toolbar button

Display a tooltip after the pointer, or pointing device, remains over the button for a short period of time. The tooltip remains displayed until the user presses the button or moves off of the control, or after another time-out. If the user moves the pointer directly to another control supporting a tooltip, ignore the time-out and display the new tooltip immediately, replacing the former one.

If you use the standard toolbar control, the system automatically provides support for tootips. It also includes a tooltip control that can be used in other contexts. If you create your own tooltip controls, make them consistent with the system-supplied controls.

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

Status Bar Messages

You can also use a status bar to provide contextual user assistance. However, if you support the user's choice of displaying a status bar, avoid using it for displaying information or access to functions that are essential to basic operation and not provided elsewhere in the application's interface. In addition, because the status bar's location may not be near the user area of activity, the user may not always notice a status bar message. As a result, it is best to consider status bar messages as a secondary or supplemental form of user assistance.

In addition to displaying state information about the context of the activity in the window, you can display descriptive messages about menu and toolbar buttons, as shown in Figure 12.7. Like tooltips, the window typically must be active to support these messages. When the user moves the pointer over a toolbar button or presses the mouse button on a menu or button, display a short message the describing use of the associated command.

Petitioner Exhibit 1031, Page 366

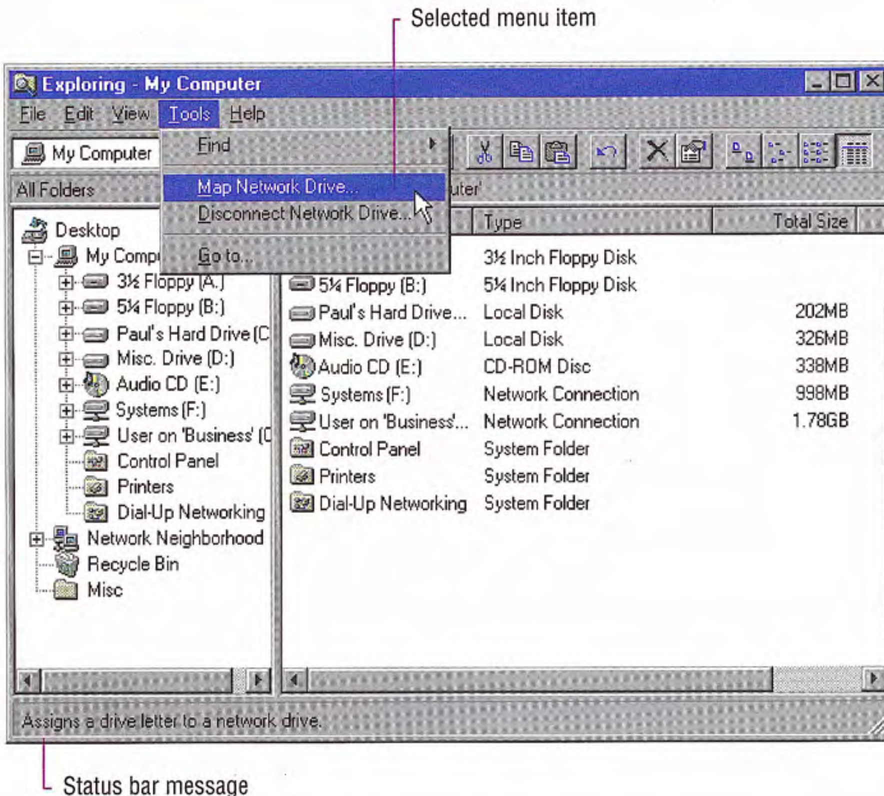


Figure 12.7 A status bar message for a selected menu command

A status bar message can include a progress indicator control or other forms of feedback about an ongoing process, such as printing or saving a file, that the user initiated in the window. Although you can display progress information in a message box, you may want to use the status bar for background processes so that the window's interface is not obscured by the message box.

Guidelines for Writing Status Bar Messages

When writing status bar messages, begin the text with a verb in the present tense and use familiar terms—avoiding jargon. For example, say "Cuts the selection and puts it on the Clipboard." Try to be as brief as possible so the text can be easily read, but avoid truncation.

Be constructive, not just descriptive, informing the user about the purpose of the command. When describing a command with a specific function, use words specific to the command. If the scope of the command has multiple functions, try to summarize. For example, say "Contains commands for editing and formatting your document."

When defining messages for your menu and toolbar buttons, don't forget their unavailable, or disabled, state. Provide an appropriate message to explain why the item is not currently available. For example, when the user selects a disabled Cut command you could display "This command is not available because no text is selected."

The Help Command Button

You can also provide contextual Help for a property sheet, dialog box, or message box by including a Help button in that window, as shown in Figure 12.8. When the user chooses the Help command button, display the Help information in a Help secondary window, rather than a context-sensitive Help pop-up window.

Figure 12.8 A Help button in a secondary window

The user assistance provided by a Help command button differs from the "What's This?" form of Help. Command button Help should provide an overview, summary assistance, or explanatory information for that window. For example, for a message box, it can provide more information about causes and remedies for the reason the message was displayed. Consider the Help command an optional, secondary form of contextual user assistance, not a substitute for context-sensitive, "What's This?" Help. Don't use it as a substitute for clear, understandable designs for your secondary windows.

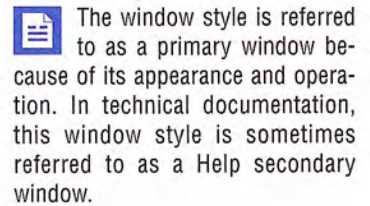
Petitioner Exhibit 1031, Page 368

Task-Oriented Help

Task-oriented help provides the steps for carrying out a task. It can involve a number of procedures. You present task-oriented Help in task Help topic windows.

Task Topic Windows

Task Help topic windows are displayed as primary windows. The user can size this window like any other primary window.



The window style is referred to as a primary window because of its appearance and operation. In technical documentation, this window style is sometimes referred to as a Help secondary window.

You provide primary access to task Help topics through the Help Topics browser, described later in this chapter. You can also include access to specific topics through other interfaces, such as navigation links placed in other Help topics.

Task topic windows include a set of command buttons at the top of the window (as shown in Figure 12.9) that provide the user access to the Help Topics browser, the previously selected topic, and other Help commands, such as copying and printing a topic. You can define which buttons appear by defining them in your Help files.

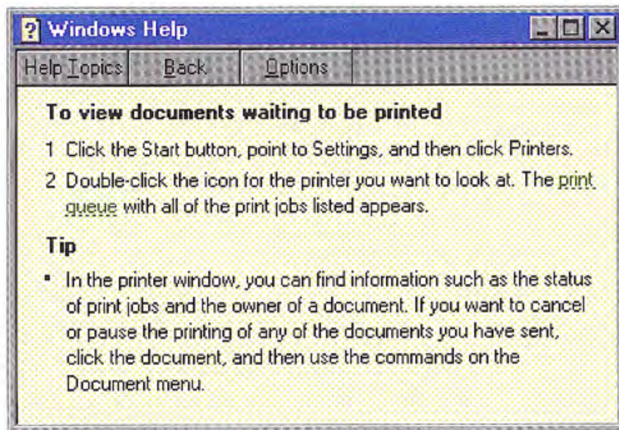


Figure 12.9 A window for a task Help topic

Petitioner Exhibit 1031, Page 369

Although you can define the size and location of a task Help topic window to the specific requirements of your application, it is best to size and position the window so as to cover the minimum of space, but make it large enough to allow the user to read the topic, preferably without having to scroll the window. This makes it easier for the novice user who may be unfamiliar with scrolling.

The title bar text of the Help topic identifies the context supplying the topic window. Consider naming the Help file to also match. Include a topic title as part of the body of the topic. Define the topic title to correspond to the entries you include in the Help Topics browser that provide direct access to the topic. This does not mean that you must use the same wording as those entries, but they should be similar enough to allow the user to recognize their relationship.

Like tooltips, the default interior color of a task topic window should use the system color setting for Help windows. This allows the user to more easily distinguish the Help topic from their other windows. However, for specialized topics, you can set the color of a task topic window.

Guidelines for Writing Task Help Topics

The buttons that appear at the top of a task Help topic window are defined by your Help file. At a minimum, you should provide a button that displays the Help Topics browser dialog box, a Back button to return the user to the previous topic, and buttons that provide access to other functions, such as Copy and Print.

To provide access to the Help Topics browser dialog box, include a Help Topics button. This displays the Help Topics browser window on the tabbed page that the user was viewing when the window was last displayed. Although this is the most common form of access to the Help Topics browser window, alternatively you can include buttons, such as Contents and Index, that correspond to the tabbed pages to provide the user with direct access to those pages when the dialog box is displayed.

Petitioner Exhibit 1031, Page 370

As with context-sensitive Help, when writing task Help information topics, make them complete, but brief. However, in task Help topics, focus on "how" information rather than "what" or "why." Task Help should assist the user in completing a task, not try to document everything there is to know about a topic. If there are multiple alternatives, pick one method — usually the simplest, most common method for a specific procedure. If you want to include information on alternative methods, provide access to them through other choices or commands.

If you keep the procedure to four or fewer steps, the user will not need to scroll the window. Avoid introductory, conceptual, or reference material in the procedure.

Also, take advantage of the context of a procedure. For example, if a property sheet includes a slider control that is labeled "Slow" at one end and "Fast" at the other, be concise. Say "Move the slider to adjust the speed" instead of "To increase the speed, move the slider to the right. To decrease the speed, move the slider to the left." If you refer to a control by its label, capitalize each word in the label, even though the label has only the first word capitalized. This helps distinguish the label from the rest of your text.

Optionally, you can include a Related Topics button in your topic window to provide access to other topics. When the user chooses this button, display the Topics Found dialog box (as shown in Figure 12.14).

Shortcut Buttons

Task Help topic windows can also include a shortcut or "do it" button that provides the user with a shortcut or automated form of performing a particular step, as shown in Figure 12.10. For example, use this to automatically open a particular dialog box, property sheet, or other object so that the user does not have to search for it.

Petitioner Exhibit 1031, Page 371

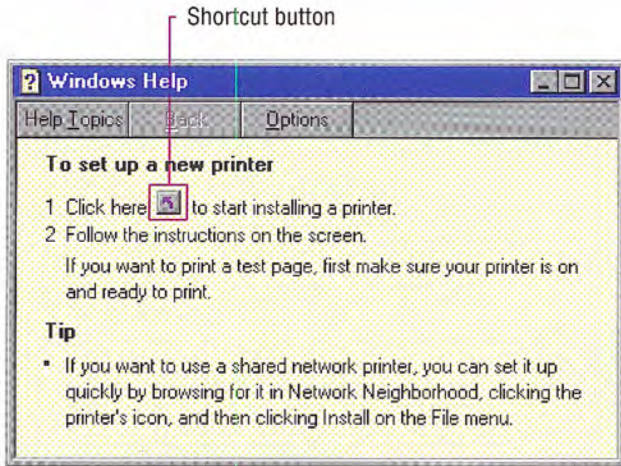


Figure 12.10 A task Help topic with a shortcut button

Shortcut buttons not only provide efficiency for the user, they also reduce the amount of information you may need to present and the user needs to read. However, you need not use the buttons as a substitute for doing the task or a specific step in the task; particularly if you want to support the user being able to accomplish the task without using Help. For common tasks, you may want a balance, including information that tells the user how to do the task, and shortcut buttons that make stepping through the task easier. For example, you might include text that reads "Click here to display the Display properties" and a shortcut button.

Reference Help

Reference Help is a form of Help information that serves more as online documentation. Use reference Help to document the features of a product or as a user's guide to a product. Often the use determines the balance of text and graphics used in the Help file. Reference-oriented documentation typically includes more text and follows a consistent presentation of information. User's guide documentation typically organizes information by specific tasks and may include more illustrations.

Petitioner Exhibit 1031, Page 372

The Reference Help Window

When designing reference Help, use a Help primary window style (sometimes called a "main" Help window), as shown in Figure 12.11, rather than the context-sensitive Help pop-up windows or task Help topic windows.

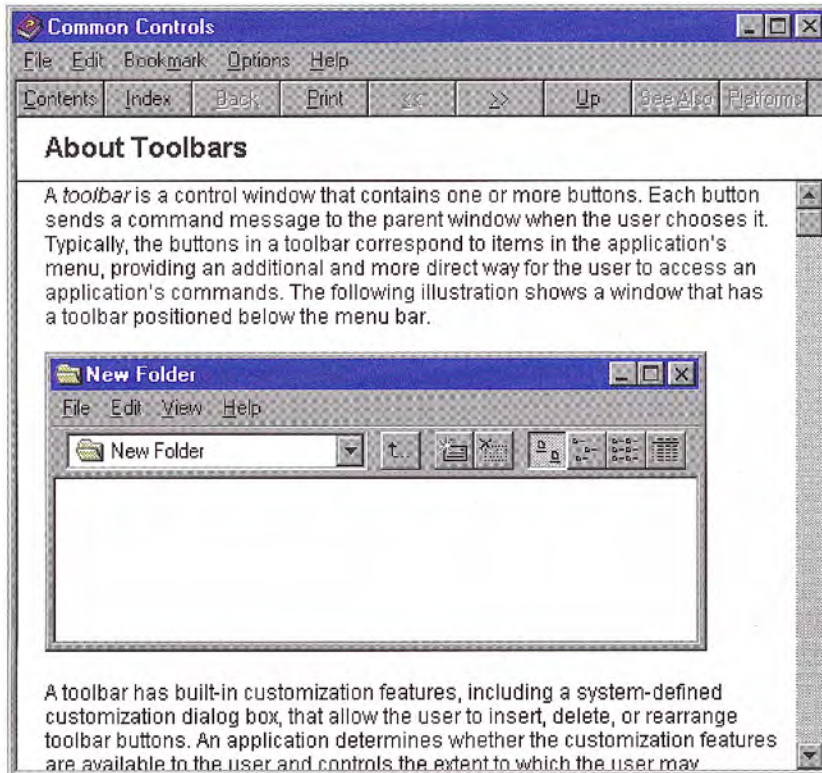


Figure 12.11 A reference Help window

You can provide access to reference Help in a variety of ways. The most common is as an explicit menu command in the Help drop-down menu, but you can also provide access using a toolbar button, or even as a specific file object (icon).

A reference Help window includes a menu bar, with File, Edit, Bookmark, Options, and Help entries and a toolbar with Contents, Index, Back, and Print buttons. The system provides these features by default for a "main" Help window. These features support user functions, such as opening a specific Help file (using the Help Topics

browser), copying and printing topics, creating annotations and bookmarks for specific topics, and setting the Help window's properties. You can add other buttons to this window to tailor your online documentation to fit your particular user needs.

Although the reference Help style can provide information similar to that provided in contextual Help and task Help, these forms of Help are not exclusive of each other. Often the combination of all these items provides the best solution for user assistance. They can also be supplemented with other forms of user assistance.

Guidelines for Writing Reference Help

Reference Help topics can include text, graphics, animations, video, and audio effects. Follow the guidelines included throughout this guide for recommendations on using these elements in the presentation of information. In addition, the system provides some special support for Help topics.

Adding Menus and Toolbar Buttons

You can author additional menus and buttons to appear in the reference Help window. However, you cannot remove existing menus.

Because reference Help files typically include related topics, include Previous Topic and Next Topic browse buttons in your Help window toolbar. Another common button you may want to include is a See Also button that either displays a pop-up window or the Topics Found dialog box (as shown in Figure 12.14) with the related topics. Other common buttons include Up for moving to the parent or overview topic and History to display a list of the topics the user has viewed so they can return directly to a particular topic.

Make toolbar buttons contextual to the topic the user is viewing. For example, if the current topic is the last in the browse chain, disable the Next Topic button. When deciding whether to disable or remove a button, follow the guidelines defined in this guide for menus.

Petitioner Exhibit 1031, Page 374

Topic Titles

Always provide a title for the current topic. The title identifies the topic and provides the user with a landmark within the Help system. The title should correspond to the entries you include in the Help Topics browser window. Use the title bar text of the window to identify the context and supplier of the topic. The Help filename should also match.

Nonscrolling Regions

If your topics are very long, you may want to include a nonscrollable region in your Help file. A nonscrolling region allows you to keep the topic title and other information visible when the user scrolls. A nonscrolling region appears with a line at its bottom edge to delineate it from the scrollable area. Display the scroll bar for the scrollable area of the topic so that its top appears below the nonscrolling region, not overlapped within that region.

