Presentation Manager Programming Guide Advanced Topics

IBM

OS/2® WARP

Version 3

Presentation Manager Programming Guide Advanced Topics

IBM

OS/2® WARP

Version 3

Note

Before using this information and the product it supports, be sure to read the general information under Appendix, "Notices" on page A-1.

First Edition (October 1994)

Contents

Presentation Manager Programming Guide - Advanced Topics

Figures

Tables

About This Book

This book provides information and sample code to enable you to write applications that use the *Presentation Manager* (PM) functions, messages and data structures in the application programming interface of the OS/2 operating system.

Who Should Read This Book

This book is intended for application programmers who want to develop programs that use the Presentation Manager programming interface. This guide introduces the advanced topics of PM.

How This Book Is Organized

All chapters of this book, except the Introduction, are divided into nine main sections:

- *About* the topic

  Covers advanced concepts, terminology, and general information about the topic.

- *Using* the topic

  Introduces many of the functions, messages, data structures, and standard controls related to the topic and provides examples in the form of sample code fragments.

- *Graphical User Interface Support* for the topic

  Describes the navigation techniques (pointing device and keyboard support) used in the topic.

- *Enhancing* the topic *Performance and Effectiveness*

  Describes various enhancement techniques that enable you to fine-tune your application's use of the topic.

- Functions

  Provides details of the interfaces for the functions covered in the chapter topic.

- Window Messages

  Provides details of the interfaces for the window messages related to the chapter topic.

- Notification Messages

  Provides details of the interfaces for the notification messages described in the chapter topic.

- Data Structures

  Provides details of the data structures related to the chapter topic.

- Summary

  Provides a brief description of each of the functions, window messages, notification messages, notification codes, data structures, and standard controls covered in the chapter.

There are also sample applications available with the *Developer's Toolkit for OS/2 Version 3*. You may find it useful to execute the samples and examine the C files, resource files, makefiles, and other files provided by the toolkit.

For information on how to compile and link your programs, refer to the compiler publications for the programming language you are using.

Prerequisite Publications

This guide is intended for application designers and programmers who are familiar with the following:

- Information contained in the *Control Program Programming Guide*
- Information contained in the *Presentation Manager Programming Guide - The Basics*
- C Programming Language

Programming experience on a multitasking operating system also would be helpful.

Related Publications

The following diagram provides an overview of the OS/2 Version 3 Technical Library.

Books can be ordered by calling toll free 1-800-342-6672 weekdays between 8:00 a.m. and 8:00 p.m. (EST). In Canada, call 1-800-465-4234.

OS/2 Warp, Version 3 Technical Library
G25H-7116

- Control Program Programming Guide — G25H-7101
- Control Program Programming Reference — G25H-7102
- Graphics Programming Interface Programming Guide — G25H-7106
- Graphics Programming Interface Programming Reference — G25H-7107
- Information Presentation Facility Programming Guide — G25H-7110
- Multimedia Application Programming Guide — G25H-7112
- Multimedia Programming Reference — G25H-7114
- Multimedia Subsystem Programming Guide — G25H-7113
- Presentation Manager Programming Guide - Advanced Topics — G25H-7104
- Presentation Manager Programming Guide - The Basics — G25H-7103
- Presentation Manager Programming Reference — G25H-7105
- REXX Reference — S10G-6268
- REXX User's Guide — S10G-6269
- Tools Reference — G25H-7111
- Workplace Shell Programming Guide — G25H-7108
- Workplace Shell Programming Reference — G25H-7109

IBM Device Driver Publications for OS/2

- Display Device Driver Reference — 71G1896
- Input/Output Device Driver Reference — 71G1898
- MMPM/2 Device Driver Reference — 71G3678
- Pen for OS/2 Device Driver Reference — 71G1899
- Physical Device Driver Reference — 10G6266
- Presentation Driver Reference — 10G6267
- Printer Device Driver Reference — 71G1895
- Storage Device Driver Reference — 71G1897
- Virtual Device Driver Reference — 10G6310

Presentation Manager Programming Guide - Advanced Topics

Chapter 1. Introduction

*Presentation Manager** (PM*) provides a message-based, event-driven, graphical user interface for the Operating System/2* (OS/2*) environment. The advanced features of PM are:

- Atom tables
- Container controls
- Combination-box controls
- Direct Manipulation
- Dynamic data exchange
- File dialog controls
- Font dialog controls
- Hooks
- Multiple-line entry field controls
- Notebook controls
- Slider controls
- Spin button controls
- Static controls
- Value set controls.

