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.

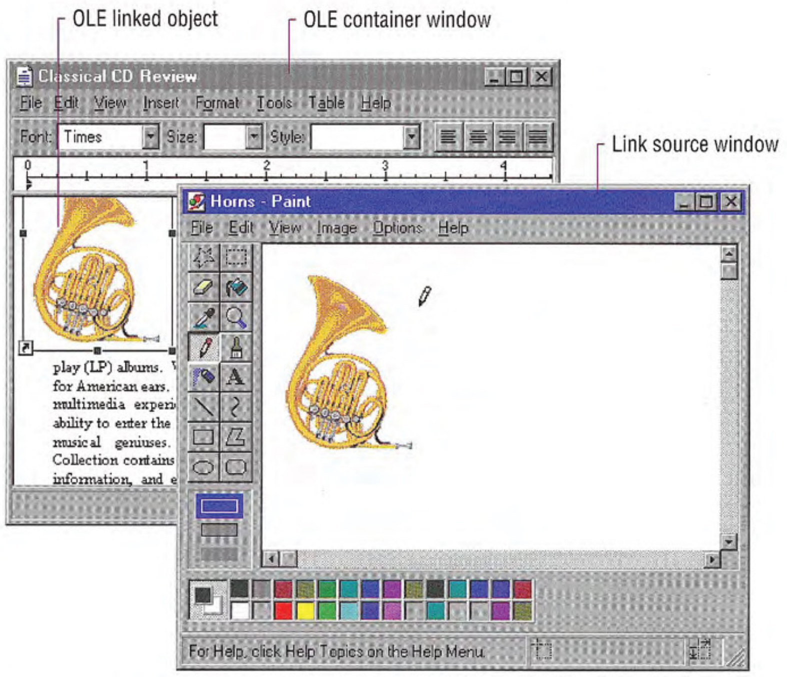
While an OLE embedded object is open, it is still a functioning member of its container. It can still be selected or unselected, and can respond to appropriate container commands. At any time, the user may open any number of OLE embedded objects. When the user closes its container window, deactivate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

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.


OLE linked object
OLE container window
Link source window
Classical CD Review
File Edit View Insert Format Tools Table Help
Font: Times Size: Style:
Horns - Paint
File Edit View Image Options Help
For Help, click Help Topics on the Help Menu.

play (LP) albums.
for American ears.
multimedia experi
ability to enter the
musical geniuses.
Collection contains
information, and e

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.

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.

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.

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.

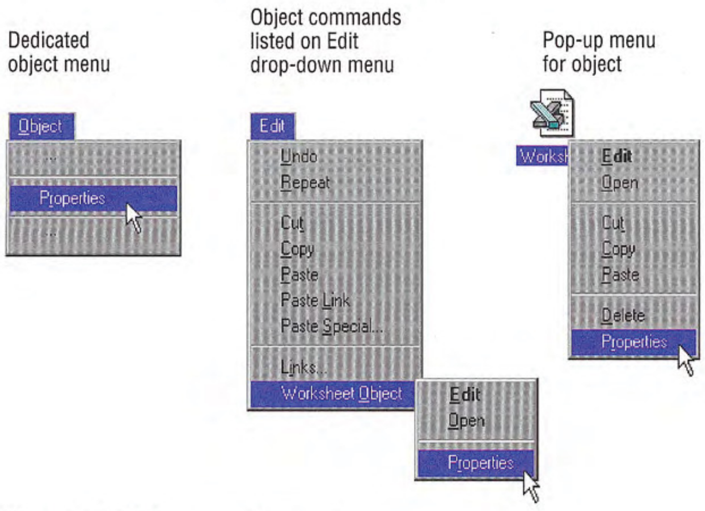
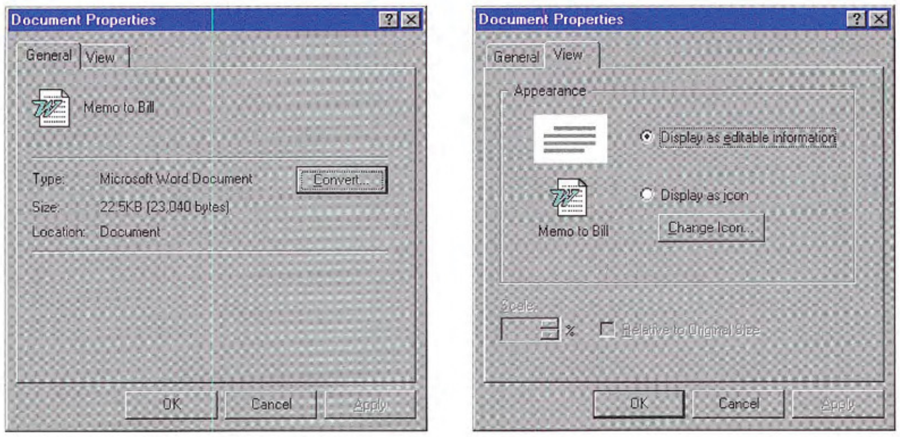


Figure 11.29 The Properties command

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.


Document Properties

General | View

Memo to Bill

Type: Microsoft Word Document

Size: 22.5KB (23,040 bytes)

Location: Document

Appearance

[✓ Display as editable information]

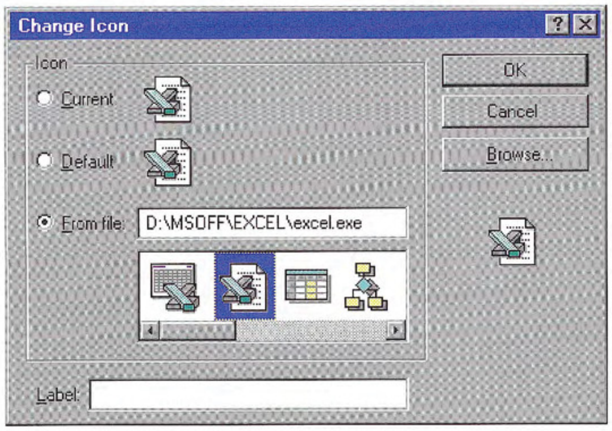
Display as icon

Scale: %

Relative to Original Size

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.

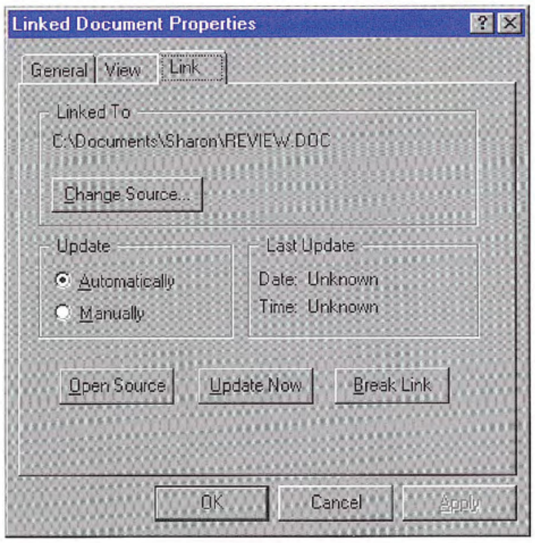

Change Icon

Icon
- Current
- Default
- [✓ From file:] D:\MSOFF\EXCEL\excel.exe

Label:

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.


Linked Document Properties

General | View | Link

Linked To
C:\Documents\Sharon\REVIEW.DOC

Update
- [✓ Automatically]
- Manually

Last Update
Date: Unknown
Time: Unknown

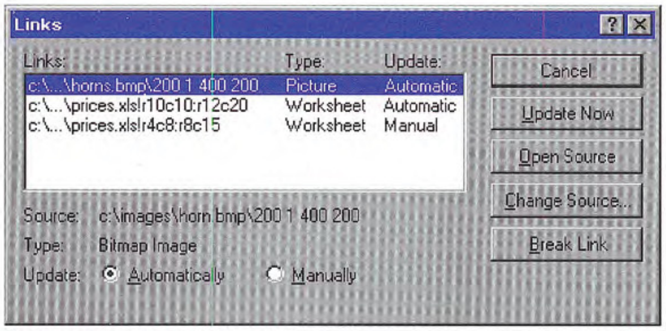
Figure 11.32 The Link page for the property sheet of an OLE linked object

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.