Jumps

A jump is a button or interactive area that triggers an event when the user clicks on it. You can use a jump as a one-way navigation link from one topic to another, either within the same topic window, to another topic window, or a topic in another Help file.

You can also use jumps to display a pop-up window. As with pop-up windows for context-sensitive Help, use this form of interaction to support a definition or explanatory information about the word or object that the user clicks.

Jumps can also carry out particular commands. Shortcut buttons used in Help task topics are this form of a jump.

You need to provide visual indications to distinguish a jump from noninteractive areas of the window. You can do this by displaying a jump as a button, changing the pointer image to indicate an interactive element, formatting the item with some other visual distinction such as color or font, or a combination of these methods. The default system presentation for text jumps is green underlined text.

Petitioner Exhibit 1031, Page 375

The Help Topics Browser

The Help Topics browser dialog box provides user access to Help information. To open this window, include a Help Topics menu item on the Help drop-down menu. Alternatively, you can include menu commands that open the window to a particular tabbed page — for example, Contents, Index, and Find Topic.

In addition, provide a Help Topics button in the toolbar of a task or reference Help topic window. When the user chooses this button, display the Help Topics browser window with the last page the user accessed. If you prefer, provide Contents, Index, and Find Topic buttons for direct access to a specific page. For example, by default, reference Help windows include Contents and Index button access to the Help Topics browser.

The Help Topics Tabs

Opening the Help Topics window displays a set of tabbed pages. The default pages include Contents, Index, and Find tabs. You can author additional tabs.

The Contents Page

The Contents page displays the list of topics organized by category, as shown in Figure 12.12. A book icon represents a category or group of related topics, and a page icon represents an individual topic. You can nest topic levels, but avoid more than three levels, as this can make access cumbersome.

Petitioner Exhibit 1031, Page 376

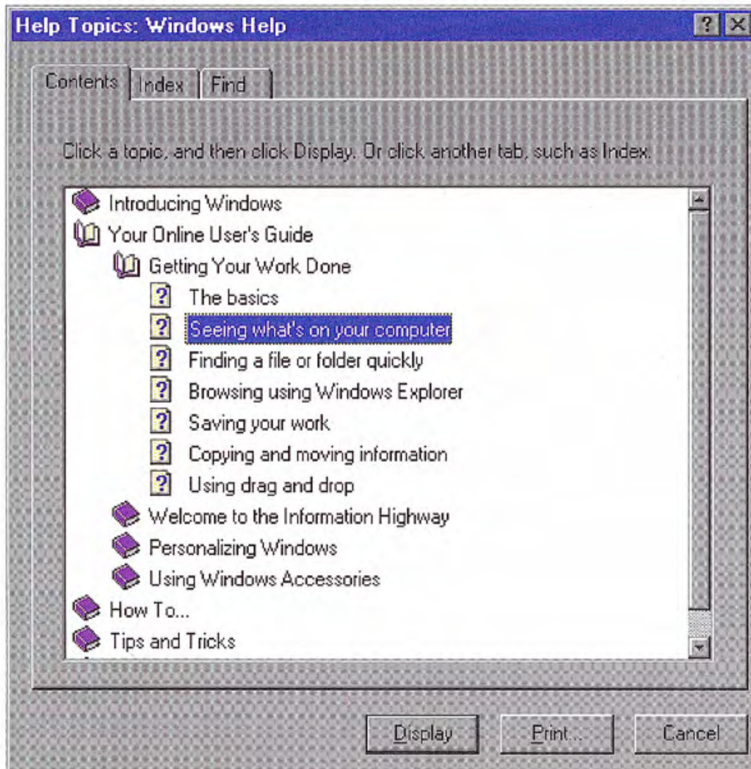


Figure 12.12 The Contents page of the Help topics browser

The buttons at the bottom of the page allow the user to open or close a "book" of topics and display a particular topic. The Print button prints either a "book" of topics or a specific topic depending on which the user selects. The outline also supports direct interaction, such as double-clicking, for opening the outline or a topic.

Guidelines for Writing Help Contents Entries

The entries listed on the Contents page are based on what you author in your Help files. Define them to allow the user to see the organizational relationship between topics. Make the topic titles you include for your software brief, but descriptive, and correspond to the actual topic titles.

Petitioner Exhibit 1031, Page 377

The Index Page

The Index page of the browser organizes the topics by keywords that you define for your topics, as shown in Figure 12.13.

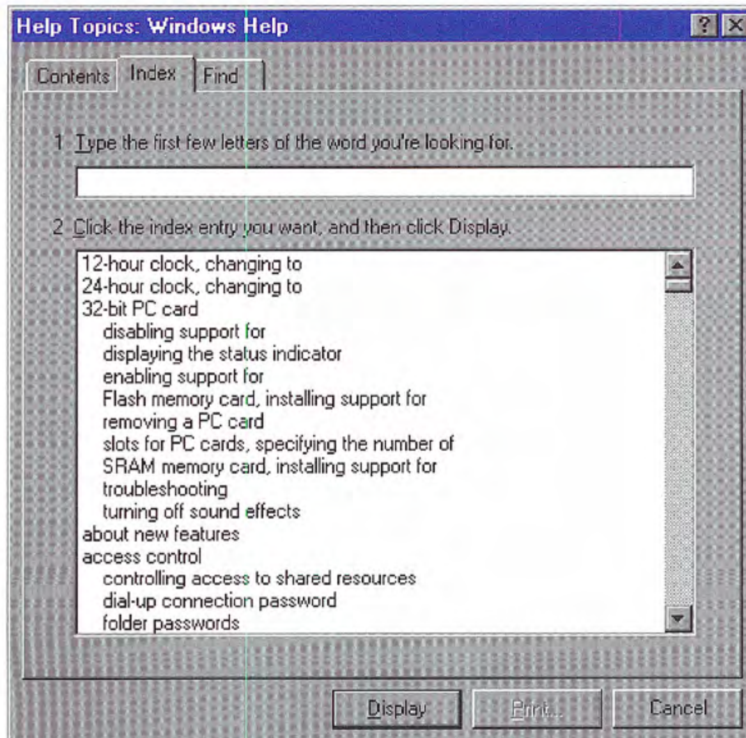


Figure 12.13 The Index page of the Help Topics browser

The user can enter a keyword or select one from the list. Choosing the Display button displays the topic associated with that keyword. If there are multiple topics that use the same keyword, then another secondary window is displayed that allows the user to choose from that set of topics, as shown in Figure 12.14. You can also use this dialog box to provide access to related topics by including a See Also button or Related Topics button in a topic window.

Petitioner Exhibit 1031, Page 378

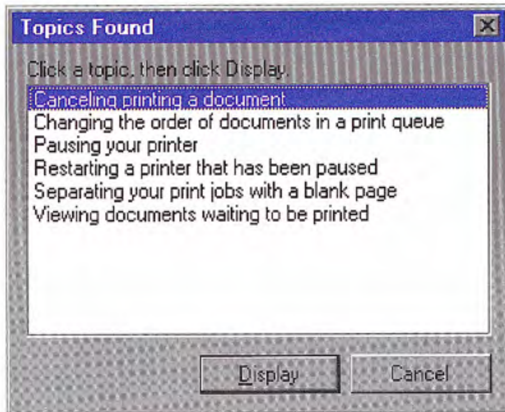


Figure 12.14 The Help topics window

Guidelines for Writing Help Index Keywords

Provide an effective keyword list to help users find the information they are looking for. When deciding what keywords to provide for your topics, consider the following categories:

- Words for a novice user.
- Words for an advanced user.
- Common synonyms of the words in the keyword list.
- Words that describe the topic generally.
- Words that describe the topic discretely.

The Find Page

The Find page, as shown in Figure 12.15, provides full-text search functionality that allows the user to search for any word or phrase in the Help file. This capability requires a full-text index file, which you can create when building the Help file, or which the user can create when using the Find page.

Petitioner Exhibit 1031, Page 379

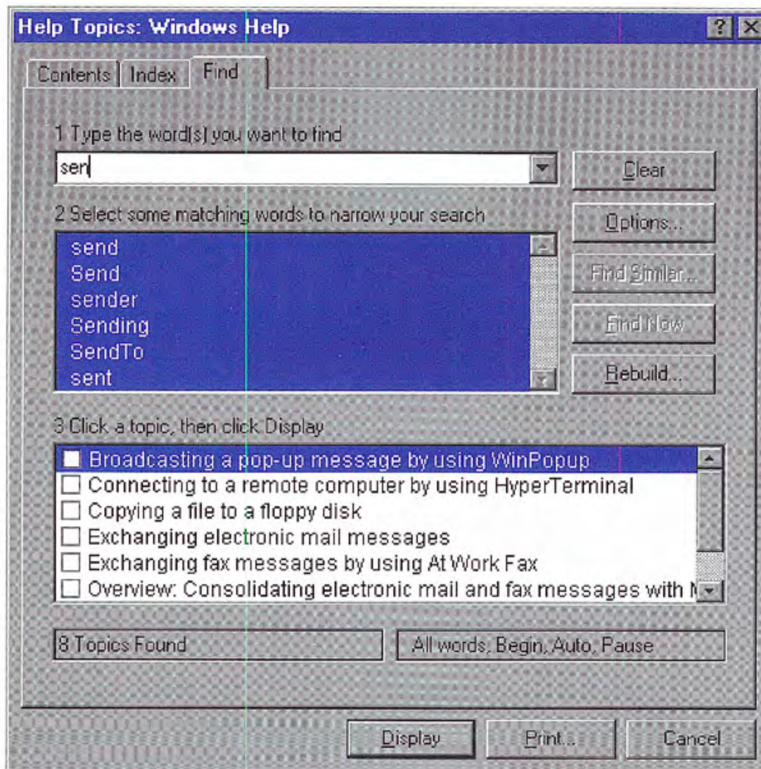
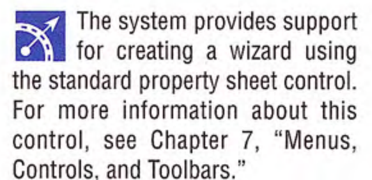


Figure 12.15 The Find page of the Help Topics browser

Wizards

A *wizard* is a special form of user assistance that automates a task through a dialog with the user. Wizards help the user accomplish tasks that can be complex and require experience. Wizards can automate almost any task, including creating new objects and formatting the presentation of a set of objects, such as a table or paragraph. They are especially useful for complex or infrequent tasks that the user may have difficulty learning or doing.



The system provides support for creating a wizard using the standard property sheet control. For more information about this control, see Chapter 7, "Menus, Controls, and Toolbars."

However, wizards are not well-suited to teach a user how to do something. Although wizards assist the user in accomplishing a task, they should be designed to hide many of the steps and much of the complexity of a given task. Similarly, wizards are not intended to be used for tutorials; wizards should operate on real data. For instructional user assistance, consider task Help or tutorial-style interfaces.

Petitioner Exhibit 1031, Page 380

Do not rely on wizards as a solution for ineffective designs; if the user relies on a wizard too much it may be an indication of an overly complicated interface, not good wizard design. In addition, consider using a wizard to supplement, rather than replace, the user's direct ability to perform a specific task. Unless the task is fairly simple or done infrequently, experienced users may find a wizard to be inefficient or not provide them with sufficient access to all functionality.

Wizards may not always appear as an explicit part of the Help interface. You can provide access to them in a variety of ways, including toolbar buttons or even specific icons, such as templates.

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

Guidelines for Designing Wizards

A wizard is a series of presentations or pages, displayed in a secondary window, that helps the user through a task. The pages include controls that you define to gather input from the user; that input is then used to complete the task for the user.

Optionally, you can define wizards as a series of secondary windows through which the user navigates. However, this can lead to increased modality and screen clutter, so using a single secondary window is recommended.

At the bottom of the window, include the following command buttons that allow the user to navigate through the wizard.

Command	Action
< Back	Returns to the previous page. (Remove or disable the button on the first page.)
Next >	Moves to the next page in the sequence, maintaining whatever settings the user provides in previous pages.
Finish	Applies user-supplied or default settings from all pages and completes the task.
Cancel	Discards any user-supplied settings, terminates the process, and closes the wizard window.

Petitioner Exhibit 1031, Page 381

Use the title bar text of the wizard window to clearly identify the purpose of the wizard. But, because wizards are secondary windows, they should not appear in the taskbar. You may optionally include a context-sensitive What's This? Help title bar button as well.

On the first page of a wizard, include a graphic in the left side of the window, as shown in Figure 12.16. The purpose of this graphic is to establish a reference point, or theme — such as a conceptual rendering, a snapshot of the area of the display that will be affected, or a preview of the result. On the top right portion of the wizard window, provide a short paragraph that welcomes the user to the wizard and explains what it does. You may also include controls for entering or editing input to be used by the wizard, if there is sufficient space. However, avoid offering too many choices or choices that may be difficult to distinguish or understand without context.

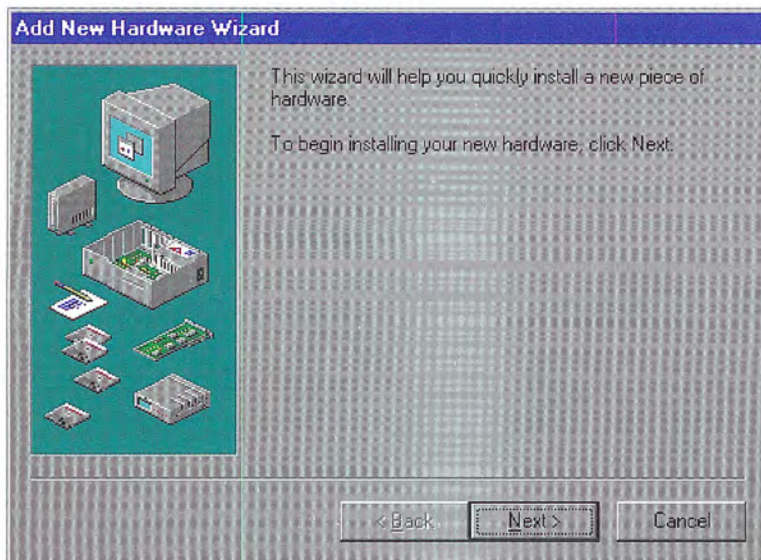


Figure 12.16 An introductory page of a wizard

On subsequent pages you can continue to include a graphic for consistency or, if space is critical, use the entire width of the window for displaying instructional text and controls for user input. When using graphics, include pictures that help illustrate the process, as shown in Figure 12.17. Include default values or settings for all controls where possible.

Petitioner Exhibit 1031, Page 382

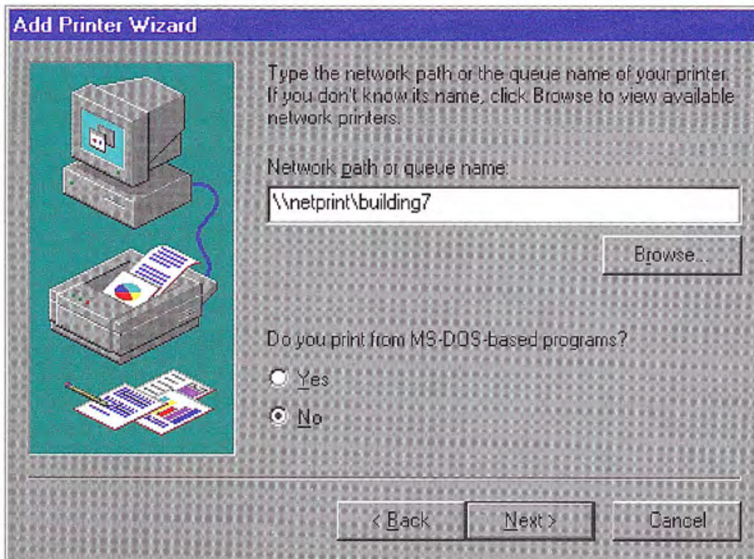


Figure 12.17 Input page for a wizard

You can include the Finish button at any point that the wizard can complete the task. For example, if you can provide reasonable defaults, you can even include the Finish button on the first page. Place the Finish button to the right and adjacent to the Next button. This allows the user to step through the entire wizard or only the page on which they wish to provide input. Otherwise, if the user needs to step through each page of the wizard, replace the Next button with the Finish button on the last page of the wizard. Also on the last page of the wizard, indicate to the user that the wizard is prepared to complete the task and instruct the user to click the Finish button.

Design your wizard pages to be easy to understand. It is important that users immediately understand what a wizard is about so they don't feel like they have to read it very carefully to understand what they have to answer. It is better to have a greater number of simple pages with fewer choices than a smaller number of complex pages

Petitioner Exhibit 1031, Page 383

with too many options or text. In addition, follow the conventions outlined in this guide and consider the following guidelines when designing a wizard:

- Minimize the number of pages that require the display of a secondary window. Novice users are often confused by the additional complexity of secondary windows.