PM enables programmers to build applications that conform to Systems Application Architecture* (SAA*) guidelines. For more information on SAA requirements, see the Systems Application Architecture: Common User Access* (CUA*) Guide to User Interface Design and the Systems Application Architecture: Common User Access Advanced Interface Design Reference.

The advanced concepts of PM are described as follows:

Atom Tables

An atom table is an operating system mechanism that an application uses to obtain unique, system-wide identifiers to manage strings efficiently. An application places a string, called an atom name, into an atom table and receives a 32-bit integer value, called an atom, that the application can use to access that string. The application can use the system atom table or a private atom table. The system atom table is available to all applications. When an application places a string in the system atom table, any application that has the atom name can obtain the atom by querying the system atom table. An application can use a private atom table to efficiently manage a large number of strings that are used only within the application. The strings in a private atom table, and the resulting atoms, are available only to the application that created the table.

Combination-Box Controls

A combination box is two controls in one: an entry field and a list box. Combination-box controls enable the user to enter data by typing in the entry field or by choosing a list in the list box. The combination-box control automatically manages the interaction between the entry field and the list box.

Container Controls

A container control provides a way for the user to group related objects for easy access and retrieval. The container also provides the capability to display its contents in different views. Each view presents different information about each object. The container control window displays and processes the user's selection of objects. This control supports direct manipulation of objects, enabling users to drag an object from a container window and drop it on another object or container window.

Direct Manipulation

Direct manipulation is a protocol that enables the user to visually drag an object in a window (the source object) and drop it on another object (the target object) in a window. This causes an interaction, or data exhange, between two windows. The source and the target can be the same window, different window within the same application, or windows belonging to different applications.

Dynamic Data Exchange

The dynamic data exchange (DDE) protocol enables applications to access one another's data. DDE uses PM messages and shared memory to pass data among applications. Data is passed in a mutually-agreed-upon format. An application that receives a handle to a data object will receive a message if the data changes. The application then can indicate, by way of a message, if it wants the changed data to be sent to it, or it can end the exchange.

File Dialog Controls

File dialog controls request file names from users and perform file-name validation. Applications initialize fields and filter strings, and can specify modal or modeless dialog boxes and single or multiple-file selections.

Font Dialog Controls

Font dialog controls request font definitions from users, provide preview windows, and return font family names, point size, type style, emphasis style, and other specifications. Applications can specify modal or modeless dialogs, color selection, and single font selection.

Hooks

A hook is a point in a system-defined function where an application can supply additional code that the system processes as though it were part of the function. Many operating system functions provide points where an application can hook in its own code to enhance or override the default processing of the function. The OS/2 operating system contains many types of hooks, and the system maintains a separate hook list for each type of hook supported.

Multiple-Line Entry Field Controls

A multiple-line entry (MLE) field is a sophisticated control window that enables a user to view and edit multiple lines of text. An MLE field control gives an application the text-editing capabilities of a simple text editor. The application can create a multiple-line entry field by using a function or by specifying the MLE statement in a dialog-window template in a resource-definition file.

Notebook Controls

The notebook control organizes access to multiple groups of controls. The overall appearance of this control is a notebook. An application can dynamically insert or delete pages, specify colors for different notebook areas, and resize parts of the notebook.

Slider Controls

A slider control displays a range of values and allows a user to set, display, or modify a value by moving a slider arm. There are two types of sliders. The linear slider is represented as a shaft along which the slider arm can be moved by the user to set a value. The circular slider is represented as a dial with the slider arm shown as the radius of the dial. Typically, linear sliders are used to easily set values that have familiar increments. The circular slider control provides an analog user interface and emulates the controls of stereo and video equipment. The application can specify different scales, sizes, and orientations for its sliders. You can use both types of sliders in a window to create a user interface that makes good use of available space and provides a familiar appearance to the user.

Spin Button Controls

A spin button control gives users quick access to a finite set of data by letting them select from a scrollable ring of choices. You can also create multi-field spin buttons for those applications in which users must select more than one value.

Static Controls

A static control is a simple text field, bit map, or icon that an application can use to label, enclose, or separate other control windows. A static control does not accept user input or send notification messages to its owner. The primary advantage of a static control is that it provides a label or graphic that requires little attention from an application. At most, an application might change the text or position of a static control.

Value Set Controls

A value set control enables a user to select one choice from a group of mutually exclusive choices. A value set can use graphic images (bit maps or icons), as well as colors, text, and numbers, to represent the items a user can select. Although text is supported, the purpose of a value set control is to display choices as graphic images. The user can see the selections instead of having to take time to read descriptions of the choices. The application can specify different types of items, sizes, and orientations for its value sets.