Links:	Type:	Update:
c:\...\horns.bmp\200 1 400 200	Picture	Automatic
c:\...\prices.xls!r10c10:r12c20	Worksheet	Automatic
c:\...\prices.xls!r4c8:r8c15	Worksheet	Manual

Source: c:\images\horn.bmp\200 1 400 200
Type: Bitmap Image
Update: [✓ Automatically] Manually

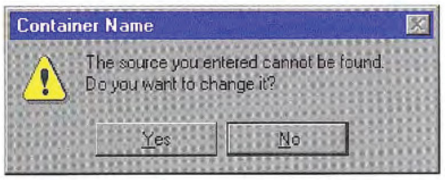
Figure 11.33 The Links dialog box

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.

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.

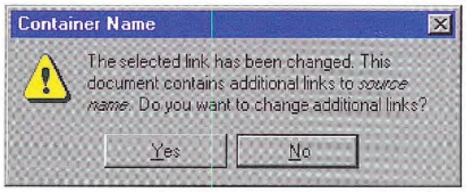

Container Name

The source you entered cannot be found.
Do you want to change it?

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.

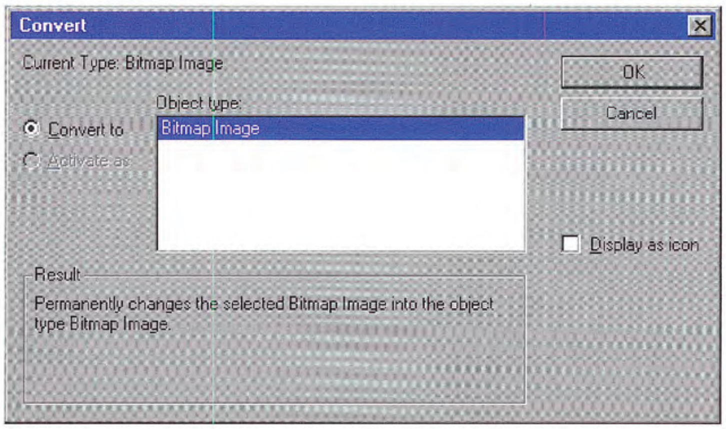


The selected link has been changed. This document contains additional links to *source name*. Do you want to change additional links?

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.



Current Type: Bitmap Image

Object type: Bitmap Image

Permanently changes the selected Bitmap Image into the object type Bitmap Image.

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.

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.

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.

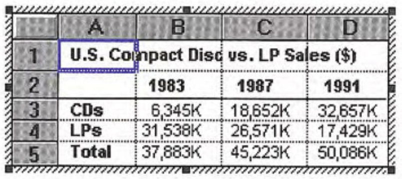
Disable the Convert option for a linked object because conversion for a link must occur on the link source. Also disable Activate As option if no types are registered for alternative activation. If the user can neither convert nor change the activation association, disable the Convert command that displays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

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.

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.

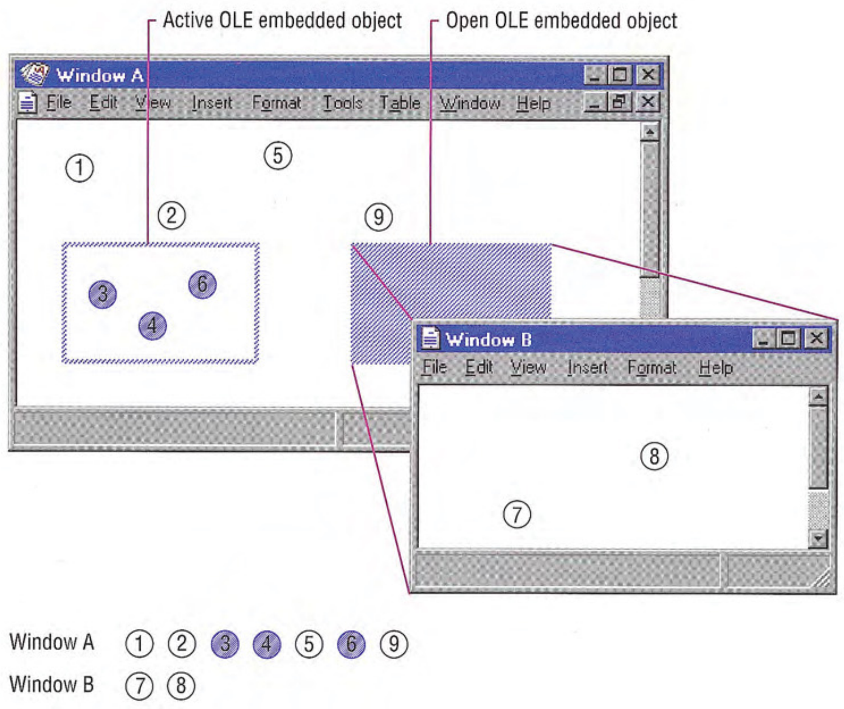

Active OLE embedded object
Open OLE embedded object
Window A
File Edit View Insert Format Tools Table Window Help
Window B
File Edit View Insert Format Help
Window A ① ② ③ ④ ⑤ ⑥ ⑨
Window B ⑦ ⑧

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 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.

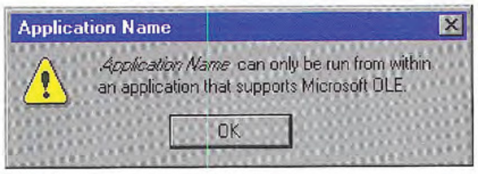
The system supplies most of the message boxes described in this chapter. For more information about how to support these, see the OLE documentation included in the Win32 SDK.

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.

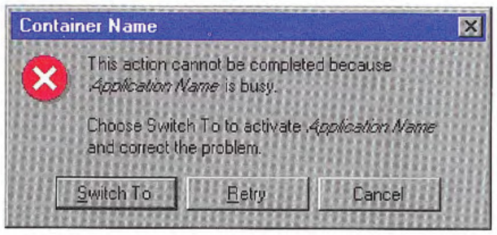

Application Name

Application Name can only be run from within an application that supports Microsoft OLE.

Figure 11.39 Object's application cannot be run standalone message

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.



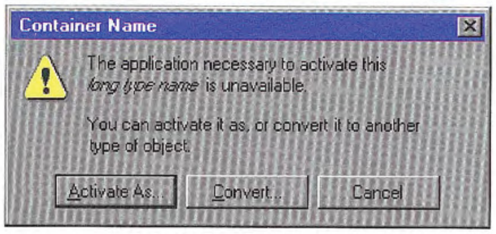
This action cannot be completed because *Application Name* is busy.

Choose Switch To to activate *Application Name* and correct the problem.

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.



The application necessary to activate this *long type name* is unavailable.

You can activate it as, or convert it to another type of object.

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.

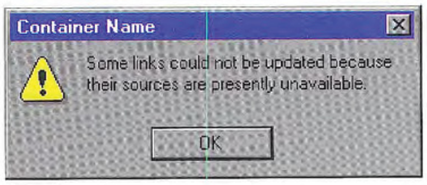
If a container supports inside-out activation for an object, display this message when the user tries to interact with that object, not when its container is opened. This avoids the display of the message to the user who only intends to view the content.

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.



Some links could not be updated because their sources are presently unavailable.