- Avoid a wizard design that requires the user to leave the wizard to complete a task. Less experienced users are often the primary users of a wizard. Asking them to leave the wizard to perform a function can make them lose their context. Instead, design your wizard so that the user can do everything from within the wizard.

- Make it visually clear that user-interface elements that are part of a graphic illustration on a wizard page are not interactive. You can do this by varying the graphic from their normal sizes or rendering them in a more abstract representation.

- Avoid advancing pages automatically. The user may not be able to read the information before a page advances. In addition, wizards are intended to allow the user to be in control of the process that the wizard automates.

- Display a wizard window so that the user can recognize it as the primary point of input. For example, if you display a wizard from a control the user chooses in a secondary window, you may need to place the wizard window so that it partially obscures that secondary window.

- Make certain that the design alternatives offered by your wizard provide the user with positive results. You can use the context, such as the selection, to determine what options may be reasonable to provide.

- Make certain that it is obvious how the user can proceed when the wizard has completed its process. This may be accomplished by the text you include on the last page of the wizard.

Petitioner Exhibit 1031, Page 384

Guidelines for Writing Text for Wizard Pages

Use a conversational, rather than instructional, writing style for the text you provide on the screens. The following guidelines can be used to assist you in writing the textual information:

- Use words like "you" and "your."
- Start most questions with phrases like "Which option do you want..." or "Would you like...." Users respond better to questions that enable them to do a task than being told what to do. For example, "Which layout do you want?" works better in wizards than "Choose a layout."
- Use contractions and short, common words. In some cases, it may be acceptable to use slang, but you must consider localization when doing so.

For more information about localization design, see Chapter 14, "Special Design Considerations."

- Avoid using technical terminology that may be confusing to a novice user.
- Try to use as few words as possible. For example, the question "Which style do you want for this newsletter?" could be written simply as "Which style do you want?"
- Keep the writing clear, concise, and simple, but remember not to be condescending.

Petitioner Exhibit 1031, Page 385

CHAPTER

Visual Design

What we see influences how we feel and what we understand. Visual information communicates nonverbally, but very powerfully. It can include cues that motivate, direct, or distract. This chapter covers the visual and graphic design principles and guidelines that you can apply to the interface design of your Microsoft Windows-based applications.

Visual Communication

Effective visual design serves a greater purpose than decoration; it is an important tool for communication. How you organize information on the screen can make the difference between a design that communicates a message and one that leaves a user feeling puzzled or overwhelmed.

Even the best product functionality can suffer if its visual presentation does not communicate effectively. If you are not trained in visual or information design, it is a good idea to work with a designer who has education and experience in this field and include that person as a member of the design team early in the development process. Good graphic designers provide a perspective on how to take the best advantage of the screen and how to use effectively the concepts of shape, color, contrast, focus, and composition. Moreover, graphic designers understand how to design and organize information, and the effects of fonts and color on perception.

Composition and Organization

We organize what we read and how we think about information by grouping it spatially. We read a screen in the same way we read other forms of information. The eye is always attracted to the colored elements before black and white, to isolated elements before elements in a group, and to graphics before text. We even read text by scanning the shapes of groups of letters. Consider the following principles when designing the organization and composition of visual elements of your interface: hierarchy of information, focus and emphasis, structure and balance, relationship of elements, readability and flow, and unity of integration.

Hierarchy of Information

The principle of hierarchy of information addresses the placement of information based on its relative importance to other visual elements. The outcome of this ordering affects all of the other composition and organization principles, and determines what information a user sees first and what a user is encouraged to do first. To further consider this principle, ask these questions:

- What information is most important to a user?

  In other words, what are the priorities of a user when encountering your application's interface. For example, the most important priority may be to create or find a document.

- What does a user want or need to do first, second, third, and so on?

  Will your ordering of information support or complicate a user's progression through the interface?

- What should a user see on the screen first, second, third, and so on?

  What a user sees first should match the user's priorities when possible, but can be affected by the elements you want to emphasize.

Focus and Emphasis

The related principle of focus and emphasis guides you in the placement of priority items. Determining focus involves identifying the central idea, or the focal point, for activity. Determine emphasis by

Petitioner Exhibit 1031, Page 388

choosing the element that must be prominent and isolating it from the other elements or making it stand out in other ways.

Where the user looks first for information is an important consideration in the implementation of this principle. Culture and interface design decisions can govern this principle. People in western cultures, for example, look at the upper left corner of the screen or window for the most important information. So, it makes sense to put a top-priority item there, giving it emphasis.

Structure and Balance

The principle of structure and balance is one of the most important visual design principles. Without an underlying structure and a balance of visual elements, there is a lack of order and meaning and this affects all other parts of the visual design. More importantly, a lack of structure and balance makes it more difficult for the user to clearly understand the interface.

Relationship of Elements

The principle of relationship of elements is important in reinforcing the previous principles. The placement of a visual element can help communicate a specific relationship of the elements of which it is a part. For example, if a button in a dialog box affects the content of a list box, there should be a spatial relationship between the two elements. This helps the user to clearly and quickly make the connection just by looking at the placement.

Readability and Flow

This principle calls for ideas to be communicated directly and simply, with minimal visual interference. Readability and flow can determine the usability of a dialog box or other interface component. When designing the layout of a window, consider the following:

- Could this idea or concept be presented in a simpler manner?
- Can the user easily step through the interface as designed?
- Do all the elements have a reason for being there?

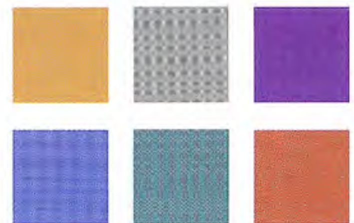
Petitioner Exhibit 1031, Page 389

Unity and Integration

The last principle, unity and integration, reflects how to evaluate a given design in relationship to its larger environment. When an application's interface is visually unified with the general interface of Windows, the user finds it easier to use because it offers a consistent and predictable work environment. To implement this principle, consider the following:

- How do all of the different parts of the screen work together visually?
- How does the visual design of the application relate to the system's interface or other applications with which it is used?

Color



Color is a very important property in the visual interface. Because color has attractive qualities, use it to identify elements in the interface to which you want to draw the user's attention — for example, the current selection. Color also has an associative aspect; we often assume there is a relationship between items of the same color. Color also carries with it emotional or psychological qualities. For example, colors are often categorized as cool or warm.

When used indiscriminately, color can have a negative or distracting effect. It can affect not only the user's reaction to your software but also productivity, by making it difficult to focus on a task.

In addition, there are a few more things to consider about using color:

- Although you can use color to show relatedness or grouping, associating a color with a particular meaning is not always obvious or easily learned.
- Color is a very subjective property. Everyone has different tastes in color. What is pleasing to you may be distasteful to someone else.
- Some percentage of your users may work with equipment that only supports monochrome presentation.

Petitioner Exhibit 1031, Page 390

- Interpretation of color can vary by culture. Even within a single culture, individual associations with color can differ.
- Some percentage of the population may have color-identification problems. For example, about 9 percent of the adult male population have some form of color confusion.

The following sections summarize guidelines for using color: color as a secondary form of information, use of a limited set of colors, allowing the option to change colors.

Color as a Secondary Form of Information

Use color as an additive, redundant, or enhanced form of information. Avoid relying on color as the only means of expressing a particular value or function. Shape, pattern, location, and text labels are other ways to distinguish information. It is also a good practice to design visuals in black and white or monochrome first, then add color.

Use of a Limited Set of Colors

Although the human eye can distinguish millions of different colors, using too many usually results in visual clutter and can make it difficult for the user to discern the purpose of the color information. The colors you use should fit their purpose. Muted, subtle, complementary colors are usually better than bright, highly saturated ones, unless you are really looking for a carnival-like appearance where bright colors compete for the user's attention.

Color also affects color. Adjacent or background colors affect the perceived brightness or shade of a particular color. A neutral color (for example, light gray) is often the best background color. Opposite colors, such as red and green, can make it difficult for the eye to focus. Dark colors tend to recede in the visual space, light colors come forward.

Petitioner Exhibit 1031, Page 391

Options to Change Colors

Because color is a subjective, personal preference, allow the user to change colors where possible. For interface elements, Windows provides standard system interfaces and color schemes. If you base your software on these system properties, you can avoid including additional controls, plus your visual elements are more likely to coordinate effectively when the user changes system colors. This is particularly important if you are designing your own controls or screen elements to match the style reflected in the system.

When providing your own interface for changing colors, consider the complexity of the task and skill of the user. It may be more helpful if you provide palettes, or limited sets of colors, that work well together rather than providing the entire spectrum. You can always supplement the palette with an interface that allows the user to add or change a color in the palette.

Fonts

Fonts have many functions in addition to providing letterforms for reading. Like other visual elements, fonts organize information or create a particular mood. By varying the size and weight of a font, we see text as more or less important and perceive the order in which it should be read.

At conventional resolutions of computer displays, fonts are generally less legible online than on a printed page. Avoid italic and serif fonts; these are often hard to read, especially at low resolutions. Figure 13.1 shows various font choices.

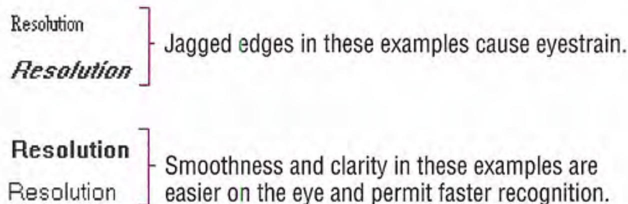


Figure 13.1 Effective and ineffective font choices

Petitioner Exhibit 1031, Page 392

Limit the number of fonts and styles you use in your software's interface. Using too many fonts usually results in visual clutter.

Wherever possible, use the standard system font for common interface elements. This provides visual consistency between your interface and the system's interface and also makes your interface more easily scaleable. Because many interface elements can be customized by the user, check the system settings for the default system font and set the fonts in your interface accordingly. For more information about system font settings, see the section, "Layout," later in this chapter.

Dimensionality

Many elements in the Windows interface use perspective, highlighting, and shading to provide a three-dimensional appearance. This emphasizes function and provides real-world feedback to the user's actions. For example, command buttons have an appearance that provides the user with natural visual cues that help communicate their functionality and differentiate them from other types of information.

Windows bases its three-dimensional effects on a common theoretical light source, the conceptual direction that light would be coming from to produce the lighting and shadow effects used in the interface. The light source in Windows comes from the upper left.

When designing your own visual elements, be careful not to overdo the use of dimensionality. Avoid unnecessary nesting of visual elements and using three-dimensional effects for an element that is not interactive. Introduce only enough detail to provide useful visual cues and use designs that blend well with the system interface.

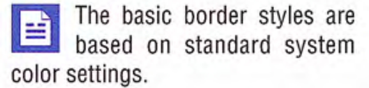
Petitioner Exhibit 1031, Page 393

Design of Visual Elements

All visual elements influence one another. Effective visual design depends on context. In a graphical user interface, a graphic element and its function are completely interrelated. A graphical interface needs to function intuitively — it needs to look the way it works and work the way it looks.

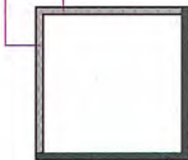
Basic Border Styles

Windows provides a unified visual design for building visual components based on the border styles shown in Figure 13.2.


The basic border styles are based on standard system color settings.

Raised Outer Border

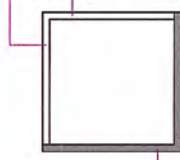
Top, left edges = Button face color



Bottom, right edges = Window frame color

Raised Inner Border

Top, left edges = Button highlight color



Bottom, right edges = Button shadow color

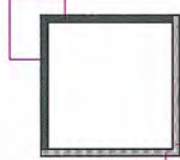
Sunken Outer Button Border

Top, left edges = Button shadow color

Bottom, right edges = Button highlight color

Sunken Inner Button Border

Top, left edges = Window frame color



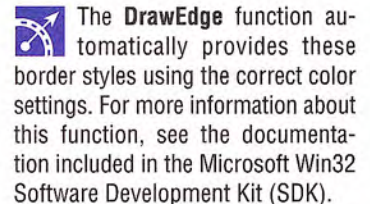
Bottom, right edges = Button face color

Figure 13.2 Basic border styles

Petitioner Exhibit 1031, Page 394

Border style	Description
Raised outer border	Uses a single-pixel width line in the button face color for its top and left edges and the window frame color for its bottom and right edges.
Raised inner border	Uses a single-pixel width line in the button highlight color for its top and left edges and the button shadow color for its bottom and right edges.
Sunken outer border	Uses a single-pixel width line in the button shadow color for its top and left border and the button highlight color for its bottom and right edges.
Sunken inner border	Uses a single-pixel width line in the window frame color for its top and left edges and the button face color for its bottom and right edges.

If you use standard Windows controls and windows, these border styles are automatically supplied for your application. If you create your own controls, your application should map the colors of those controls to the appropriate system colors so that the controls fit in the overall design of the interface when the user changes the basic system colors.


The **DrawEdge** function automatically provides these border styles using the correct color settings. For more information about this function, see the documentation included in the Microsoft Win32 Software Development Kit (SDK).

Window Border Style

The borders of primary and secondary windows, except for pop-up windows, use the window border style. Menus, scroll arrow buttons, and other situations where the background color may vary also use this border style. The border style is composed of the raised outer and raised inner basic border styles, as shown in Figure 13.3.

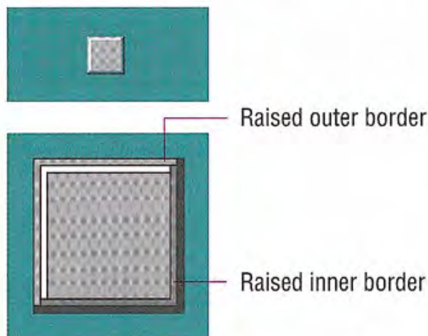


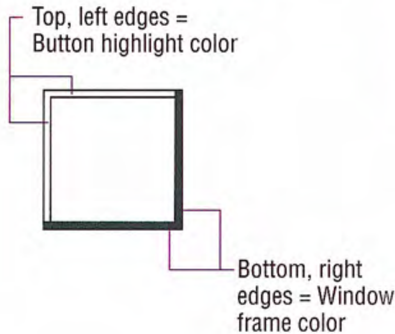
Figure 13.3 Window border style

Petitioner Exhibit 1031, Page 395

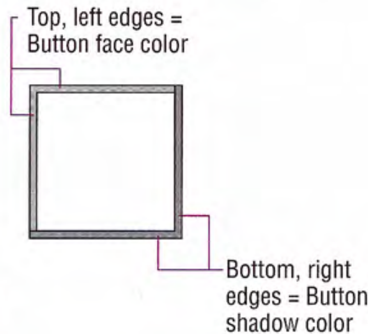
Button Border Styles

Command buttons use the button border style. The button border style uses a variation of the basic border styles where the colors of the top and left outer and inner borders are swapped when combining the borders, as shown in Figure 13.4.

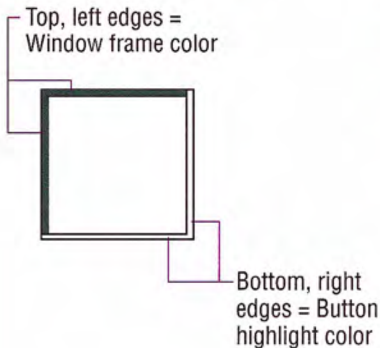
Raised Outer Button Border



Raised Inner Button Border



Sunken Outer Button Border



Sunken Inner Button Border

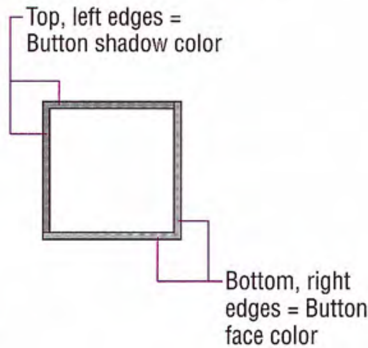


Figure 13.4 Button border styles

The normal button appearance combines the raised outer and raised inner button borders. When the user presses the button, the sunken outer and sunken inner button border styles are used, as shown in Figure 13.5.