Chapter 2. Combination Box

A *combination box* is two controls in one: an entry field and a list box. This chapter describes how to use *combination-box controls*, also called *combination boxes* and *prompted entry fields*, to let the user choose and edit items from a list in PM applications.

About Combination-Box Controls

Combination-box controls enable the user to enter data by typing in the entry field or by choosing from a list in the list box. Figure 2-1 shows an example of a combination box.

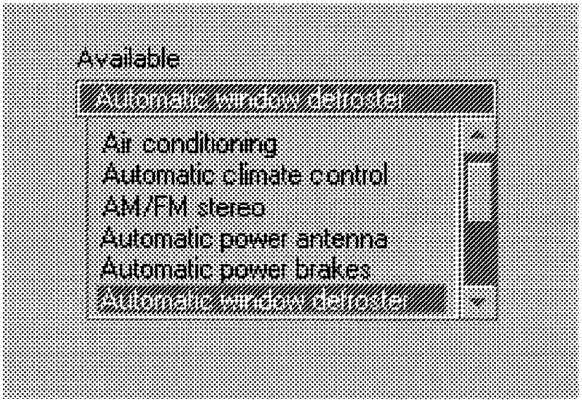

Figure 2-1. Combination-Box Example

A combination-box control automatically manages the interaction between the entry field and the list box. For example, when the user chooses an item in the list box, the combination-box control displays the text for that item in the entry field. Then, the user can edit the text without affecting the item in the list box. When the user types a letter in the entry field, the combination-box control scrolls the list box contents so that items beginning with that letter become visible.

Combination-Box Styles

Table 2-1 shows the combination-box styles.

Table 2-1. Combination-Box Control Styles	
Style Name	**Description**
CBS_SIMPLE	Creates a simple combination box that always displays its list box. The user can enter and edit text in the entry field or choose items from the list box.
CBS_DROPDOWN	Creates a drop-down combination box that displays its list box only if the user clicks the drop-down icon at the right end of the entry field. The combination-box control hides the list box when the user clicks the icon a second time. In a drop-down combination box, the user can enter and edit text in the entry field or choose items from the list box.
CBS_DROPDOWNLIST	Creates a drop-down-list combination box that is similar to the drop-down combination box, except that the user can choose items only from the list box. The user cannot enter or edit text in the entry field.

For combination boxes that have the CBS_DROPDOWN or CBS_DROPDOWNLIST styles, an application can display the list by using the CBM_SHOWLIST message. Figure 2-2 and Figure 2-3 on page 2-3 show an example of a drop-down combination box and a drop-down list box, respectively.

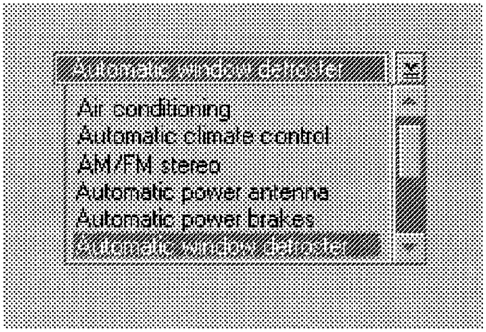

Figure 2-2. Drop-Down Combination-Box Example

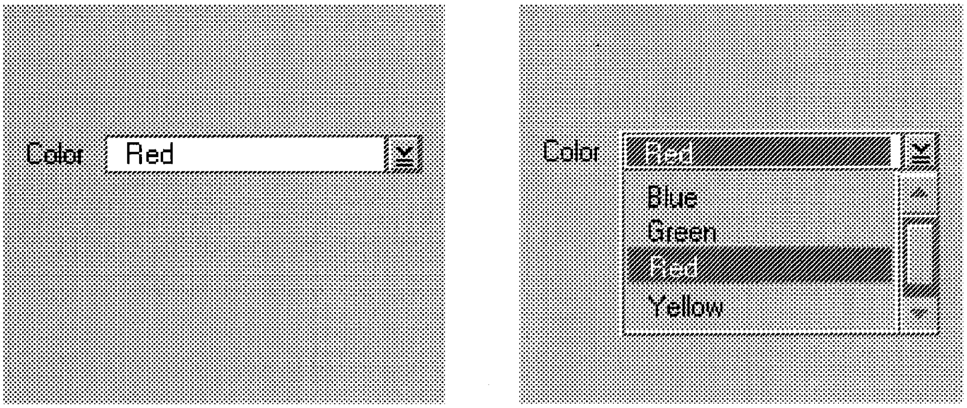

Figure 2-3. Drop-Down List-Box Example

An application can determine whether the list is already showing by using the CBM_ISLISTSHOWING message.