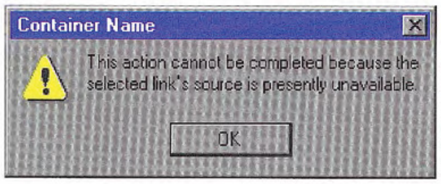
Figure 11.42 Link source files are unavailable message

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

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.


Container Name

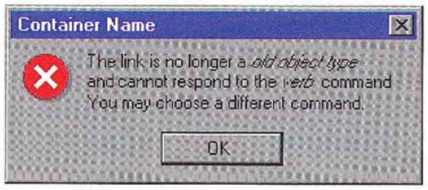
This action cannot be completed because the selected link's source is presently unavailable.

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.


Container Name

The link is no longer a *old object type* and cannot respond to the *verb* command You may choose a different command.

Figure 11.44 Link source's type has changed message

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.

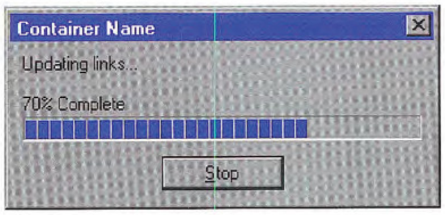


Figure 11.45 Progress indicator while links update message

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.

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.

CHAPTER 12

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.

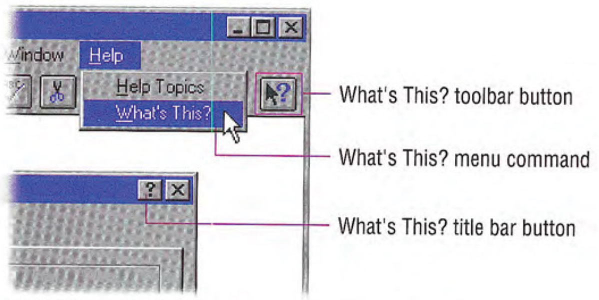

Window
Help
Help Topics
What's This?
What's This? toolbar button
What's This? menu command
What's This? title bar button

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.

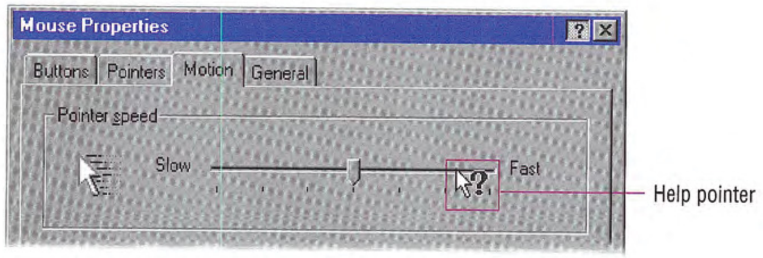
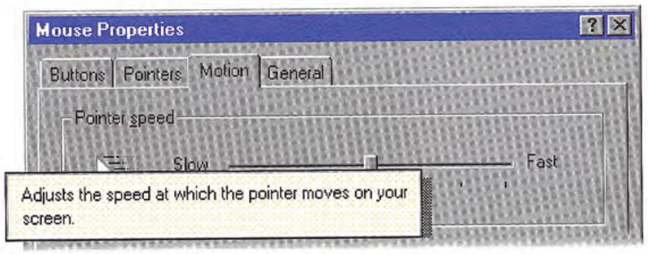


Figure 12.2 A context-sensitive Help pointer

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.



Adjusts the speed at which the pointer moves on your screen.

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.

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.

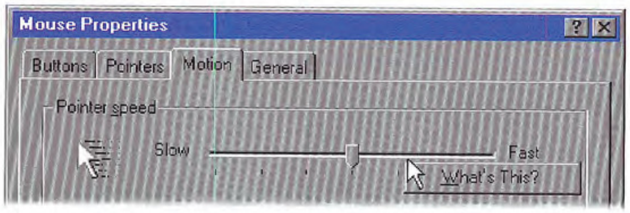


Figure 12.4 A pop-up menu for a control

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.

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."

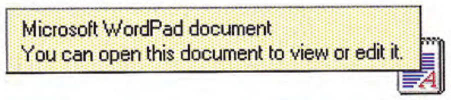

Microsoft WordPad document
You can open this document to view or edit it.

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.

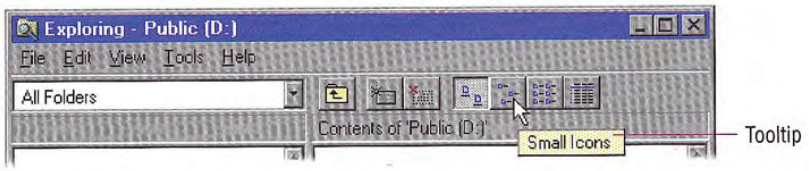


Figure 12.6 A tooltip for a toolbar button

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.

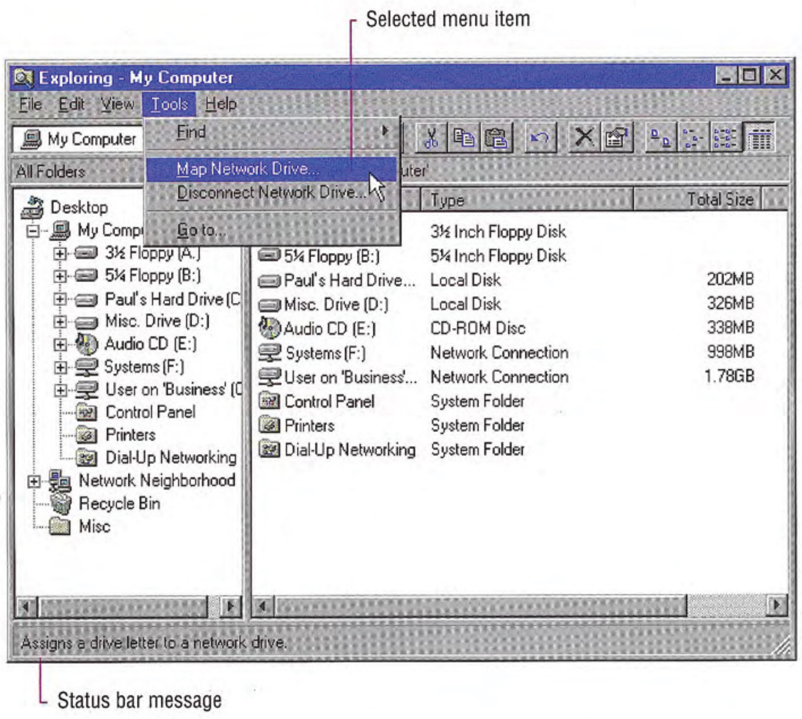
If you use the standard toolbar control, the system automatically provides support for tootips. It also includes a tooltip control that can be used in other contexts. If you create your own tooltip controls, make them consistent with the system-supplied controls.

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.



	Type	Total Size
	3½ Inch Floppy Disk	
5¼ Floppy (B:)	5¼ Inch Floppy Disk	
Paul's Hard Drive...	Local Disk	202MB
Misc. Drive (D:)	Local Disk	326MB
Audio CD (E:)	CD-ROM Disc	338MB
Systems (F:)	Network Connection	998MB
User on 'Business'...	Network Connection	1.78GB
Control Panel	System Folder	
Printers	System Folder	
Dial-Up Networking	System Folder	

Assigns a drive letter to a network drive.

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.

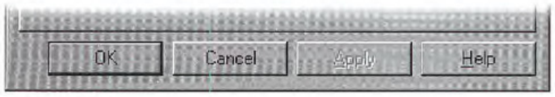


Figure 12.8 A Help button in a secondary window

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.

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.