Petitioner Exhibit 1031, Page 396

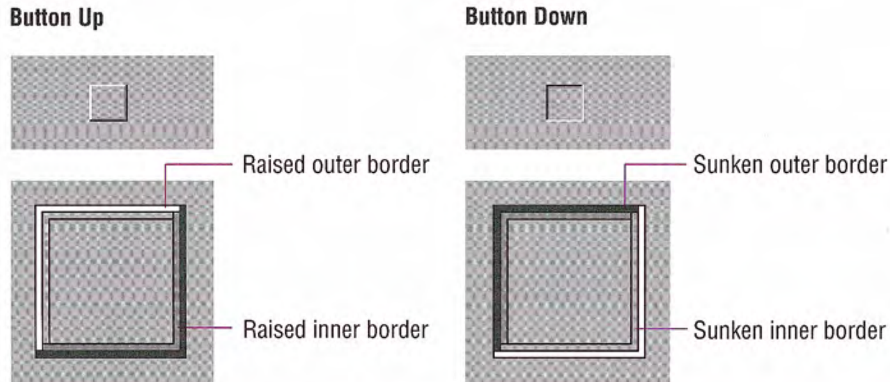


Figure 13.5 Button up and button down border styles

Field Border Style

Text boxes, check boxes, drop-down combo boxes, drop-down list boxes, spin boxes, list boxes, and wells use the field border style, as shown in Figure 13.6. You can also use the style to define the work area within a window. It uses the sunken outer and sunken inner basic border styles.

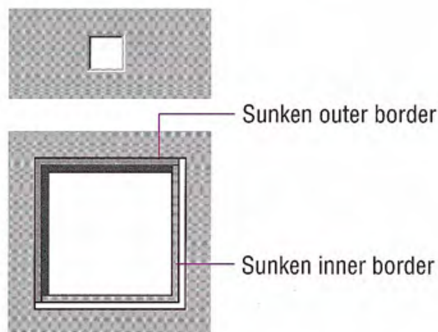


Figure 13.6 The field border style

For most controls, the interior of the field uses the button highlight color. However, in wells, the color may vary based on how the field is used or what is placed in the field, such as a pattern or color sample. For text fields, such as text boxes and combo boxes, the interior uses the button face color when the field is read-only or disabled.

Petitioner Exhibit 1031, Page 397

Status Field Border Style

Status fields use the status field border style, as shown in Figure 13.7. This style uses only the sunken outer basic border style.

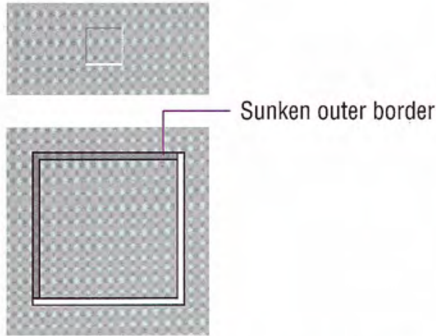


Figure 13.7 The status field border style

You use the status field style in status bars and any other read-only fields where the content of the file can dynamically change.

Grouping Border Style

Group boxes and menu separators use the grouping border style, as shown in Figure 13.8. The style uses the sunken outer and raised inner basic border styles.

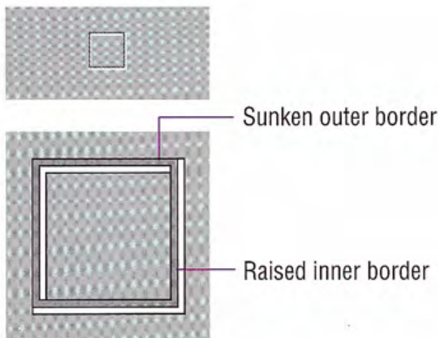
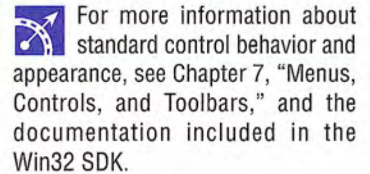


Figure 13.8 The group border style

Petitioner Exhibit 1031, Page 398

Visual States for Controls

The visual design of controls includes the various states supported by the control. If you use standard Windows controls, Windows automatically provides specific appearances for these states. If you design your own controls, use the information in the previous section for the appropriate border style and information in the following sections to make your controls consistent with standard Windows controls.



For more information about standard control behavior and appearance, see Chapter 7, "Menus, Controls, and Toolbars," and the documentation included in the Win32 SDK.

Pressed Appearance

When the user presses a control, it provides visual feedback on the down transition of the mouse button. (For the pen, the feedback provided is for when the pen touches the input surface and for the keyboard, upon the down transition of the key.)

For standard Windows check boxes and option buttons, the background of the button field is drawn using the button face color, as shown in Figure 13.9.

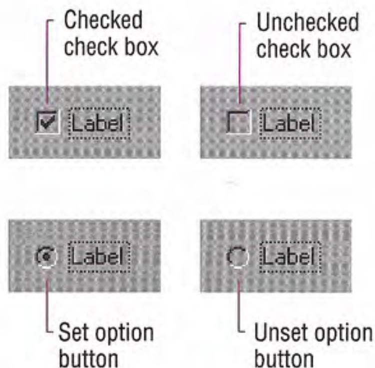


Figure 13.9 Pressed appearance for check boxes and option buttons

For command buttons, the button-down border style is used and the button label moves down and to the right by one pixel, as shown in Figure 13.10.

Petitioner Exhibit 1031, Page 399

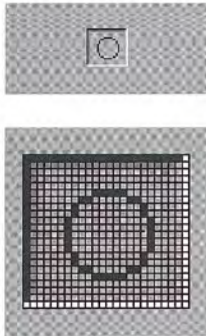


Figure 13.10 Pressed appearance for a command button

Option-Set Appearance

When using buttons to indicate when its associated value or state applies or is currently set, the controls provide an *option-set appearance*. The option-set appearance is used upon the up transition of the mouse button or pen tip, and the down transition of a key. It is visually distinct from the pressed appearance.

Standard check boxes and option buttons provide a special visual indicator when the option corresponding to that control is set. A check box uses a check mark, and an option button uses a dot that appears inside the button, as shown in Figure 13.11.

Figure 13.11 Option-set appearance for check boxes and option buttons

When using command buttons to represent properties or other state information, the button face reflects when the option is set. The button continues to use the button-down border style, but a checkerboard pattern (dither) using the color of the button face and button highlight is displayed on the interior background of the button, as shown in Figure 13.12. For configurations that support 256 or more colors, if the button highlight color setting is not white, the button interior background is drawn in a halftone between button highlight color and button face color. The glyph on the button does not otherwise change from the pressed appearance.

Petitioner Exhibit 1031, Page 400

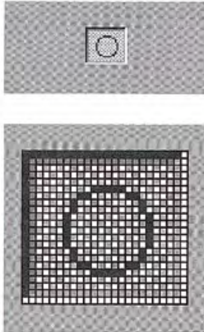


Figure 13.12 Option-set appearance for a command button

For well controls (shown in Figure 13.13), when a particular choice is set, place a border around the control, using the window text color and the button highlight color.

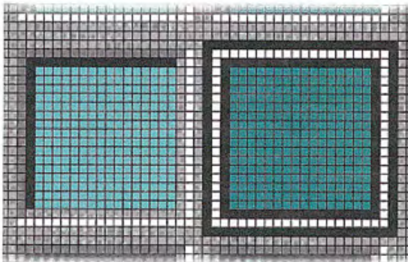


Figure 13.13 Option-set appearance for a well

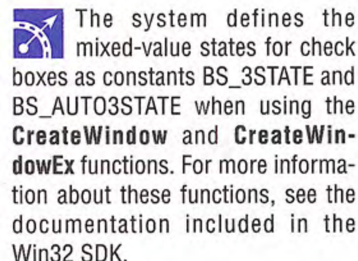
Mixed-Value Appearance

When a control represents a property or other setting that reflects a set of objects where the values are different, the control is displayed with a *mixed-value* appearance (also referred to as indeterminate appearance), as shown in Figure 13.14.

For most standard controls, leave the field with no indication of a current set value if it represents a mixed value. For example, for a drop-down list, the field is blank.

Petitioner Exhibit 1031, Page 401

Standard check boxes support a special appearance for this state that displays the check mark, in the button shadow color, against a checkerboard background that uses the button highlight color and button face color. For configurations that support 256 or more colors, if the button highlight color setting is not white, the interior of the control is drawn in a halftone between button highlight color and button face color.



The system defines the mixed-value states for check boxes as constants BS_3STATE and BS_AUTO3STATE when using the **CreateWindow** and **CreateWindowEx** functions. For more information about these functions, see the documentation included in the Win32 SDK.

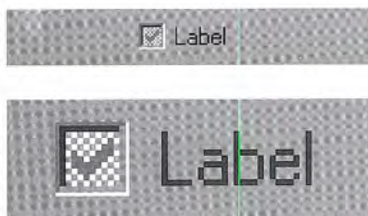


Figure 13.14 Mixed-value appearance for a check box

For graphical command buttons, such as those used on toolbars, the checkerboard pattern, using the button highlight color and button face color, or the halftone color, is drawn on the background of the button face, as shown in Figure 13.15. The image is converted to a monochrome presentation and drawn in the button shadow color.

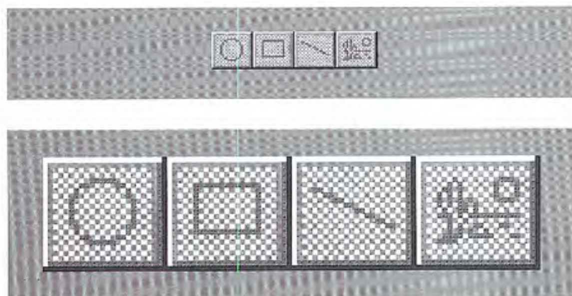


Figure 13.15 Mixed-value appearance for buttons

For check box and command button controls displaying mixed-value appearance, when the user clicks the button, the property value or state is set. Clicking a second time clears the value. As an option, you can support a third click to return the button to the mixed-value state.

Petitioner Exhibit 1031, Page 402

Unavailable Appearance

When a control is unavailable (also referred to as disabled), its normal functionality is no longer available to the user (though it can still support access to contextual Help information) because the functionality represented does not apply or is inappropriate under the current circumstances. To reflect this state, the label of the control is rendered with a special *unavailable appearance*, as shown in Figure 13.16.

Figure 13.16 Unavailable appearance for check boxes and option buttons

For graphical or text buttons, create the engraved effect by converting the label to monochrome and drawing it in the button highlight color. Then overlay it, at a small offset, with the label drawn in the button shadow color, as shown in Figure 13.17.

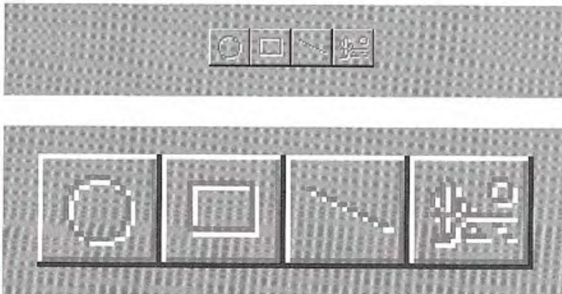


Figure 13.17 Unavailable appearance for buttons

If a check box or option button is set, but the control is unavailable, then the control's label is displayed with an unavailable appearance, and its mark appears in the button shadow color, as shown in Figure 13.18.

Figure 13.18 Unavailable appearance for check boxes and option buttons (when set)

Petitioner Exhibit 1031, Page 403

If a graphical button needs to reflect both the set and unavailable appearance (as shown in Figure 13.19), omit the background checkerboard pattern and combine the option-set and the unavailable appearance for the button's label.

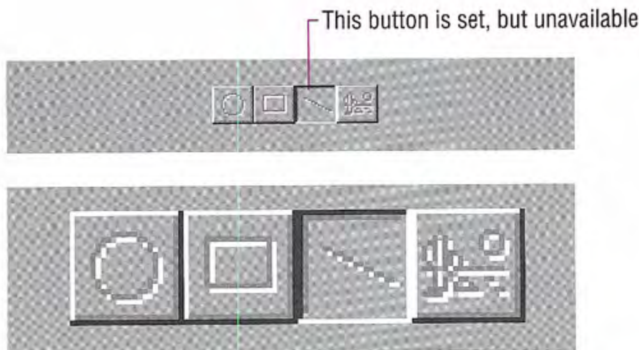


Figure 13.19 Unavailable and option-set appearance for buttons

Input Focus Appearance

You can provide a visual indication so the user knows where the input focus is. For text boxes, the system provides a blinking cursor, or insertion point. For other controls a dotted outline is drawn around the control or the control's label, as shown in Figure 13.20.

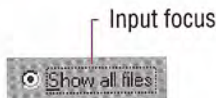
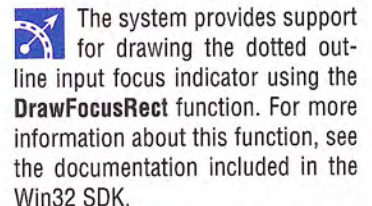


Figure 13.20 Example of input focus in a control

The system provides the input focus appearance for standard controls. To use it with your own custom controls, specify the rectangle to allow at least one pixel of space around the extent of the control. If the input focus indicator would be too intrusive, as an option, you can include it around the label for the control. Display the input focus when the mouse button (pen tip) is pressed while over a control; for the keyboard, display the input focus when a navigation or access key for the control is pressed.



The system provides support for drawing the dotted outline input focus indicator using the **DrawFocusRect** function. For more information about this function, see the documentation included in the Win32 SDK.

Petitioner Exhibit 1031, Page 404

Flat Appearance

When you nest controls inside of a scrollable region or control, avoid using a three-dimensional appearance because it may not work effectively against the background. Instead, use the flat appearance style, as shown in Figure 13.21.

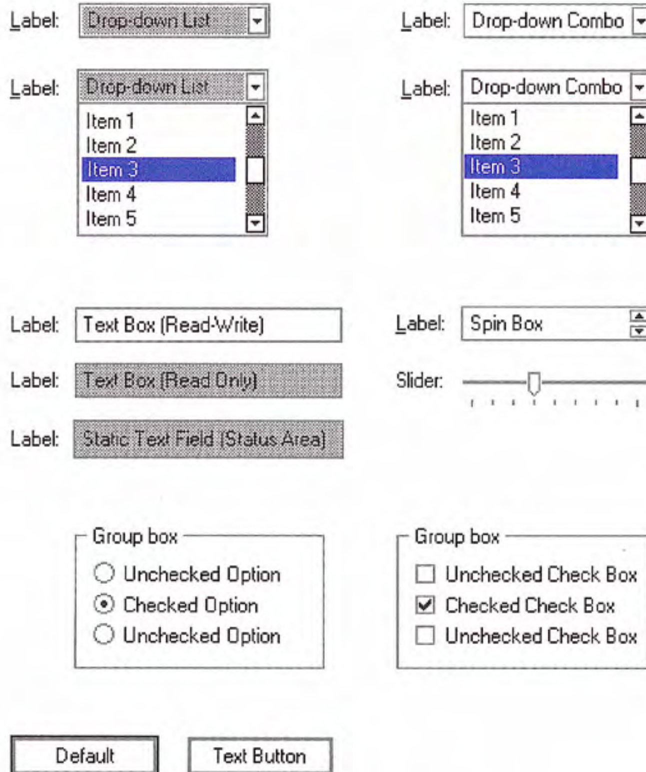
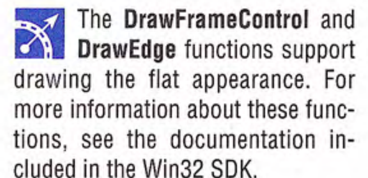


Figure 13.21 Flat appearance for standard controls

The system provides support for the flat appearance style for standard controls. It also includes support for drawing the edges of your own custom controls so you can match the appearance used by standard system controls.



The **DrawFrameControl** and **DrawEdge** functions support drawing the flat appearance. For more information about these functions, see the documentation included in the Win32 SDK.

Petitioner Exhibit 1031, Page 405

Layout

Size, spacing, and placement of information are critical in creating a visually consistent and predictable environment. Visual structure is also important for communicating the purpose of the elements displayed in a window. In general, follow the layout conventions for how information is read. In western countries, this means left-to-right, top-to-bottom, with the most important information located in the upper left corner.

Font and Size

The default system font is a key metric in the presentation of visual information. The default font used for interface elements in Windows (U.S. release) is MS® Sans Serif for 8-point. Menu bar titles, menu items, control labels, and other interface text all use 8-point MS Sans Serif. Title bar text also uses the 8-point MS Sans Serif bold font, as shown in Figure 13.22. However, because the user can change the system font, make certain you check this setting and adjust the presentation of your interface appropriately rather than assuming a fixed size for fonts or other visual elements. Also adjust your presentation when the system notifies your application that these settings have changed.