Applications also can use any of the entry-field (EM_) and list-box (LM_) messages with combination boxes. Entry-field messages affect the entry field; list-box messages affect the list box. For example, an application can use the LM_INSERTITEM message to insert items into the list box.

Combination-Box Notification Codes

A combination-box control sends WM_CONTROL messages containing notification codes to its parent window. These notification codes are similar to those sent by entry-field and list-box controls. A combination-box control sends a notification codes to its owner window.

Using Combination-Box Controls

You can create a combination box by using WinCreateWindow or by specifying a COMBOBOX statement in a dialog-window template in a resource file. When creating a combination box using WinCreateWindow, you must specify the predefined class WC_COMBOBOX. If you do not specify a style, the function uses the default styles WS_GROUP, WS_TABSTOP, and WS_VISIBLE.

Related Window Messages

This section covers the window messages that are related to combination-box controls.

CBM_HILITE

This message sets the highlighting state of the entry field control.

Parameters

param1

usHilite (USHORT)
Highlighting indicator.

TRUE	Highlight the entry field control.
FALSE	Do not highlight the entry field control.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Changed indicator.

TRUE	The highlighting state of the entry field has been changed.
FALSE	The highlighting state of the entry field has not been changed.

CBM_ISLISTSHOWING

This message determines if the list box control is showing.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Showing indicator.

TRUE	The list box control is showing.
FALSE	The list box control is not showing.

CBM_SHOWLIST

This message sets the showing state of the list box control.

Parameters

param1

usShowing (USHORT)
Showing indicator.

TRUE	Show the list box control.
FALSE	Do not show the list box control.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Changed indicator.

TRUE	The list box showing state has been changed.
FALSE	The list box showing state has not been changed.

Related Notification Message

This section covers the notification message that is related to combination-box controls.

WM_CONTROL (in Combination Boxes)

For the cause of this message, see "WM_CONTROL" on page 5-15.

Parameters

param1

usid (USHORT)
Control window identity.

usnotifycode (USHORT)
Notify code.

CBN_EFCHANGE	The content of the entry field control has changed, and the change has been displayed on the screen.
CBN_MEMERROR	The entry field control cannot allocate the storage necessary to accommodate window text of the length implied by the EM_SETTEXTLIMIT message.
CBN_EFSCROLL	The entry field control is about to scroll horizontally. This can happen in these circumstances: • The application has issued a WinScrollWindow call. • The content of the entry field control has changed. • The caret has moved. The entry field control must scroll to show the caret position.
CBN_LBSELECT	An item in the list box control has been selected.
CBN_LBSCROLL	The list box is about to scroll.
CBN_SHOWLIST	The list box is about to be displayed.
CBN_ENTER	The user has depressed the ENTER key or double clicked (single clicked in the case of a drop-down list) on an item in the list box control.

param2

hwndcontrolspec (HWND)
Combination (combo) window handle.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

Summary

Following are tables that describe the OS/2 window messages, notification message, and notification codes used with combination-box controls:

Table 2-2. Combination-Box Control Window Messages

Message Name	Description
CBM_HILITE	Sets the highlighting state of the entry field control.
CBM_ISLISTSHOWING	Determines whether the list box control is showing.
CBM_SHOWLIST	Sets the showing state of the list box control.

Table 2-3. Combination-Box Control Notification Message

Message Name	Description
WM_CONTROL	Occurs when a control has a significant event to notify to its owner.

Table 2-4. Combination-Box Control Notification Codes

Code Name	Description
CBN_EFCHANGE	Indicates that the text in a combination-box entry field has changed.
CBN_EFSCROLL	Indicates that the text in a combination-box entry field has been scrolled.
CBN_ENTER	Indicates that a combination-box item has been selected.
CBN_LBSCROLL	Indicates that a combination-box list has been scrolled.
CBN_LBSELECT	Indicates that a combination-box list item has been selected.
CBN_MEMERROR	Indicates that the combination-box control cannot allocate sufficient memory.
CBN_SHOWLIST	Indicates that a combination-box list has dropped down (is visible).

Chapter 3. Multiple-Line Entry Field Controls

A *multiple-line entry (MLE) field* is a sophisticated control window that enables a user to view and edit multiple lines of text. This chapter describes how to create and use multiple-line entry field controls in PM applications.

About Multiple-Line Entry Field Controls

An MLE field control gives an application the text-editing capabilities of a simple text editor. The application can create a multiple-line entry field by using WinCreateWindow or by specifying the MLE statement in a dialog-window template in a resource-definition file.

MLE Styles

The style of an MLE field control determines how the MLE field appears and behaves. An application can specify a combination of the styles listed in Table 3-1.