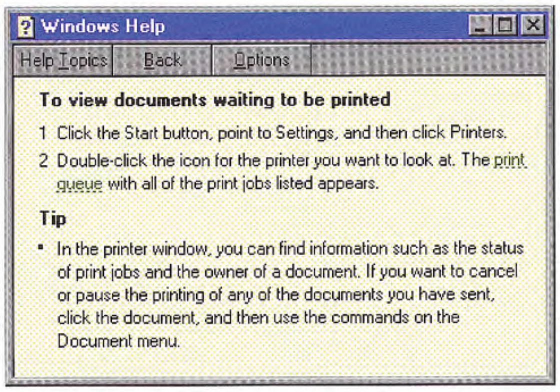
Task Topic Windows

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.



To view documents waiting to be printed

1 Click the Start button, point to Settings, and then click Printers.

2 Double-click the icon for the printer you want to look at. The print queue with all of the print jobs listed appears.

Tip

- In the printer window, you can find information such as the status of print jobs and the owner of a document. If you want to cancel or pause the printing of any of the documents you have sent, click the document, and then use the commands on the Document menu.

Figure 12.9 A window for a task Help topic

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.

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.

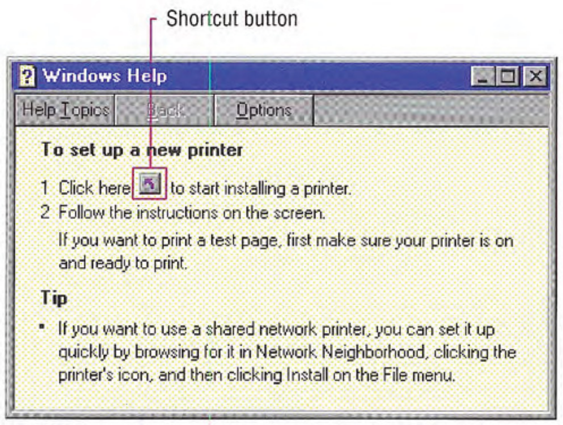
If you keep the procedure to four or fewer steps, the user will not need to scroll the window. Avoid introductory, conceptual, or reference material in the procedure.

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.



To set up a new printer

1 Click here to start installing a printer.
2 Follow the instructions on the screen.
If you want to print a test page, first make sure your printer is on and ready to print.

Tip

- If you want to use a shared network printer, you can set it up quickly by browsing for it in Network Neighborhood, clicking the printer's icon, and then clicking Install on the File menu.

Figure 12.10 A task Help topic with a shortcut button

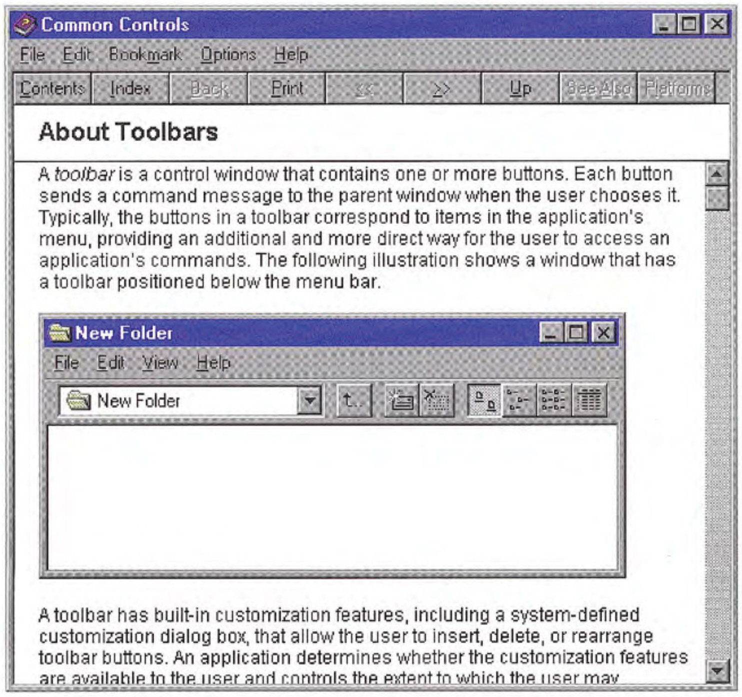
Shortcut buttons not only provide efficiency for the user, they also reduce the amount of information you may need to present and the user needs to read. However, you need not use the buttons as a substitute for doing the task or a specific step in the task; particularly if you want to support the user 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.

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.


Common Controls
File Edit Bookmark Options Help
Contents Index Back Print << >> Up See Also Platforms
New Folder
File Edit View Help
New Folder

About Toolbars

A *toolbar* is a control window that contains one or more buttons. Each button sends a command message to the parent window when the user chooses it. Typically, the buttons in a toolbar correspond to items in the application's menu, providing an additional and more direct way for the user to access an application's commands. The following illustration shows a window that has a toolbar positioned below the menu bar.

A toolbar has built-in customization features, including a system-defined customization dialog box, that allow the user to insert, delete, or rearrange toolbar buttons. An application determines whether the customization features are available to the user and controls the extent to which the user may

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.

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.

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.

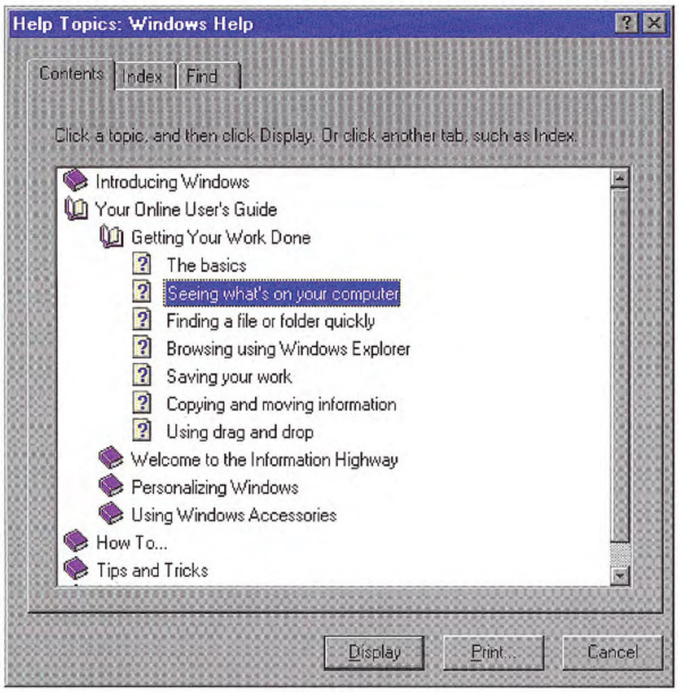
In addition, provide a Help Topics button in the toolbar of a task or reference Help topic window. When the user chooses this button, display the Help Topics browser window with the last page the user accessed. If you prefer, provide Contents, Index, and Find Topic buttons for direct access to a specific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


Help Topics: Windows Help

Click a topic, and then click Display. Or click another tab, such as Index.

- Introducing Windows
- Your Online User's Guide
  - Getting Your Work Done
    - The basics
    - Seeing what's on your computer
    - Finding a file or folder quickly
    - Browsing using Windows Explorer
    - Saving your work
    - Copying and moving information
    - Using drag and drop
  - Welcome to the Information Highway
  - Personalizing Windows
  - Using Windows Accessories
- How To...
- Tips and Tricks

Figure 12.12 The Contents page of the Help topics browser

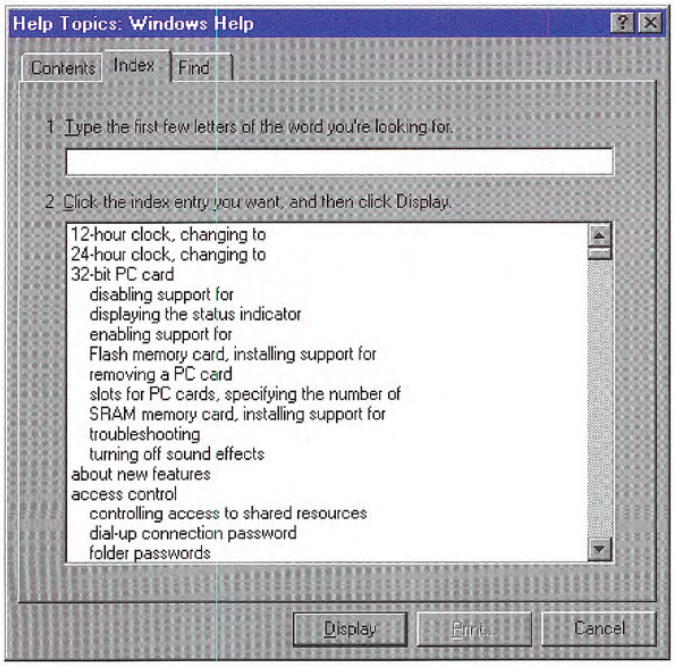
The buttons at the bottom of the page allow the user to open or close a "book" of topics and display a particular topic. The Print button prints either a "book" of topics or a specific topic depending on which the user selects. The outline also supports direct interaction, such as double-clicking, for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

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.