The **GetSystemMetrics** (standard windows elements), **SystemParametersInfo** (primary windows fonts), and **GetStockObject** (secondary windows fonts) functions provide the current system settings. The WM_SETTINGCHANGE message notifies applications when these settings change. For more information about these APIs, see the documentation included in the Win32 SDK.

Petitioner Exhibit 1031, Page 406

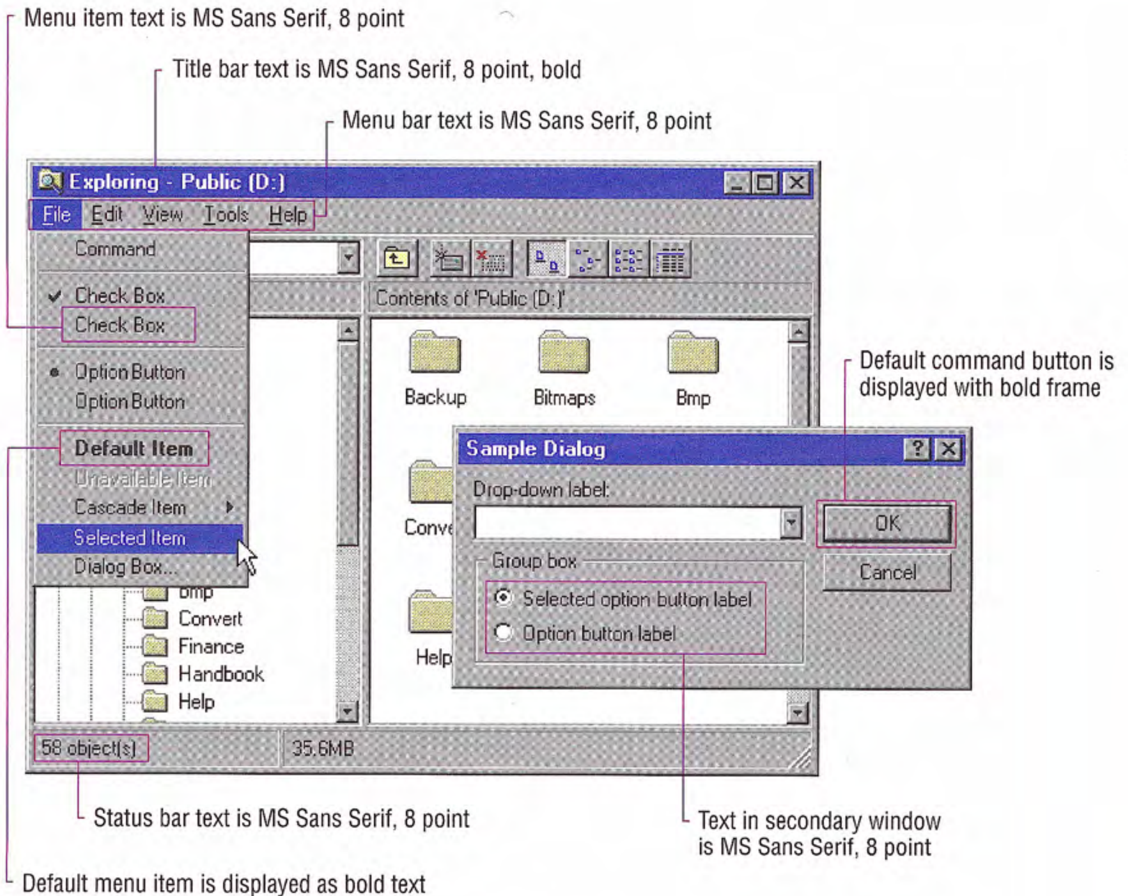


Figure 13.22 Default font usage in windows

The system also provides settings for the font and size of many system components including title bar height, menu bar height, border width, title bar button height, icon title text, and scroll bar height and width. When designing your window layouts, take these variables into consideration so that your interface will scale appropriately. In addition, use these standard system settings to determine the size of your custom interface elements.

Petitioner Exhibit 1031, Page 407

The system defines the size and location of user-interface elements in a window based on dialog base units, not pixels. A *dialog base unit* is the device-independent measure to use for layout. One horizontal dialog base unit is equal to one-fourth of the average character width for the current system font. One vertical dialog base unit is equal to one-eighth of an average character height for the current system font. The default height for most single-line controls is 14 dialog base units. Be careful when using a pixel-based drawing program because this may not provide an accurate representation when you translate your design into dialog base units.

Your application can retrieve the number of pixels per base unit for the current display using the **GetDialogBaseUnits** function. For more information about this function, see the documentation included in the Win32 SDK.

If a menu item represents a default command, the text is bold. Default command buttons use a bold outline around the button. In general, use nonbold text in your windows. Use bold text only when you want to call attention to an area or create a visual hierarchy.

Avoid displaying any secondary window larger than 263-dialog bases units x 263-dialog base units. The recommended sizes for property sheets are 252-dialog base units wide x 218-dialog base units high, 227-dialog bases units wide x 215-dialog base units high, and 212-dialog base units x 188-dialog base units high. These sizes keep the window from becoming too large to display at some resolutions, and still provide reasonable space to display supportive information, such as Help windows, that apply to the dialog box or property sheet.

For easy readability, make buttons a consistent length. However, if maintaining this consistency greatly expands the space required by a set of buttons, it may be reasonable to have one button larger than the rest.

Similarly, when using tabs, try to maintain a consistent width for all tabs in the same window (and same dimension). However, if a particular tab's label makes this unworkable, you can size it larger, and maintaining a smaller, consistent size for the other tabs. If a tab's label contains variable text, you may want to size the tab to fit the label, up to some reasonable maximum, after which you truncate and add an ellipsis.

Provide toolbar buttons in at least two different sizes: 24-pixels wide x 22-pixels high and 32-pixels wide x 30-pixels high. This includes the border. The image size you include as the button's label should be 16 x 16 pixels and 24 x 24 pixels, respectively. It is best to center

For more information about common toolbar images, see Chapter 7 "Menus, Controls, and Toolbars."

Petitioner Exhibit 1031, Page 408

the image on the button's face. Use the smaller size as your default presentation and provide an option for users to change the size. Be certain that you include button images to support all visual states for the buttons.

For larger buttons on very high resolution displays, you can proportionally size the button to be the same height as a text box control. This allows the button to maintain its proportion with respect to other controls in the toolbar. You can stretch the image when the button is an even multiple of the basic sizes. Alternatively, you can supply additional image sizes. This may be preferable, because it provides better visual results.

Toolbar buttons generally have only graphical labels and no accompanying textual label. You can use a tooltip to provide the name of the button.

Capitalization

When displaying text in menus, command buttons, and tabs, use conventional book title capitalization. For example, for U.S. versions, capitalize the first letter in each word unless it is an article or preposition not occurring at the beginning or end of the name, or unless the word's conventional usage is not capitalized. For example:

Insert Object
Paste Link
Save As
Go To
Always on Top
By Name

Use this same convention for default names you provide for filenames, title bar text, or icon labels. Of course, if the user supplies a name for an object, display the name as the user specifies it, regardless of case.

Petitioner Exhibit 1031, Page 409

Field labels, such as those used for option buttons, check boxes, text boxes, group boxes, and page tabs, should use sentence-style capitalization. For U.S. versions, this means capitalize only the first letter of the initial word and any words that are normally capitalized. For example:

Extended (XMS) memory
Working directory
Print to
Find whole words only

Grouping and Spacing

Group related components together. You can use group box controls or spacing. Leave at least four dialog base units between controls. Although you can also use color to visually group objects, it is not a common convention and could result in undesirable effects when the user changes color schemes.

Maintain a consistent margin (seven dialog base units is recommended) from the edge of the window. Use spacing between groups within the window. Figure 13.23 shows the recommended spacing.

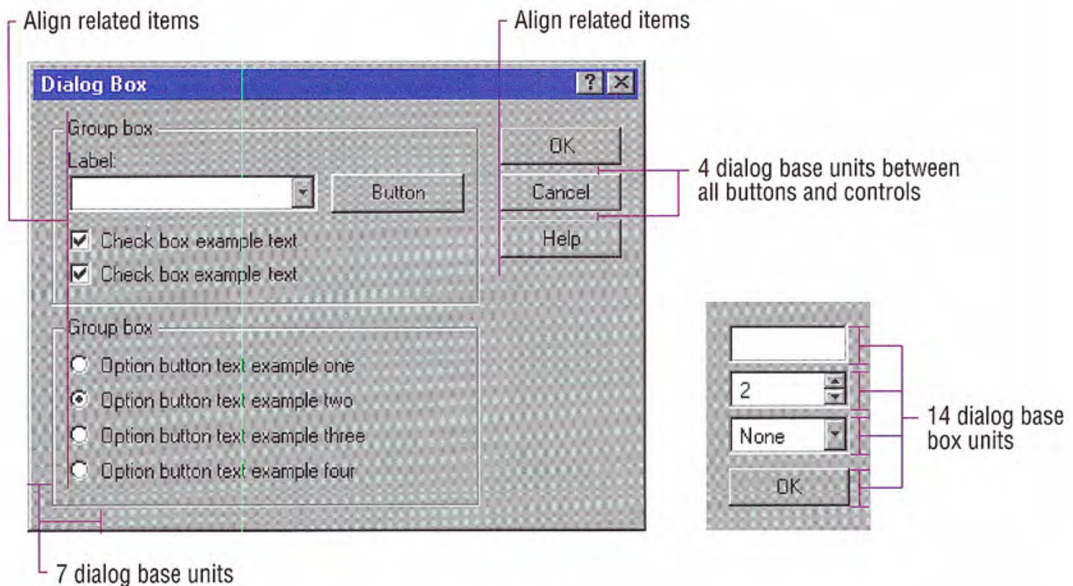


Figure 13.23 Recommended layout and spacing of controls and text

Petitioner Exhibit 1031, Page 410

Position controls in a toolbar so that there is at least a window's border width from the edges of the toolbar. Use at least four dialog base units spacing between controls, unless you want to align a set of related toolbar buttons adjacently. Use adjacent alignment for toolbar buttons that are related. For example, when using toolbar buttons like a set of option buttons, align them without any spacing between them.

Alignment

When information is positioned vertically, align fields by their left edges (in western countries). This usually makes it easier for the user to scan the information. Text labels are usually left aligned and placed above or to the left of the areas to which they apply. When placing text labels to the left of text box controls, align the height of the text with text displayed in the text box.

Placement

Stack the main command buttons in a secondary window in the upper right corner or in a row along the bottom, as shown in Figure 13.24. If there is a default button, it is typically the first button in the set. Place OK and Cancel buttons next to each other. The last button is a Help button (if supported). If there is no OK button, but other command buttons, it is best to place the Cancel button at the end of a set of action buttons, but before a Help button. If a particular command button applies only to a particular field, group it with that field.

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

Petitioner Exhibit 1031, Page 411

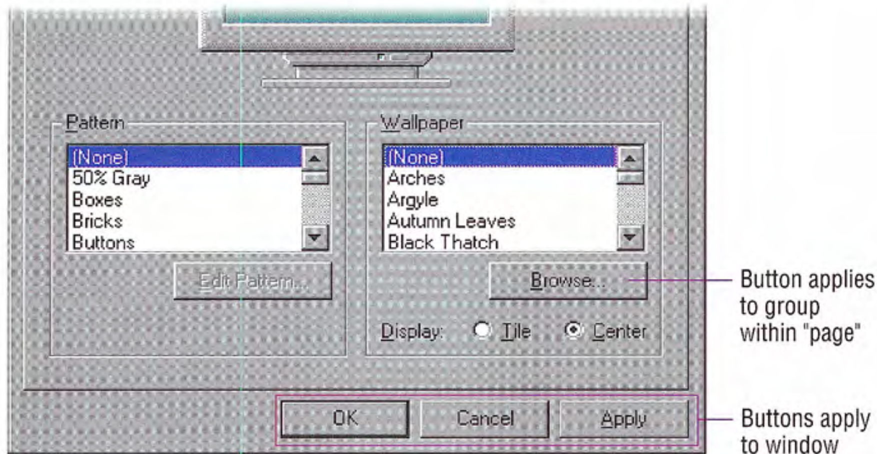


Figure 13.24 Examples of layout of buttons

Placement of command buttons (or other controls) within a tabbed page implies the application of only the transactions on that page. If command buttons are placed within the window, but not on the tabbed page, they apply to the entire window.

Design of Graphic Images

When designing pictorial representations of objects, whether they are icons or graphical buttons, begin by defining the icon's purpose and its use. Brainstorm about possible ideas, considering real-world metaphors. It is often difficult to design icons that define operations or processes — activities that rely on verbs. Consider nouns instead. For example, scissors can represent the action of Cut.

Draw your ideas using an icon-editing utility or pixel (bitmapped) drawing package. Drawing them directly on the screen provides immediate feedback about their appearance. It is a good idea to begin the design in black and white. Consider color as an enhancing property. Also, test your images on different backgrounds. They may not always be seen against white or gray backgrounds.

Petitioner Exhibit 1031, Page 412

Consistency is also important in the design of graphic images. As with other interface elements, design images assuming a light source from the upper left. In addition, make certain the scale (size) and orientation of your graphics are consistent with the other objects to which they are related and fit well within the working environment.

Avoid using a triangular arrow graphic similar to the one used in cascading menus, drop-down controls, and scroll arrows. When this image appears on a button, it implies that the control will display additional information. For example, you can use an arrow graphic to designate a menu button.

You may want to use a technique called anti-aliasing when designing graphic images. *Anti-aliasing* involves adding colored pixels to smooth the jagged edges of a graphic. However, avoid using anti-aliasing on the outside edge of an icon because the contrasting pixels may look jagged or fuzzy on varying backgrounds.

Finally, remember to consider the potential cultural impact of your graphics. What may have a certain meaning in one country or culture may have unforeseen meanings in another. It is best to avoid letters or words, if possible, as this may make the graphics difficult to apply for other cultures.

For more information about designing for international audiences, see Chapter 14, "Special Design Considerations."

Icon Design

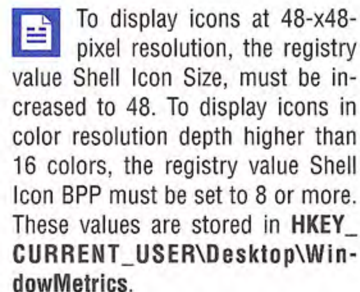
Icons are used throughout the Windows interface to represent objects or tasks. Because the system uses icons to represent your software's objects, it is important to not only supply effective icons, but to design them to effectively communicate their purpose.

When designing icons, design them as a set, considering their relationship to each other and to the user's tasks. Do several sketches or designs and test them for usability.

Petitioner Exhibit 1031, Page 413

Sizes and Types

Supply icons for your application in all standard sizes: 16 x 16 pixel (16 color), 32 x 32 pixel (16 color), and 48 x 48 pixel (256 color), as shown in Figure 13.25. Although you can also include greater color depth for the smaller sizes, it increases the storage requirements for the icons and may not be displayable on all computer configurations. Therefore, if you choose to provide 256 color icons in the smaller sizes, do so in addition to providing the standard (16 color) format.



To display icons at 48-x48-pixel resolution, the registry value Shell Icon Size, must be increased to 48. To display icons in color resolution depth higher than 16 colors, the registry value Shell Icon BPP must be set to 8 or more. These values are stored in **HKEY_CURRENT_USER\Desktop\WindowMetrics**.

Figure 13.25 Three sizes of icons

The system automatically maps colors in your icon design for monochrome configurations. However, you should check your icon design in monochrome configuration. If the result is not satisfactory, author and supply monochrome icons as well.

Define icons not only for your application executable file, but also for all data file types supported by your application, as shown in Figure 13.26.

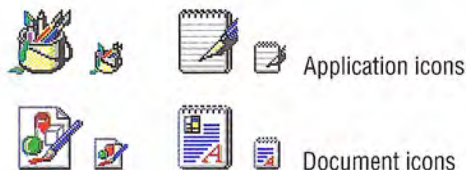
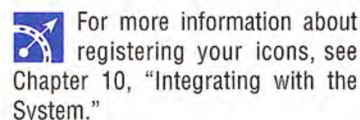


Figure 13.26 Application and supported document icons

Icons for documents and data files should be distinct from the application's icon. Include some common element of the application's icon, but focus on making the document icon recognizable and representative of the file's content.

Register the icons you supply in the system registry. If your software does not register any icons, the system automatically provides one based on your application's icon, as shown in Figure 13.27. However, it is unlikely to be as detailed or distinctive as one you can supply.



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

Petitioner Exhibit 1031, Page 414

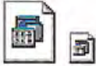
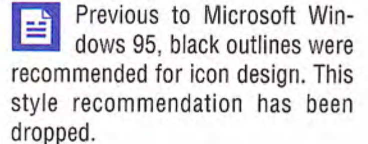


Figure 13.27 System-generated icon for file type without a registered icon

Icon Style

When designing your icons, use a common style across all icons. Repeat common characteristics, but avoid repeating unrelated elements.



Previous to Microsoft Windows 95, black outlines were recommended for icon design. This style recommendation has been dropped.

An illustrative style tends to communicate metaphorical concepts more effectively than abstract symbols. However, in designing an image based on a real-world object, use only the amount of detail that is really necessary for user recognition and recall. Where possible and appropriate, use perspective and dimension (lighting and shadow) to better communicate the real-world representation, as shown in Figure 13.28.

Figure 13.28 Perspective and dimension improve graphics

User recognition and recollection are two important factors to consider in icon design. Recognition means that the icon is identifiable by the user and easily associated with a particular object. Support user recognition by using effective metaphors. Use real-world objects to represent abstract ideas so that the user can draw from previous learning and experiences. Exploit the user's knowledge of the world and allude to the familiar.

To facilitate recollection, design your icons to be simple and distinct. Applying the icon consistently also helps build recollection; therefore, design your small icons to be as similar as possible to their larger counterparts. It is generally best to try to preserve general shape and any distinctive detail. 48 x 48-pixel icons can be rendered in 256 colors. This allows very realistic-looking icons, but focus on simplicity and careful use of color. If your software is targeted at computers that can only display 256 colors, make certain you only use colors from the system's standard 256-color palette. If you aim at computers configured for 65,000 or more colors, you can use any combination of colors.

Petitioner Exhibit 1031, Page 415

Pointer Design

You can use a pointer's design to help the user identify objects and provide feedback about certain conditions or states. However, use pointer changes conservatively so that the user is not distracted by excessive flashing of multiple pointer changes while traversing the screen. One way to handle this is to use a time-out before making noncritical pointer changes.

For more information about some of the common pointers, see Chapter 4, "Input Basics." For information about displaying pointers for drag and drop operations, see Chapter 5, "General Interaction Techniques."

When you use a pointer to provide feedback, use it only over areas where that state applies. For example, when using the hourglass pointer to indicate that a window is temporarily noninteractive, if the pointer moves over a window that is interactive, change it to its appropriate interactive image. If a process makes the entire interface non-interactive, display the hourglass pointer wherever the user moves the pointer.

Pointer feedback may not always be sufficient. For example, for processes that last longer than a few seconds, it is better to use a progress indicator that indicates progressive status, elapsed time, estimated completion time, or some combination of these to provide more information about the state of the operation. In other situations, you can use command button states to reinforce feedback; for example, when the user chooses a drawing tool.

Use a pointer that best fits the context of the activity. The I-beam pointer is best used to select text. The normal arrow pointer works best for most drag and drop operations, modified when appropriate to indicate copy and link operations.

The location for the hot spot of a pointer (shown in Figure 13.29) is important for helping the user target an object. The pointer's design should make the location of the hot spot intuitive. For example, for a cross-hair pointer, the implied hot spot is the intersection of the lines.

Petitioner Exhibit 1031, Page 416

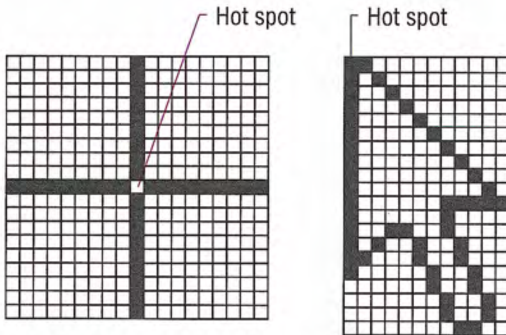


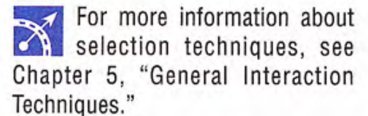
Figure 13.29 Pointer hot spots

Animating a pointer can be a very effective way of communicating information. However, remember that the goal is to provide feedback, not to distract the user. In addition, pointer animation should not restrict the user's ability to interact with the interface.

Selection Appearance

When the user selects an item, provide visual feedback to enable the user to distinguish it from items that are not selected. Selection appearance generally depends on the object and the context in which the selection appears.

Display an object with selection appearance as the user performs a selection operation. For example, display selection appearance when the user presses the mouse button to select an object.


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

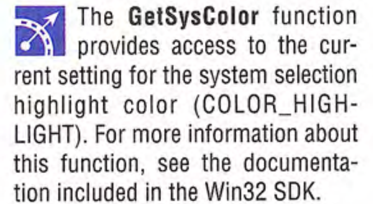
It is best to display the selection appearance only for the scope, area, or level (window or pane) that is active. This helps the user recognize which selection currently applies and the extent of the scope of that selection. Therefore, avoid displaying selections in inactive windows or panes, or at nested levels.

Petitioner Exhibit 1031, Page 417

However, in other contexts, it may still be appropriate to display selection appearance simultaneously in multiple contexts. For example, when the user selects an object and then selects a menu item to apply to that object, selection appearance is always displayed for both the object and the menu item because it is clear where the user is directing the input. In cases where you need to show simultaneous selection, but with the secondary selection distinguished from the active selection, you can draw an outline in the selection highlight color around the secondary selection or use some similar variant of the standard selection highlight technique.

Highlighting

For many types of objects, you can display the object or its background or some distinguishing part of the object using the system highlight color. Figure 13.30 shows examples of selection appearances.



The **GetSysColor** function provides access to the current setting for the system selection highlight color (COLOR_HIGHLIGHT). For more information about this function, see the documentation included in the Win32 SDK.

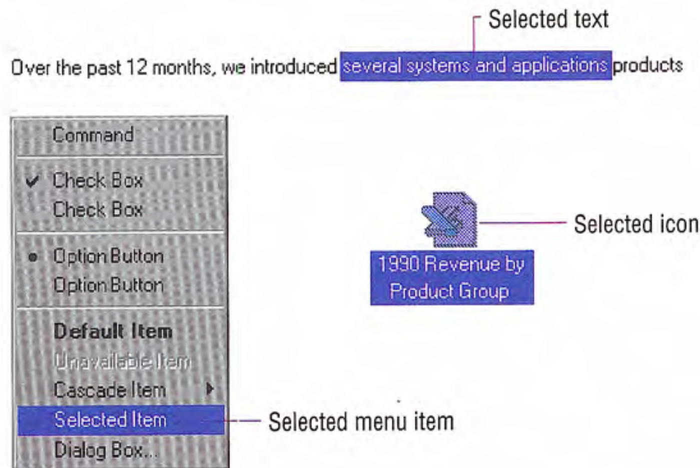


Figure 13.30 Examples of selection appearance

In a secondary window, it may be appropriate to display selection highlighting when the highlight is also being used to reflect the setting for a control. For example, in list boxes, highlighting often indicates a current setting. In cases like this, provide an input focus indication as well so the user can distinguish when input is being directed to another control in the window; you can also use check marks instead of highlighting to indicate the setting.

Petitioner Exhibit 1031, Page 418

Handles

Handles provide access to operations for an object, but they can also indicate selection for some kinds of objects. The typical handle is a solid, filled square box that appears on the edge of the object, as shown in Figure 13.31.

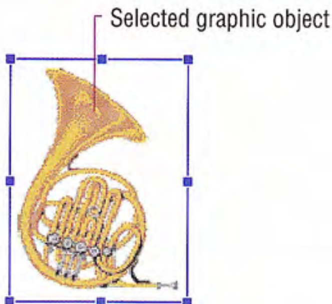


Figure 13.31 Selected graphic object with handles

The handle is "hollow" when the handle indicates selection, but is not a control point by which the object may be manipulated. Figure 13.32 shows a solid and a hollow handle.

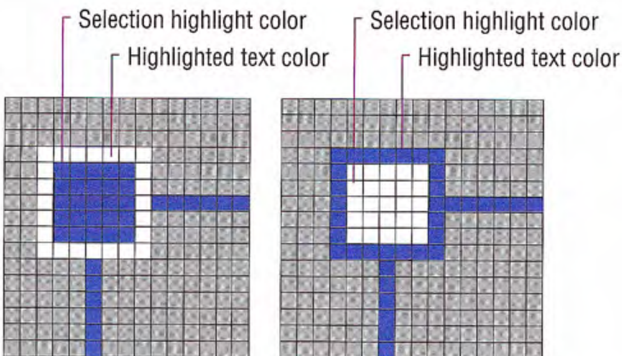
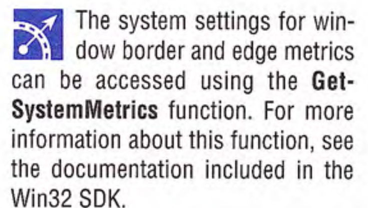


Figure 13.32 Solid and hollow handles

Base the default size of a handle on the current system settings for window border and edge metrics so that your handles are appropriately sized when the user explicitly changes window border widths or to accommodate higher resolutions. Similarly, base the colors you use to draw handles on system color metrics so that when the user changes the default system colors, handles change appropriately.



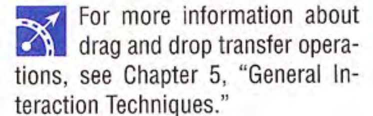
The system settings for window border and edge metrics can be accessed using the **GetSystemMetrics** function. For more information about this function, see the documentation included in the Win32 SDK.

Petitioner Exhibit 1031, Page 419

When using handles to indicate selection, display the handle in the system highlight color. To help distinguish the handle from the variable background, draw a border and the edge of the handle using the system's setting for highlighted text. For hollow handles use the opposite: selection highlight color for the border and highlighted text color for the fill color. If you display handles for an object even when it is not selected, display handles in a different color, such as the window text color, so that the user will not confuse it as part of the active selection.

Transfer Appearance

When the user drags an object to perform an operation, for example, move, copy, print, and so on, display a representation of the object that moves with the pointer. In general, do not simply change the pointer to be the object, as this may obscure the insertion point at some destinations. Instead, use a translucent or outline representation of the object that moves with the pointer, as shown in Figure 13.33.



For more information about drag and drop transfer operations, see Chapter 5, "General Interaction Techniques."

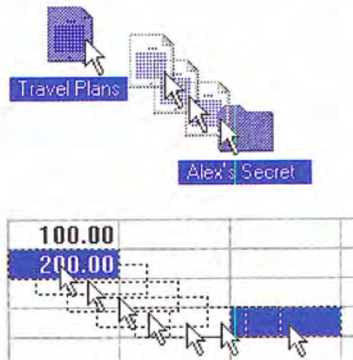


Figure 13.33 Translucent and outline representation (drag transfer)

You can create a translucent representation by using a checkerboard mask made up of 50 percent transparent pixels. When used together with the object's normal appearance, this provides a representation that allows part of the destination to show through. An outline representation should also use a translucent or transparent interior and a gray or dotted outline.

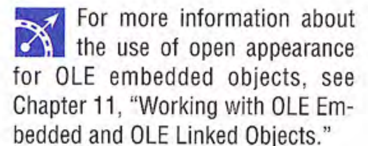
Petitioner Exhibit 1031, Page 420

The presentation of an object being transferred displayed is always defined by the destination. Use a representation that best communicates how the transferred object will be incorporated when the user completes the drag transfer. For example, if the object being dragged will be displayed as an icon, then display an icon as its representation. If, on the other hand, it will be incorporated as native content, then display an appropriate representation. For example, you could display a graphics object as an outline or translucent image of its shape, a table cell as the outline of a rectangular box, and text selection as the first few characters of a selection with a transparent background.

Set the pointer image to be whatever pointer the target location uses for directly inserting information. For example, when dragging an object into normal text context, use the I-beam pointer. In addition, include the copy or link image at the bottom right of the pointer if that is the interpretation for the operation.

Open Appearance

Open appearance is most commonly used for an OLE embedded object, but it can also apply in other situations where the user opens an object into its own window. To indicate that an object is "open," display the object in its container's window overlaid with a set of hatched (45 degree) lines drawn every four pixels, as shown in Figure 13.34.



For more information about the use of open appearance for OLE embedded objects, see Chapter 11, "Working with OLE Embedded and OLE Linked Objects."

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

	1983	1987	1991
CD's	6,345K	16,652K	32,657K
LP's	31,538K	28,571K	17,429K
Total	37,883K	45,223K	50,086K

Figure 13.34 An object with opened appearance

If the opened object is selected, display the hatched lines using the system highlight color. When the opened object is not selected, display the lines using the system's setting for window text color.

Petitioner Exhibit 1031, Page 421

Animation

Animation can be an effective way to communicate information. For example, it can illustrate the operation of a particular tool or reflect a particular state. It can also be used to include an element of fun in your interface. You can use animation effects for objects within a window and interface elements, such as icons, buttons, and pointers.

Effective animation involves many of the same design considerations as other graphics elements, particularly with respect to color and sound. Fluid animation requires presenting images at 16 (or more) frames per second.

When you add animation to your software, ensure that it does not affect the interactivity of the interface. Do not force the user to remain in a modal state to allow the completion of the animation. Unless animation is part of a process, make it interruptible by the user or independent of the user's primary interaction.

Avoid gratuitous use of animation. When animation is used for decorative effect it can distract or annoy the user. You may want to provide the user with the option of turning off the animation or otherwise customizing the animation effects.

Petitioner Exhibit 1031, Page 422

CHAPTER

Special Design Considerations

A well-designed application for Microsoft Windows must consider other factors to appeal to the widest possible audience. This chapter covers special user interface design considerations, such as support for sound, accessibility, internationalization, and network computing.

Sound

You can incorporate sound as a part of an application in several ways — for example, music, speech, or sound effects. Such auditory information can take the following forms:

- A primary form of information, such as the composition of a particular piece of music or a voice message.
- An enhancement of the presentation of information that is not required for the operation of the software.
- A notification or alerting of users to a particular condition.

Sound can be an effective form of information and interface enhancement when appropriately used. However, avoid using sound as the only means of conveying information. Some users may be hard-of-hearing or deaf. Others may work in a noisy environment or in a setting that requires that they disable sound or maintain it at a low volume. In addition, like color, sound is a very subjective part of the interface.

Petitioner Exhibit 1031, Page 423

As a result, sound is best incorporated as a redundant or secondary form of information, or supplemented with alternative forms of communication. For example, if a user turns off the sound, consider flashing the window's title bar, taskbar button, presenting a message box, or other means of bringing the user's attention to a particular situation. Even when sound is the primary form of information, you can supplement the audio portion by providing visual representation of the information that might otherwise be presented as audio output, such as captioning or animation.

The taskbar can also provide visual status or notification information. For more information about using the taskbar for this purpose, see Chapter 10, "Integrating with the System."

Always allow the user to customize sound support. Support the standard system interfaces for controlling volume and associating particular sounds with application-specific sound events. You can also register your own sound events for your application.

The system provides a global system setting, ShowSounds. The setting indicates that the user wants a visual representation of audio information. Your software should query the status of this setting and provide captioning for the output of any speech or sounds. Captioning should provide as much information visually as is provided in the audible format. It is not necessary to caption ornamental sounds that do not convey useful information.

The **GetSystemMetrics** function provides access to the ShowSounds and SoundSentry settings. For more information about this function and the settings, see the documentation included in the Microsoft Win32 Software Development Kit (SDK).

Do not confuse ShowSounds with the system's SoundSentry option. When the user sets the SoundSentry option, the system automatically supplies a visual indication whenever a sound is produced. Avoid relying on SoundSentry alone if the ShowSounds option is set because SoundSentry only provides rudimentary visual indications, such as flashing of the display or screen border, and it cannot convey the meaning of the sound to the user. The system provides SoundSentry primarily for applications that do not provide support for ShowSounds. The user sets either of these options with the Microsoft Windows Accessibility Options.

In Microsoft Windows 95, SoundSentry only works for audio output directed through the internal PC speaker.

Petitioner Exhibit 1031, Page 424

Accessibility

Accessibility means making your software usable and accessible to a wide range of users, including those with disabilities. Many users may require special accommodation because of temporary or permanent disabilities.