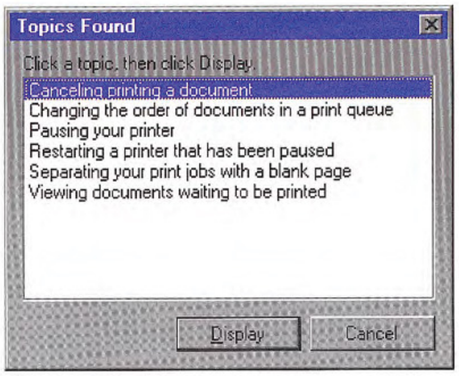
1 Type the first few letters of the word you're looking for.

2 Click the index entry you want, and then click Display.

12-hour clock, changing to
24-hour clock, changing to
32-bit PC card
    disabling support for
    displaying the status indicator
    enabling support for
    Flash memory card, installing support for
    removing a PC card
    slots for PC cards, specifying the number of
    SRAM memory card, installing support for
    troubleshooting
    turning off sound effects
about new features
access control
    controlling access to shared resources
    dial-up connection password
    folder passwords

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.



Click a topic, then click Display.

Canceling printing a document
Changing the order of documents in a print queue
Pausing your printer
Restarting a printer that has been paused
Separating your print jobs with a blank page
Viewing documents waiting to be printed

Figure 12.14 The Help topics window

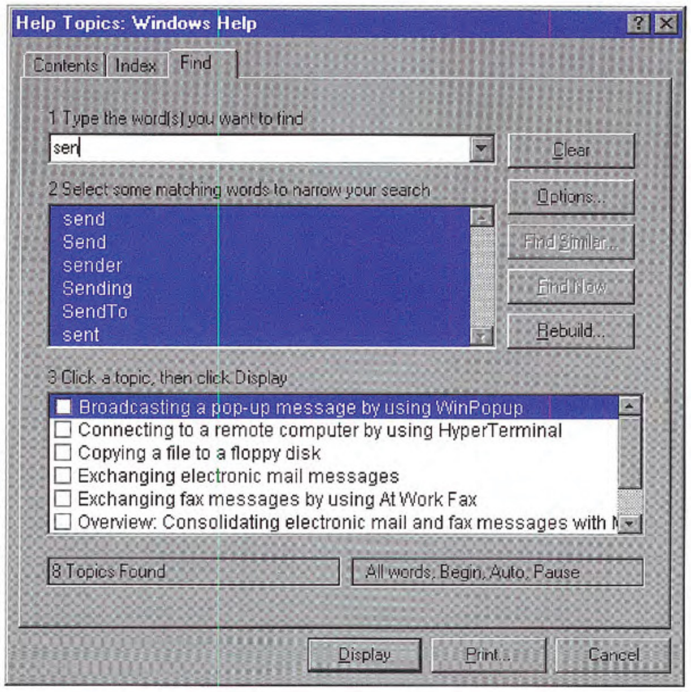
Guidelines for Writing Help Index Keywords

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.


Help Topics: Windows Help

Contents | Index | Find

1 Type the word(s) you want to find

sen

2 Select some matching words to narrow your search

send
Send
sender
Sending
SendTo
sent

3 Click a topic, then click Display

- Broadcasting a pop-up message by using WinPopup
- Connecting to a remote computer by using HyperTerminal
- Copying a file to a floppy disk
- Exchanging electronic mail messages
- Exchanging fax messages by using At Work Fax
- Overview: Consolidating electronic mail and fax messages with M

8 Topics Found

All words, Begin, Auto, Pause

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.

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.

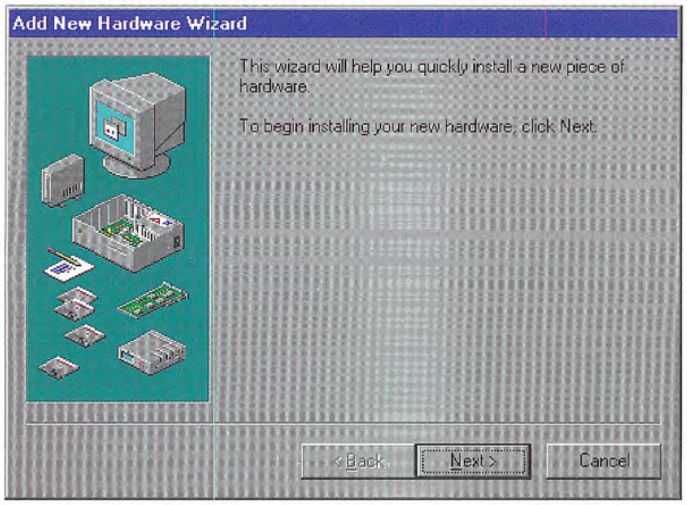
Optionally, you can define wizards as a series of secondary windows through which the user navigates. However, this can lead to increased modality and screen clutter, so using a single secondary window is recommended.

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.

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.

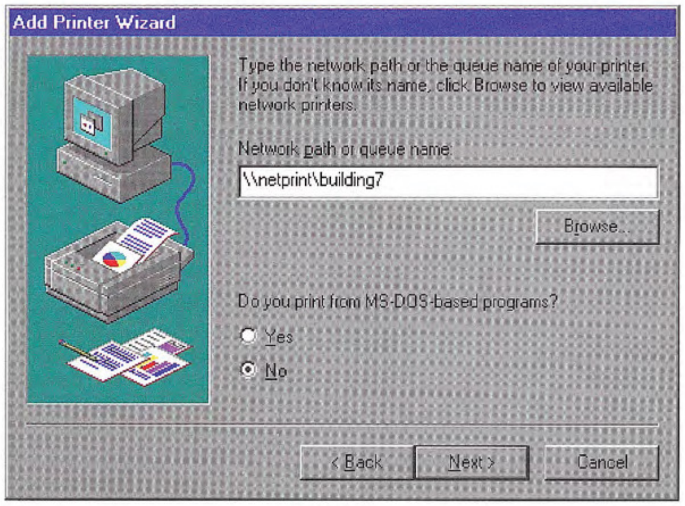


This wizard will help you quickly install a new piece of hardware.

To begin installing your new hardware, click Next.

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.



Type the network path or the queue name of your printer. If you don't know its name, click Browse to view available network printers.

Network path or queue name:

\\netprint\building7

Do you print from MS-DOS-based programs?

Yes [✓ No]

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 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.

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.

CHAPTER 13

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 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:

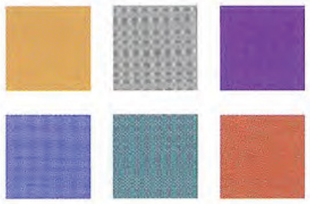
- Could this idea or concept be presented in a simpler manner?
- Can the user easily step through the interface as designed?
- Do all the elements have a reason for being there?

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.

- 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.