The issue of software accessibility in the home and workplace is becoming increasingly important. Nearly one in five Americans have some form of disability — and it is estimated that 30 million people in the U.S. alone have disabilities that may be affected by the design of your software. In addition, between seven and nine out of every ten major corporations employ people with disabilities who may need to use computer software as part of their jobs. As the population ages and more people become functionally limited, accessibility for users with disabilities will become increasingly important to the population as a whole. Legislation, such as the Americans with Disabilities Act, requires that most employers provide reasonable accommodation for workers with disabilities. Section 508 of the Rehabilitation Act is also bringing accessibility issues to the forefront in government businesses and organizations receiving government funding.

Designing software that is usable for people with disabilities does not have to be time consuming or expensive. However, it is much easier if you include this in the planning and design process rather than attempting to add it after the completion of the software. Following the principles and guidelines in this guide will help you design software for most users. Often recommendations, such as the conservative use of color or sound, often benefit all users, not just those with disabilities. In addition, keep the following basic objectives in mind:

- Provide a customizable interface to accommodate a wide variety of user needs and preferences.
- Provide compatibility with accessibility utilities that users install.
- Avoid creating unnecessary barriers that make your software difficult or inaccessible to certain types of users.

Petitioner Exhibit 1031, Page 425

The following sections provide information on types of disabilities and additional recommendations about how to address the needs of customers with those disabilities.

Types of Disabilities

There are many types of disabilities, but they are often grouped into several broad categories. These include visual, hearing, physical movement, speech or language impairments, and cognitive and seizure disorders.

Visual Disabilities

Visual disabilities range from slightly reduced visual acuity to total blindness. Those with reduced visual acuity may only require that your software support larger text and graphics. For example, the system provides scalable fonts and controls to increase the size of text and graphics. To accommodate users who are blind or have severe impairments, make your software compatible with speech or Braille utilities, described later in this chapter.

Color blindness and other visual impairments may make it difficult for users to distinguish between certain color combinations. This is one reason why color is not recommended as the only means of conveying information. Always use color as an additive or enhancing property.

Hearing Disabilities

Users who are deaf or hard-of-hearing are generally unable to detect or interpret auditory output at normal or maximum volume levels. Avoiding the use of auditory output as the only means of communicating information is the best way to support users with this disability. Instead, use audio output only as a redundant, additive property or provide visual output as an option to supplement the audio information. For more information about supporting sound, see the section "Sound" earlier in this chapter.

Petitioner Exhibit 1031, Page 426

Physical Movement Disabilities

Some users have difficulty or are unable to perform certain physical tasks — for example, moving a mouse or simultaneously pressing two keys on the keyboard. Other individuals have a tendency to inadvertently strike multiple keys when targeting a single key. Consideration of physical ability is important not only for users with disabilities, but also for beginning users who need time to master all the motor skills necessary to interact with the interface. The best way to support these users is by supporting all your basic operations using simple keyboard and mouse interfaces.

Speech or Language Disabilities

Users with language disabilities, such as dyslexia, find it difficult to read or write. Spell- or grammar-check utilities can help children, users with writing impairments, and users with a different first language. Supporting accessibility tools and utilities designed for users who are blind can also help those with reading impairments. Most design issues affecting users with oral communication difficulties apply only to utilities specifically designed for speech input.

Cognitive Disabilities

Cognitive disabilities can take many forms, including perceptual differences and memory impairments. You can accommodate users with these disabilities by allowing them to modify or simplify your software's interface, such as supporting menu or dialog box customization. Similarly, using icons and graphics to illustrate objects and choices can be helpful for users with some types of cognitive impairments.

Seizure Disorders

Some users are sensitive to visual information that alternates its appearance or flashes at particular rates — often the greater the frequency, the greater the problem. However, there is no perfect flash rate. Therefore, base all modulating interfaces on the system's cursor blink rate. Because users can customize this value, a particular frequency can be avoided. If that is not practical, provide your own interface for changing the flash rate.

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.

Petitioner Exhibit 1031, Page 427

Types of Accessibility Aids

There are a number of accessibility aids to assist users with certain types of disabilities. To allow these users to effectively interact with your application, make certain it is compatible with these utilities. This section briefly describes the types of utilities and how they work.

One of the best ways to accommodate accessibility in your software's interface is to use standard Windows conventions wherever possible. Windows already provides a certain degree of customization for users and most accessibility aids work best with software that follows standard system conventions.

Screen Enlargement Utilities

Screen enlargers (also referred to as screen magnification utilities or large print programs) allow users to enlarge a portion of their screen. They effectively turn the computer monitor into a viewport showing only a portion of an enlarged virtual display. Users then use the mouse or keyboard to move this viewport to view different areas of the virtual display. Enlargers also attempt to track where users are working, following the input focus and the activation of windows, menus, and secondary windows, and can automatically move the viewport to the active area.

Screen Review Utilities

People who cannot use the visual information on the screen can interpret the information with the aid of a screen review utility (also referred to as a screen reader program or speech access utility). Screen review utilities take the displayed information on the screen and direct it through alternative media, such as synthesized speech or a refreshable Braille display. Because both of these media present only text information, the screen review utility must render other information on the screen as text; that is, determine the appropriate text labels or descriptions for graphical screen elements. They must also track users' activities to provide descriptive information about what the user is doing. These utilities often work by monitoring the system interfaces that support drawing on the screen. They build an off-screen database of the objects on the screen, their properties, and their spatial relationships. Some of this information is presented to

Petitioner Exhibit 1031, Page 428

users as the screen changes, and other information is maintained until users request it. Screen review utilities often include support for configuration files (also referred to as set files or profiles) for particular applications.

Voice Input Systems

Users who have difficulty typing can choose a voice input system (also referred to as a speech recognition program) to control software with their voice instead of a mouse and keyboard. Like screen reader utilities, voice input systems identify objects on the screen that users can manipulate. Users activate an object by speaking the label that identifies the object. Many of these utilities simulate keyboard interfaces, so if your software includes a keyboard interface, it can be adapted to take advantage of this form of input.

On-Screen Keyboards

Some individuals with physical disabilities cannot use a standard keyboard, but can use special devices designed to work with an on-screen keyboard. Switching devices display groups of commands displayed on the screen, and the user employs one or more switches to choose a selected group, then a command within the group. Another technique allows a user to use a special mouse or headpointer (a device that lets users manipulate the mouse pointer on the screen through head motion) to point to graphic images of keys displayed on the screen to generate keystroke input.

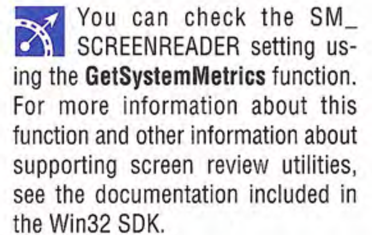
Keyboard Filters

Impaired physical abilities, such as erratic motion, tremors, or slow response, can sometimes be compensated by filtering out inappropriate keystrokes. The Windows Accessibility Options supports a wide range of keyboard filtering options. These are generally independent of the application with which users are interacting and therefore require no explicit support except for the standard system interfaces for keyboard input. However, users relying on these features may type slowly.

Petitioner Exhibit 1031, Page 429

Compatibility with Screen Review Utilities

You can use the following techniques to ensure software compatibility with screen review utilities. The system allows your application to determine whether the system has been configured to provide support for a screen review utility, allowing your software to enable or disable certain capabilities.



You can check the SM_SCREENREADER setting using the **GetSystemMetrics** function. For more information about this function and other information about supporting screen review utilities, see the documentation included in the Win32 SDK.

Controls

Use standard Windows controls wherever possible. Most of these have already been implemented to support screen review and voice input utilities. However, custom controls you create may not be usable by screen review utilities.

Always include a label for every control, even if you do not want the control's label to be visible. This applies regardless of whether you use standard controls or your own specialized controls, such as owner drawn controls or custom controls. If the control does not provide a label, you can create a label using a static text control.

Follow the normal layout conventions by placing the static text label before the control (above or to the left of the control). Also, set the keyboard TAB navigation order appropriately so that tabbing to a label navigates to the associated control it identifies instead of a label. To make certain that the label is recognized correctly, include a colon at the end of the label's text string, unless you are labeling a button, tab, or group box control. In cases where a label is not needed or would be visually distracting, provide the label but do not make it visible. Although the label is not visible, it is accessible to a screen review utility.

Text labels are also effective for choices within a control. For example, you can enhance menus or lists that display colors or line widths by including some form of text representation, as shown in Figure 14.1.

Petitioner Exhibit 1031, Page 430

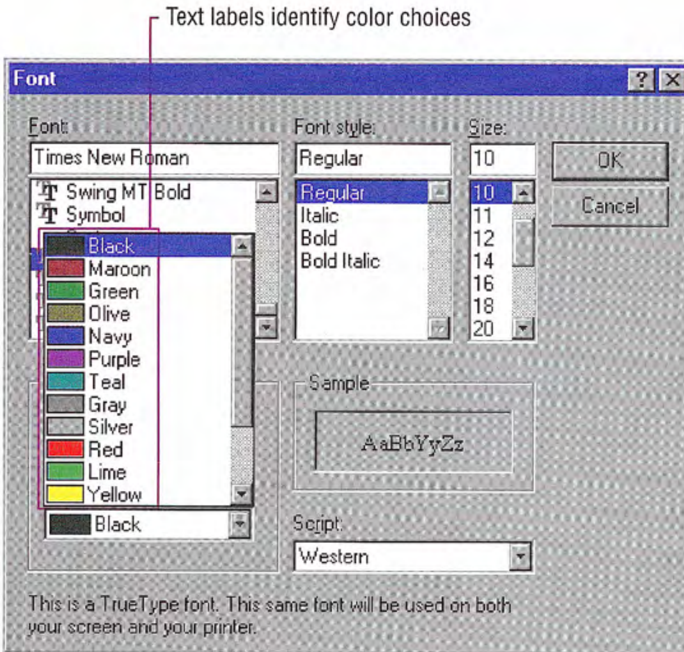


Figure 14.1 Using text to help identify choices

If providing a combined presentation is too difficult, offer users the choice between text and graphical representation, or choose one of them based on the system's screen review utility setting.

Text Output

Screen review utilities usually interpret text — including properties such as font, size, and face — that is displayed with standard system interfaces. However, text displayed as graphics (for example, bit-mapped text) is not accessible to a screen review utility. To make it accessible, your application can create an invisible text label and associate the graphical representation of text with it by drawing the text over the graphic with a null operator (NOP). Screen review utilities can read standard text representations in a metafile, so you can also use metafiles, instead of bitmap images, for graphics information that includes text.

Petitioner Exhibit 1031, Page 431

Graphics Output

Users with normal sight may be able to easily distinguish different elements of a graphic or pictorial information, such as a map or chart, even if they are drawn as a single image; however, a screen review utility must distinguish between different components. There are a number of ways to do this. Any of these methods can be omitted when the system's screen review setting is not set.

When using bitmap images, consider separately drawing each component that requires identification. If performance is an issue, combine the component images in an off-screen bitmap using separate drawing operations and then display the bitmap on the screen with a single operation. You can also draw multiple bitmap images with a single metafile.

Alternatively, you can redraw each component separately, or draw a separate image to identify each region using a null operator. This will not have an effect on the visible image, but allows a screen review utility to identify the region. You can also use this method to associate a text label with a graphic element.

When drawing graphics, use standard Windows drawing functions wherever possible. If you change an image directly — for example, clearing a bitmap by writing directly into its memory — a screen review utility will not be able to recognize the content change and will inappropriately describe it to users.

Icons and Windows

Accompany icons that represent objects with a text label (title) of the object's name. Use the system font and color for icon labels, and follow the system conventions for placement of the text relative to the icon. This allows a screen review utility to identify the object without special support.

Similarly, make certain that all your windows have titles. Even if the title is not visible, it is still available to access utilities. The more unique your window titles, the easier users can differentiate between them, especially when using a screen review utility. Using unique window class names is another way to provide for distinct window identification, but providing appropriate window titles is preferred.

Petitioner Exhibit 1031, Page 432

The User's Point of Focus

Many accessibility aids must follow where the user is working. For example, a screen review utility conveys to users where the input focus is; a screen enlarger pans its viewport to ensure that users' focus is always kept on the visible portion of the screen. Most utilities give users the ability to manually move the viewport, but this becomes a laborious process, especially if it has to be repeated each time the input focus moves.

When the system handles the move of the input focus, such as when the user selects a menu, navigates between controls in a dialog box, or activates a window, an accessibility utility can track the change. However, the utility may not detect when an application moves the input focus within its own window. Therefore, whenever possible, use standard system functions to place the input focus, such as the text insertion point. Even when you provide your own implementation of focus, you can use the system functions to indicate focus location without making the standard input focus indicator visible.

The **SetCaretPos** function is an example of a system function you can use to indicate focus location. For more information about this function, see the documentation included in the Win32 SDK.

Timing and Navigational Interfaces

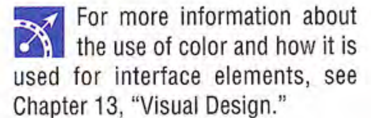
Some users read text or press keys very slowly and do not respond to events as quickly as the average user. Avoid displaying critical feedback or messages briefly and then automatically removing them because many users cannot read or respond to them. Similarly, limit your use of time-out based interfaces. If you do include a time-based interface, always provide a way for users to configure the time-out.

Also, avoid displaying or hiding information based on the movement of the pointer, unless it is part of a standard system interface (for example, tooltips). Although such techniques can benefit some users, they may not be available for those using accessibility utilities. If you do provide such support, consider making these features optional so that users can turn them on or off when a screen review utility is installed.

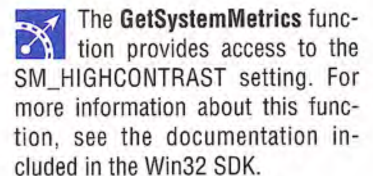
Similarly, you should avoid using general navigation to trigger operations, because users of accessibility aids may need to navigate through all controls. For example, basic TAB keyboard navigation in a dialog box should not activate actions associated with a control, such as setting a check box or carrying out a command button. However, navigation can be used to facilitate further user interaction, such as validating user input or opening a drop-down control.

Petitioner Exhibit 1031, Page 433

Color

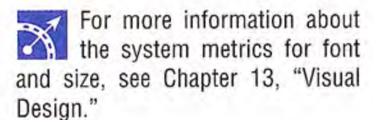

For more information about the use of color and how it is used for interface elements, see Chapter 13, "Visual Design."

Base the color properties of your interface elements on the system colors for window components, rather than defining specific colors. Remember to use appropriate foreground and background color combinations. If the foreground of an element is rendered with the button text color, use the button face color as its background rather than the window background color. If the system does not provide standard color settings that can be applied to some elements, you can include your own interface that allows users to customize colors. In addition, you can provide graphical patterns as an optional substitute for colors as a way to distinguish information.


The **GetSystemMetrics** function provides access to the SM_HIGHCONTRAST setting. For more information about this function, see the documentation included in the Win32 SDK.

The system also provides a global setting called High Contrast Mode that users can set through the Windows Accessibility Options. The setting provides contrasting color settings for foreground and background visual elements. Your application should check for this setting's status when it starts, and whenever it receives notification of system setting changes. When set, adjust your interface colors based on those set for the high contrast color scheme. In addition, whenever High Contrast Mode is set, hide any images that are drawn behind text (for example, watermarks or logos) to maintain the legibility of the information on the screen. You can also display monochrome versions of bitmaps and icons using the appropriate foreground color.

Scalability


For more information about the system metrics for font and size, see Chapter 13, "Visual Design."

Another important way to provide for visual accessibility is to allow for the scalability of screen elements. Sometimes, this simply means allowing users to change the font for the display of information. The system allows users to change the size and font of standard Windows components. You should use these same metrics for appropriately adjusting the size of other visual information you provide. For your own custom elements, you can provide scaling by including a TrueType font or metafiles for your graphics images.

It may also useful to provide scaling features within your application. For example, many application provide a "Zoom" command that scales the presentation of the information displayed in a window, or other commands that make the presentation of information easier to read. You may need to add scroll bars if the scaled information exceeds the current size of the window.