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.

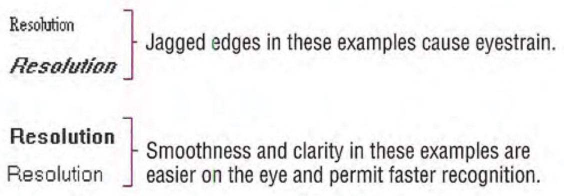

Resolution
Resolution
Jagged edges in these examples cause eyestrain.
Resolution
Resolution
Smoothness and clarity in these examples are easier on the eye and permit faster recognition.

Figure 13.1 Effective and ineffective font choices

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.

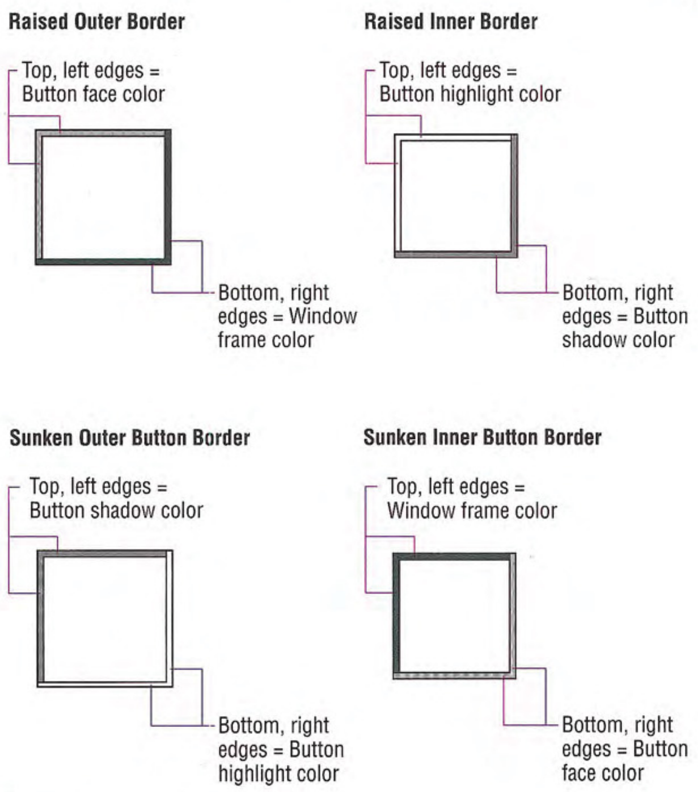
Design of Visual Elements

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

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.

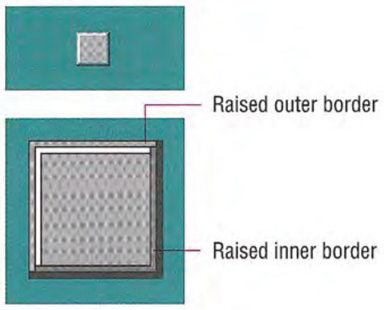


Figure 13.3 Window border style

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.

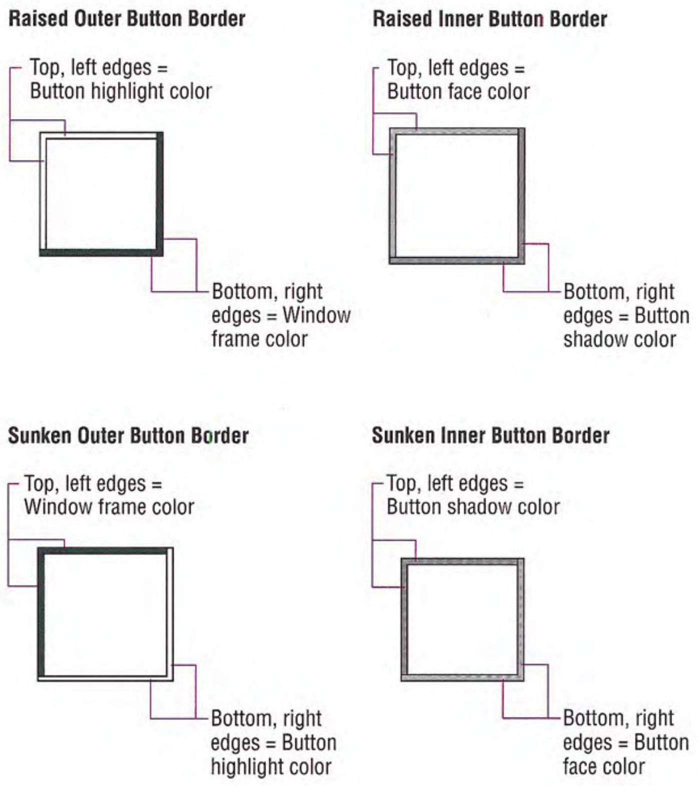

Raised Outer Button Border
Top, left edges = Button highlight color
Bottom, right edges = Window frame color
Raised Inner Button Border
Top, left edges = Button face color
Bottom, right edges = Button shadow color
Sunken Outer Button Border
Top, left edges = Window frame color
Bottom, right edges = Button highlight color
Sunken Inner Button Border
Top, left edges = Button shadow color
Bottom, right edges = Button face color

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.

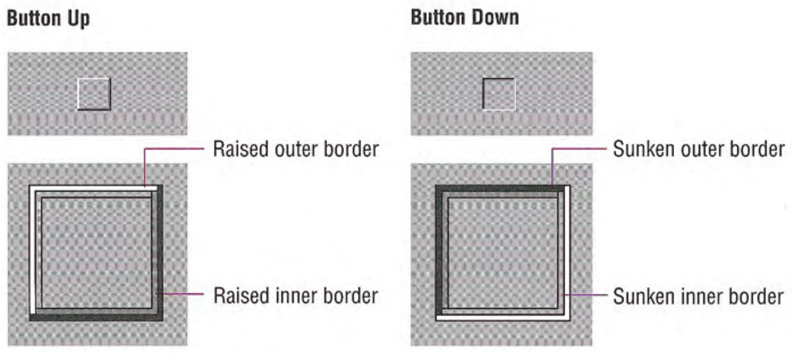


Figure 13.5 Button up and button down border styles

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.

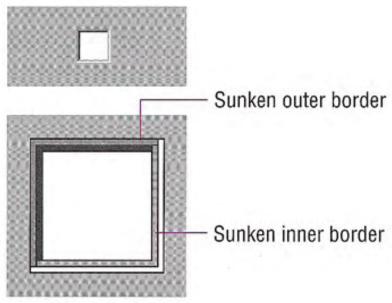


Figure 13.6 The field border style

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.

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.

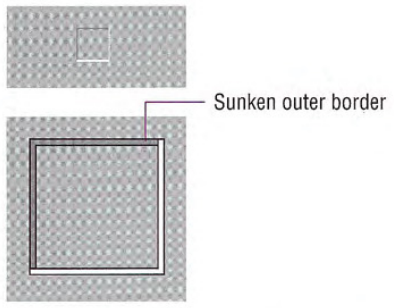


Figure 13.7 The status field border style

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.

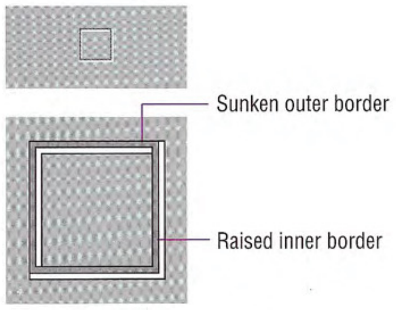


Figure 13.8 The group border style

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.

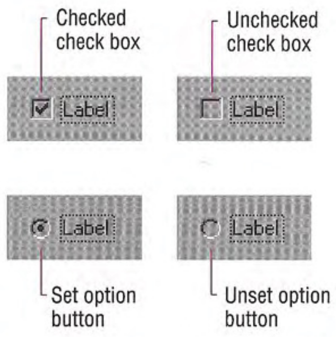


Figure 13.9 Pressed appearance for check boxes and option buttons

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.