Petitioner Exhibit 1031, Page 434

Keyboard and Mouse Interface

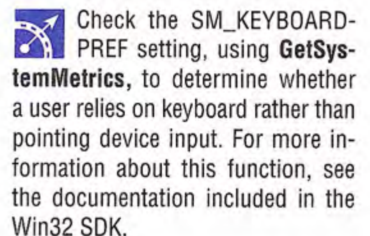
Providing a good keyboard interface is an important step in accessibility because it affects users with a wide range of disabilities. For example, a keyboard interface may be the only option for users who are blind or use voice input utilities, and those who cannot use a mouse. The Windows Accessibility Options often compensate for users with disabilities related to keyboard interaction; however, it is more difficult to compensate for problems related to pointing device input.

You should follow the conventions for keyboard navigation techniques presented in this guide. For specialized interfaces within your software, model your keyboard interface on conventions that are familiar and appropriate for that context. Where they apply, use the standard control conventions as a guide for your defining interaction. For example, support TAB and SHIFT+TAB key and access keys to support navigation to controls.

Make certain the user can navigate to all objects. Avoid relying only on navigational design that requires the user to understand the spatial relationship between objects. Accessibility utilities may not be able to convey such relationships.

Providing a well-designed mouse interface is also important. Pointing devices may be more efficient than keyboards for some users. When designing the interface for pointing input, avoid making basic functions available only through multiple clicking, drag and drop manipulation, and keyboard-modified mouse actions. Such actions are best considered shortcut techniques for more advanced users. Make basic functions available through single click techniques.

The system also allows your application to determine when the user relies on the keyboard, rather than pointing device input. You can use this to present special keyboard interfaces that might otherwise be hidden.



Check the SM_KEYBOARDPREF setting, using **GetSystemMetrics,** to determine whether a user relies on keyboard rather than pointing device input. For more information about this function, see the documentation included in the Win32 SDK.

Where possible, avoid making the implementation of basic functions dependent on a particular device. This is critical for supporting users with physical disabilities and users who may not wish to use or install a particular device.

Petitioner Exhibit 1031, Page 435

Documentation, Packaging, and Support

Although this guide focuses primarily on the design of the user interface, a design that provides for accessibility needs to take into consideration other aspects of a product. For example, consider the documentation needs of your users. For users who have difficulty reading or handling printed material, provide online documentation for your product. If the documentation or installation instructions are not available online, you can provide documentation separately in alternative formats, such as ASCII text, large print, Braille, or audio tape format. Organizations that can help you produce and distribute such documentation are listed in the accessibility section of the Bibliography of this guide.

When possible, choose a format and binding for your documentation that makes it accessible for users with disabilities. As in the interface, information in color should be a redundant form of communication. Bindings that allow a book to lie flat are usually better for users with limited dexterity.

Packaging is also important because many users with limited dexterity can have difficulty opening packages. Consider including an easy-opening overlap or tab that helps users remove shrink-wrapping.

Finally, although support is important for all users, it is difficult for users with hearing impairments to use standard support lines. Consider making these services available to customers using text telephones (also referred to as "TT" or "TDD"). You can also provide support through public bulletin boards or other networking services.

Usability Testing

Just as it is important to test the general usability of your software, it is a good idea to test how well it provides for accessibility. There are a variety of ways of doing this. One way is to include users with disabilities in your prerelease or usability test activities. In addition, you can establish a working relationship with companies that provide accessibility aids. Information about accessibility vendors or potential test sites is included in the Bibliography.

Petitioner Exhibit 1031, Page 436

You can also try running your software in a fashion similar to that used by a person with disabilities. Try some of the following ideas for testing:

- Use the Windows Accessibility Options and set your display to a high contrast scheme, such as white text on a black background. Are there any portions of your software that become invisible or hard to use or recognize?

- Try using your software for a week without using a mouse. Are there operations that you cannot perform? Was anything especially awkward?

- Increase the size of the default system fonts. Does your software still look good? Does your software fonts appropriately adjust to match the new system font?

Internationalization

To successfully compete in international markets, your software must easily accommodate differences in language, culture, and hardware. This section does not cover every aspect of preparing software for the international market, but it does summarize some of the key design issues.

For more information about the technical details for localizing your application, see the documentation included in the Win32 SDK.

The process of translating and adapting a software product for use in a different country is called *localization*. Like any part of the interface, include international considerations early in the design and development process. In addition to adapting screen information and documentation for international use, Help files, scenarios, templates, models, and sample files should all be a part of your localization planning.

Language is not the only relevant factor when localizing an interface. Several countries can share a common language but have different conventions for expressing information. In addition, some countries can share a language but use a different keyboard convention.

Petitioner Exhibit 1031, Page 437

A more subtle factor to consider when preparing software for international markets is cultural differences. For example, users in the U.S. may recognize a rounded mail box with a flag on the side as an icon for a mail program, but this image may not be recognized by users in other countries. Sounds and their associated meanings may also vary from country to country.

It is helpful to create a supplemental document for your localization team that covers the terms and other translatable elements your software uses, and describes where they occur. Documenting changes between versions saves time in preparing new releases. Appendix E of this guide provides recommended translations of many words used in the Windows interface.

Text

A major aspect of localizing an interface involves translating the text used by the software in its title bars, menus and other controls, messages, and some registry entries. To make localization easier, store interface text as resources in your application's resource file rather than including it in the source code of the application. Remember to also translate menu commands your application stores for its file types in the system registry.

Translation is a challenging task. Each foreign language has its own syntax and grammar. Following are some general guidelines to keep in mind for translation:

- Do not assume that a word always appears at the same location in a sentence, that word order is always the same, that sentences or words always have the same length, or that nouns, adjectives, and verbs always keep the same form.

- Avoid using vague words that can have several meanings in different contexts.

- Avoid colloquialisms, jargon, acronyms, and abbreviations.

- Use good grammar. Translation is a difficult enough task without a translator having to deal with poor grammar.

Petitioner Exhibit 1031, Page 438

- Avoid dynamic, or run-time, concatenation of different strings to form new strings — for example, composing messages by combining frequently used strings. An exception is the construction of filenames and names of paths.

- Avoid hard coding filenames in a binary file. Filenames may need to be translated.

- Avoid including text in images and icons. Doing so requires that these also be translated.

Translation of interface text from English to other languages often increases the length of text by 30 percent or more. In some extreme cases, the character count can increase by more than 100 percent; for example, the word "move" becomes "verschieben" in German. Accordingly, if the amount of the space for displaying text is strictly limited, as in a status bar, restrict the length of the English interface text to approximately one half of the available space. In contexts that allow more flexibility, such as dialog boxes and property sheets, allow 30 percent for text expansion in the interface design. Message text in message boxes, however, should allow for text expansion of about 100 percent. Avoid having your software rely on the position of text in a control or window because translation may require movement of the text.

Expansion due to translation affects other aspects of your product. A localized version is likely to affect file sizes, potentially changing the layout of your installation disks and setup software.

Translation is not always a one-to-one correspondence. A single word in English can have multiple translations in another language. Adjectives and articles sometimes change spelling according to the gender of the nouns they modify. Therefore, be careful when reusing a string in multiple places. Similarly, several English words may have only a single meaning in another language. This is particularly important when creating keywords for the Help index for your software.

Graphics

It is best to review the proposed graphics for international applicability early in your design cycle. Localizing graphics can be a time consuming process.

Petitioner Exhibit 1031, Page 439

Although graphics communicate more universally than text, graphical aspects of your software — especially icons and toolbar button images — may also need to be revised to address an international audience. For example, a toolbar image that includes a magic wand to represent access to a wizard interface does not have meaning in many European countries and requires a different image.

When possible, choose generic images and glyphs. Even if you can create custom designs for each language, having different images for different languages can confuse users who work with more than one language version.

Many symbols with a strong meaning in one culture do not have any meaning in another. For example, many symbols for U.S. holidays and seasons are not shared around the world. Importantly, some symbols can be offensive in some cultures (for example, the open palm commonly used at U.S. crosswalk signals is offensive in some countries). Some metaphors also may not apply in all languages.

Keyboards

International keyboards also differ from those used in the U.S. Avoid using punctuation character keys as shortcut keys because they are not always found on international keyboards or easily produced by the user. Remember too, that what seems like an effective shortcut because of its mnemonic association (for example, CTRL+B for Bold) can warrant a change to fit a particular language. Similarly, macros or other utilities that invoke menus or commands based on access keys are not likely to work in an international version, because the command names on which the access keys are based differ.

Keys do not always occupy the same positions on all international keyboards. Even when they do, the interpretation of the unmodified keystroke can be different. For example, on U.S. keyboards, SHIFT+8 results in an asterisk character. However, on French keyboards, it generates the number 8. Similarly, avoid using CTRL+ALT combinations, because the system interprets this combination for some language versions as the ALTGR key, which generates some alphanumeric characters. Similarly, avoid using the ALT key as a modifier because it is the primary keyboard interface for accessing menus and controls. In addition, the system uses many specialized versions for special input. For example, ALT+~ invokes special input

Petitioner Exhibit 1031, Page 440

editors in Far East versions of Windows. For text fields, pressing ALT+*number* enters characters in the upper range of a character set. Similarly, avoid using the following characters when assigning the following shortcut keys.

@ £ $ { } [ ] \ ~ | ^ ' < >

Character Sets

Some international countries require support for different character sets (sometimes referred to as *code pages*). The system provides a standard interface for supporting multiple character sets and sort tables. Use these interfaces wherever possible for sorting and case conversion. In addition, consider the following guidelines:

The **SystemParametersInfo** function allows you to determine the current keyboard configuration. For more information about this function, see the documentation included in the Win32 SDK.

- Do not assume that the character set is U.S. ANSI. Many ANSI character sets are available. For example, the Russian version of Windows 95 uses the Cyrillic ANSI character set which is different than the U.S. ANSI set.
- Use the system functions for supporting font selection (such as the common font dialog box).
- Always save the character set with font names in documents.

Formats

Different countries often use substantially different formats for dates, time, money, measurements, and telephone numbers. This collection of language-related user preferences are referred to as a *locale*. Designing your software to accommodate international audiences requires supporting these different formats.

Windows provides a standard means for inquiring what the default format is and also allows the user to change those properties. Your software can allow the user to change formats, but restrict these changes to your application or document type, rather than affecting the system defaults. Table 14.1 lists the most common format categories.

For more information about the functions that provide access to the current locale formats, see the documentation included in the Win32 SDK.

Petitioner Exhibit 1031, Page 441

Table 14.1 Formats for International Software

Category	Format considerations
Date	Order, separator, long or short formats, and leading zero
Time	Separator and cycle (12-hour vs. 24-hour), leading zero
Physical quantity	Metric vs. English measurement system
Currency	Symbol and format (for example, trailing vs. preceding symbol)
Separators	List, decimal, and thousandths separator
Telephone numbers	Separators for area codes and exchanges
Calendar	Calendar used and starting day of the week
Addresses	Order and postal code format
Paper sizes	U.S. vs. European paper and envelope sizes

Layout

For layout of controls or other elements in a window, it is important to consider alignment in addition to expansion of text labels. In Hebrew and Arabic countries, information is written right to left. So when localizing for these countries, reverse your U.S. presentation.

Some languages include diacritical marks that distinguish particular characters. Fonts associated with these characters can require additional spacing.

In addition, do not place information or controls into the title bar area. This is where Windows places special user controls for configurations that support multiple languages.

References to Unsupported Features

Avoid confusing your international users by leaving in references to features that do not exist in their language version. Adapt the interface appropriately for features that do not apply. For example, some language versions may not include a grammar checker or support for bar codes on envelopes. Remove references to features such as menus, dialog boxes, and Help files from the installation program.

Petitioner Exhibit 1031, Page 442

Network Computing

Windows provides an environment that allows the user to communicate and share information across the network. When designing your software, consider the special needs that working in such an environment requires.

Conceptually, the network is an expansion of the user's local space. The interface for accessing objects from the network should not differ significantly from or be more complex than the user's desktop.

Leverage System Support

When designing for network access, support standard conventions and interfaces, including the following:

- Use universal naming convention (UNC) paths to refer to objects stored in the file system. This convention provides transparent access to objects on the network.
- Use system-supported user identification that allows you to determine access without including your own password interface.
- Adjust window sizes and positions based on the local screen properties of the user.
- Avoid assuming the presence of a local hard disk. It is possible that some of your users work with diskless workstations.

Client-Server Applications

Users operating on a network may wish to run your application from a network server. For applications that store no state information, no special support is required. However, if your application stores state information, design your application with a server set of components and a client set of components. The server components include the main executable files, dynamic link libraries, and any other files that need to be shared across the network. The client components consist of the components of the application that are specific to the user, including local registry information and local files that provide the user with access to the server components.

For information about installing the client and server components of your application, see Chapter 10, "Integrating with the System."

Petitioner Exhibit 1031, Page 443

Shared Data Files

When storing a file in the shared space of the network, it should be readily accessible to all users, so design the file to be opened multiple times. The granularity of concurrent access depends on the file type; that is, for some files you may only support concurrent access by word, paragraph, section, page, and so on. Provide clear visual cues as to what information can be changed and what cannot. Where multiple access is not easily supported, provide users with the option to open a copy of the file, if the original is already open.

Record Processing

Record processing or transaction-based applications may require somewhat different structuring than the typical productivity application. For example, rather than opening and saving discrete files, the interface for such applications focuses on accessing and presenting data as records through multiple views, forms, and reports. One of the distinguishing and most important design aspects of record-processing applications is the definition of how the data records are structured. This dictates what information can be stored and in what format.

However, you can apply much of the information in this guide to record-oriented applications. For example, the basic principles of design and methodology are just as applicable as they are for individual file-oriented applications. You can also apply the guide's conventions for input, navigation, and layout when designing forms and report designs. Similarly, your can apply other secondary window conventions for data-entry design, including the following:

- Provide reasonable default values for fields.
- Use the appropriate controls. For example, use drop-down list boxes instead of long lists of option buttons.
- Distinguish text entry fields from read-only text fields.

Petitioner Exhibit 1031, Page 444

- Design for logical and smooth user navigation. Order fields as the user needs to move through them. Auto-exit text boxes are often good for input of predefined data formats, such as time or currency inputs.

- Provide data validation as close to the site of data entry as possible. You can use input masks to restrict data to specific types or list box controls to restrict the range of input choices.

Telephony

Windows provides support for creating applications with telephone communications, or *telephony*, services. Those services include the Assisted Telephony services for adding minimal, but useful telephonic functionality to applications, and the full Telephony API, for implementing full telephonic applications.

For more information about creating applications using the Microsoft Windows Telephony API (TAPI), see the documentation included in the Win32 SDK.

You should consider the following guidelines when developing telephony applications:

- Provide separate fields for users to enter country code, area code, and a local number. You may use auto-exit style navigation to facilitate the number entry. You can also use a drop-down list box to allow users to select a country code. The system provides support for listing available codes.

- Provide access to the TAPI Dialing Properties property sheet window wherever a user enters a phone number. This window provides a consistent and easy interface for users.

- Use the modem configuration interfaces provided by the system. If the user has not installed a modem, run the Windows TAPI modem installation wizard.

Petitioner Exhibit 1031, Page 445

Microsoft Exchange

Microsoft Exchange is the standard Windows interface for email, voice mail, FAX, and other communication media. Applications interact with Microsoft Exchange by using the Messaging API (MAPI) and support services and components.

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

Microsoft Exchange allows you to create support for an information service. An information service is a utility that enables messaging applications to send and receive messages and files, store items in an information store, obtain user addressing information, or any combination of these functions.

Coexisting with Other Information Services

Microsoft Exchange is designed to simultaneously support different information services. Therefore when designing an information service, avoid:

- Initiating lengthy operations.
- Assuming exclusive use of key hardware resources, such as communications (COMM) ports and modems.
- Adding menu commands that are incompatible with other services.

Adding Menu Items and Toolbar Buttons

Microsoft Exchange allows you to add menu items and toolbar buttons to the main viewer window. Follow the recommendations in this guide for defining menu and toolbar entries. In addition, where possible, define your menu items and toolbar entries (or their tooltips) in a way that allows the user to clearly associate the functionality with a specific information service.

Petitioner Exhibit 1031, Page 446

Supporting Connections

When the user selects an information service that you support, provide the user with a dialog box to confirm the choice and allow the user access to configuration properties. Because simultaneous services run at the same time, clearly identify the service. At the top of the window, display the icon and name of the service. You can include an option to not display the dialog box.

Installing Information Services

Microsoft Exchange includes a special wizard for installation of information services. You can support this wizard to allow the user to easily install your service.

The system also provides profiles and files that define which services are available to users when they log on. When the user installs your service, ask the user which profile they would like to include with your service, such as their default profile.

Petitioner Exhibit 1031, Page 447

Appendixes

Petitioner Exhibit 1031, Page 449