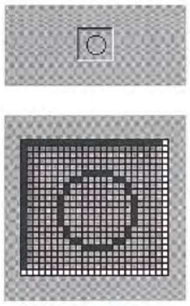


Figure 13.10 Pressed appearance for a command button

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.

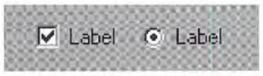


Figure 13.11 Option-set appearance for check boxes and option buttons

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.

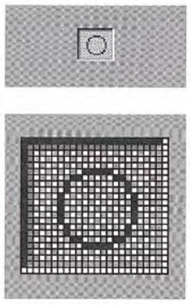


Figure 13.12 Option-set appearance for a command button

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.

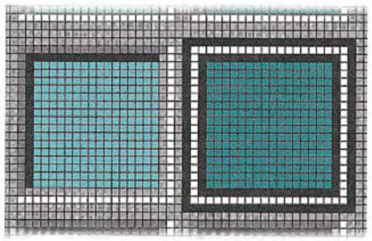


Figure 13.13 Option-set appearance for a well

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.

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.

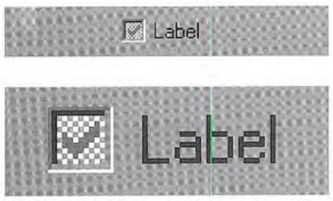


Figure 13.14 Mixed-value appearance for a check box

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.

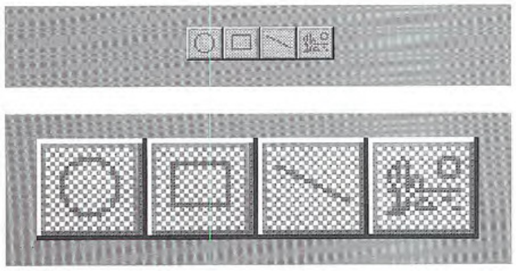


Figure 13.15 Mixed-value appearance for buttons

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.

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.

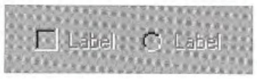


Figure 13.16 Unavailable appearance for check boxes and option buttons

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.

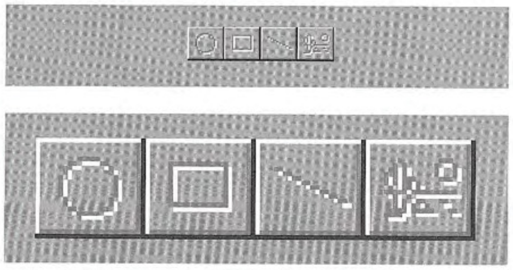


Figure 13.17 Unavailable appearance for buttons

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.

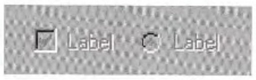


Figure 13.18 Unavailable appearance for check boxes and option buttons (when set)

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.

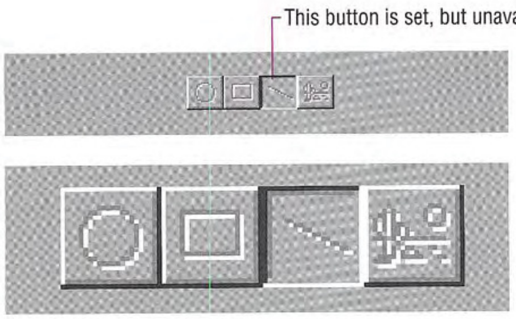


Figure 13.19 Unavailable and option-set appearance for buttons

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.

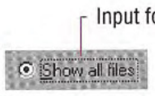


Figure 13.20 Example of input focus in a control

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.

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.

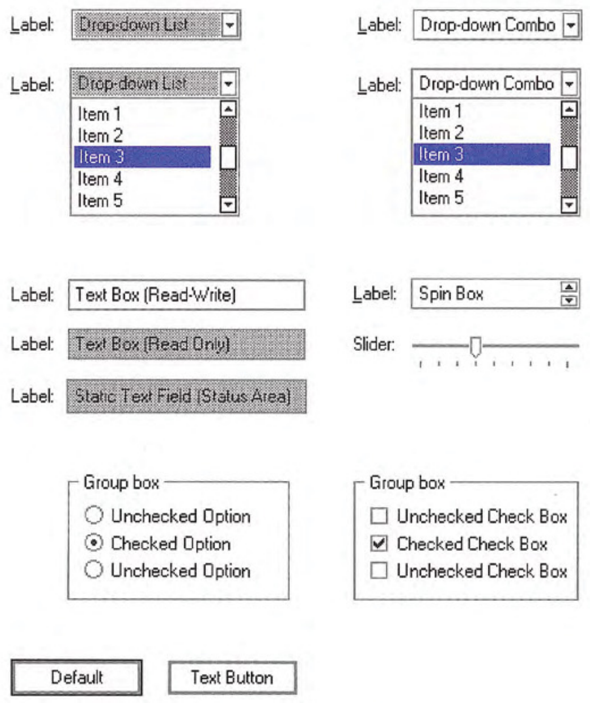


Figure 13.21 Flat appearance for standard controls

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.

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.

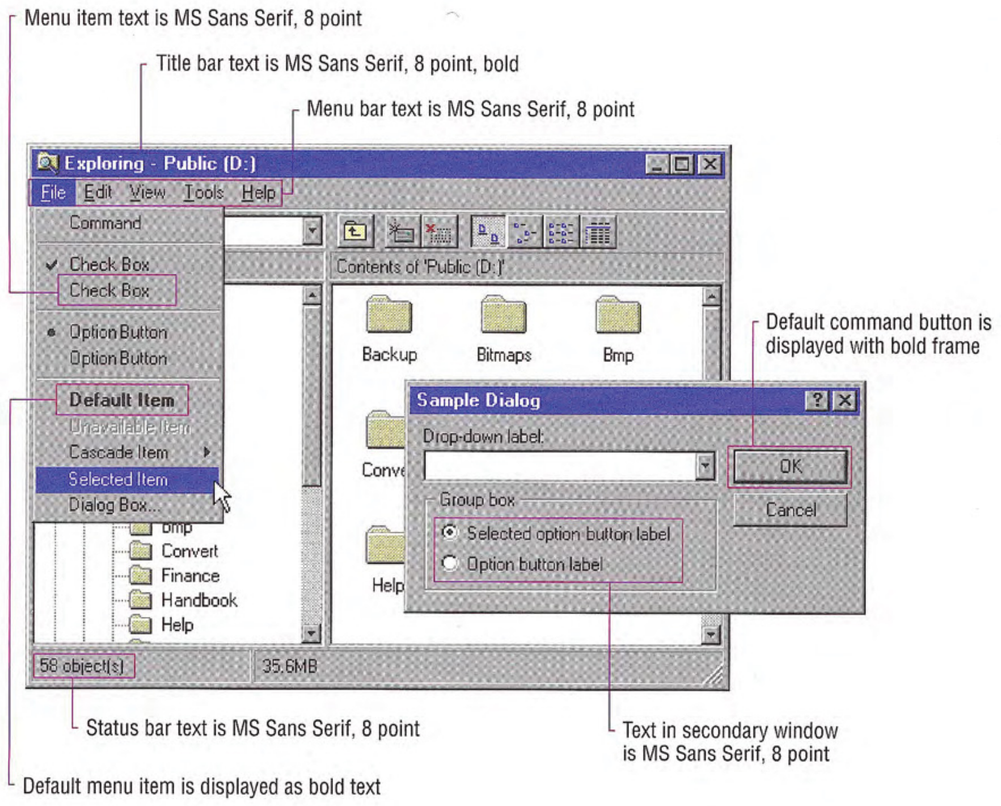

Menu item text is MS Sans Serif, 8 point
Title bar text is MS Sans Serif, 8 point, bold
Menu bar text is MS Sans Serif, 8 point
Exploring - Public (D:)
File Edit View Tools Help
Command
Check Box
Check Box
Option Button
Option Button
Default Item
Unavailable Item
Cascade Item
Selected Item
Dialog Box...
Contents of 'Public (D:)'
Backup
Bitmaps
Bmp
Convert
Finance
Handbook
Help
Default command button is displayed with bold frame
Sample Dialog
Drop-down label:
OK
Cancel
Group box
Selected option button label
Option button label
58 object(s)
35.6MB
Status bar text is MS Sans Serif, 8 point
Text in secondary window is MS Sans Serif, 8 point
Default menu item is displayed as bold text

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.

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."

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.

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.

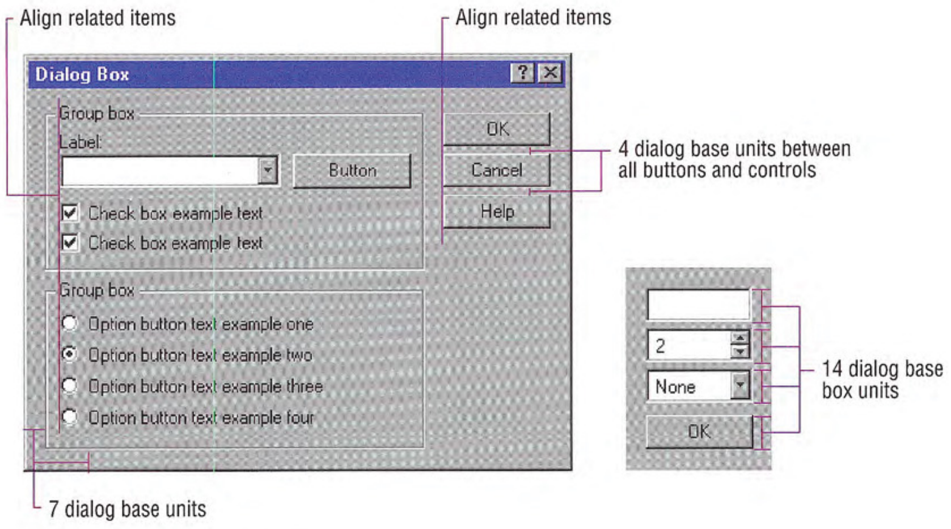


Figure 13.23 Recommended layout and spacing of controls and text

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."

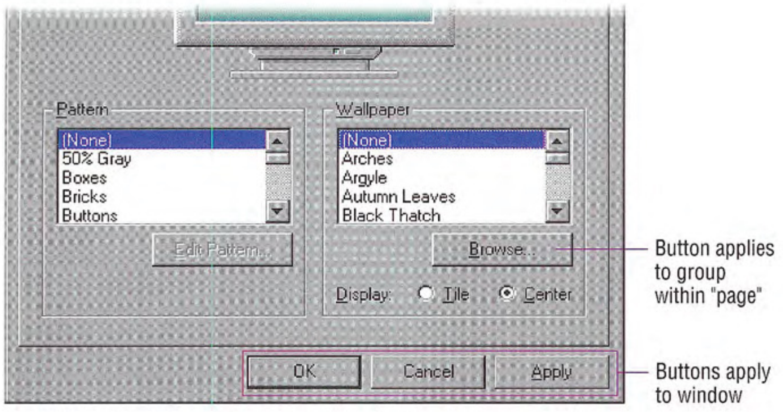


Figure 13.24 Examples of layout of buttons

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.

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.

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**.

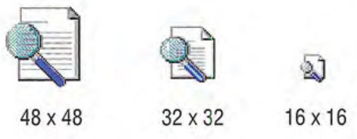


Figure 13.25 Three sizes of icons

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.

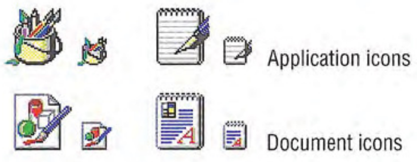


Figure 13.26 Application and supported document icons

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."

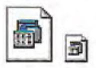


Figure 13.27 System-generated icon for file type without a registered icon

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.

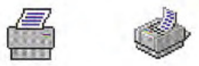


Figure 13.28 Perspective and dimension improve graphics

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.

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.

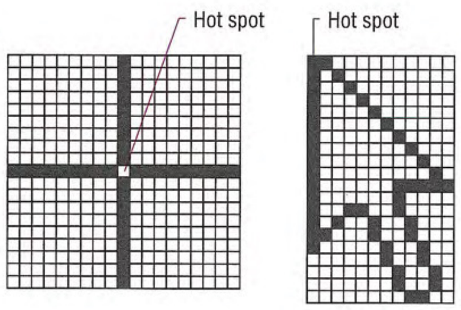


Figure 13.29 Pointer hot spots

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.

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.

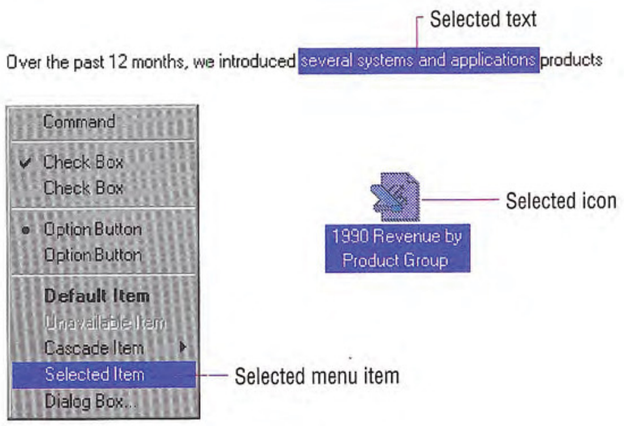


Figure 13.30 Examples of selection appearance

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.

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.

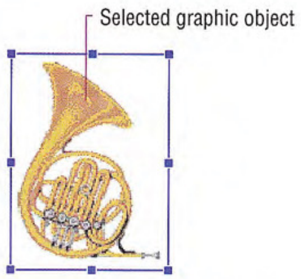


Figure 13.31 Selected graphic object with handles

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.

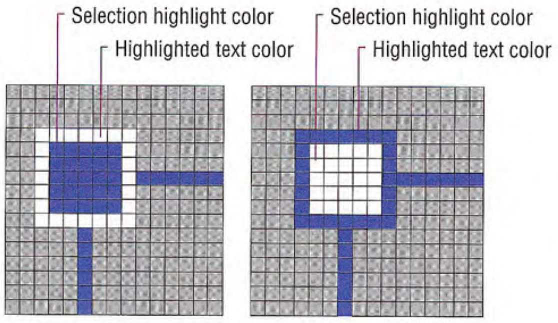


Figure 13.32 Solid and hollow handles

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.

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."

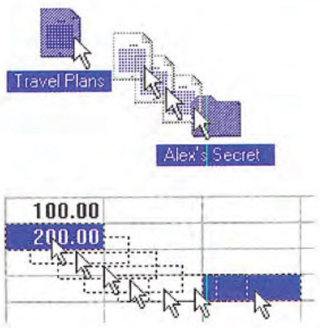


Figure 13.33 Translucent and outline representation (drag transfer)

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.

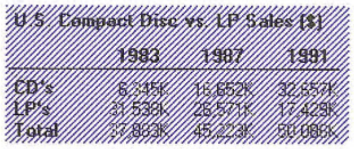
The presentation of an object being transferred displayed is always defined by the destination. Use a representation that best communicates how the transferred object will be incorporated when the user completes the drag transfer. For example, if the object being dragged will be displayed as an icon, 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	18,652K	32,657K
LP's	31,538K	26,571K	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.

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.

CHAPTER 14

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.

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.

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.