Table 3-1. MLE Styles

Style Name	Description
MLS_BORDER	Draws a border around the MLE field.
MLS_DISABLEUNDO	Directs the MLE control not to allow undo actions.
MLS_HSCROLL	Adds a horizontal scroll bar to the MLE field. The MLE control enables this scroll bar whenever any line exceeds the width of the MLE field.
MLS_IGNORETAB	Directs the MLE control to ignore the Tab key.
MLS_READONLY	Prevents the MLE field from accepting text from the user. This style is useful for displaying lengthy static text in a client or dialog window.
MLS_VSCROLL	Adds a vertical scroll bar to the MLE field. The MLE control enables this scroll bar whenever the number of lines exceeds the height of the MLE field.
MLS_WORDWRAP	Automatically breaks lines that are longer than the width of the MLE field.

MLE Notification Codes

An MLE field control sends WM_CONTROL messages containing notification codes to its owner whenever certain events occur, for example, when the user or application tries to insert too much text, or when the user uses the scroll bars. The owner window uses the notification codes either to carry out custom operations for the MLE field or to respond to errors.

The MLE field control sends the MLN_HSCROLL or MLN_VSCROLL notification codes when the user enables the scroll bars so that the application can monitor the visible contents of the MLE field. The application also can monitor the contents of an MLE field by using the MLM_QUERYFIRSTCHAR message, which specifies the offset of the character in the upper-left corner of the MLE field. This represents the first MLE character that is visible to the user. To provide an alternative way of scrolling the contents of an MLE field, an application can move the character at the specified offset to the upper-left corner of an MLE field using the MLM_SETFIRSTCHAR message.

The MLE field control sends an MLN_CHANGE notification code when the user changes the text in some way. This notification code is especially useful when the MLE field is in a dialog window, because the dialog procedure can use this code to determine whether it should process the contents of the MLE field. If an application does not process MLN_CHANGE notification codes, it can use the MLM_QUERYCHANGED message to determine whether the user has made changes to the MLE text. The MLM_SETCHANGED message makes the MLE field control send an MLN_CHANGE notification code with every event that occurs in the MLE field, regardless of whether the user has changed anything. This code also can be used to hide a change made by a user.

MLE Text Editing

An MLE field contains one or more lines of text. Each line consists of one or more characters and ends with one or more characters that represent the end of the line. The end-of-line characters are determined by the format of the text.

The user can type text in an MLE field when the MLE field has the focus. The application can insert text at any time by using the MLM_INSERT message and specifying the text as a null-terminated string. The MLE field control inserts the text at the cursor position or replaces the selected text.

The MLE field control entry mode, insert or overstrike, determines what happens when the user inserts text. The user sets the entry mode by pressing the Insert key. The entry mode alternates each time the user presses Insert. When overstrike mode is enabled, at least one character is selected. This means that the MLM_INSERT message always replaces at least one character. If insert mode is enabled, the MLM_INSERT message replaces only those characters the user or application has selected. Otherwise, the MLE field makes room for the inserted characters by moving existing characters to the right, starting at the cursor position.

The cursor position, identified by a blinking bar, is specified as a character offset relative to the beginning of the text. The user can set the cursor position by using the mouse or Arrow keys to move the blinking bar. An application can set the cursor position by using the MLM_SETSEL message, which directs the MLE field control to move the blinking bar to a given character position. The MLM_SETSEL message also can set the selection.

The *selection* is one or more characters of text on which the MLE field control carries out an operation, such as deleting or copying. The user selects text by pressing the Shift key while moving the cursor or by pressing mouse button 1 while moving the mouse. The user also can select a word in a block of text by double-clicking on the word. An application selects text by using the MLM_SETSEL message to specify the cursor position and the anchor point. The selection is all the text between the cursor position and the anchor point. If the cursor position and anchor point are equal, there is no selection. An application can retrieve the cursor position, anchor point, or both, by using the MLM_QUERYSEL message.

The user can delete characters, one at a time, by pressing the Delete key or the Backspace key. Pressing the Delete key deletes the character to the right of the cursor; pressing the Backspace key deletes the character to the left of the cursor and changes the cursor position. An application can delete one or more characters by using the MLM_DELETE message, which directs the MLE field control to delete a specified number of characters, starting at the given position. This message does not change the cursor position. An application can delete selected text by using the MLM_CLEAR message.

An application can reverse the previous operation by using the MLM_UNDO message, which restores the MLE field to its previous state. This is a quick way to fix editing mistakes. However, not all operations can be undone.

The application determines whether the previous operation can be undone by using the MLM_QUERYUNDO message, which returns TRUE and indicates the type of operation that can be undone. Using the MLM_RESETUNDO message, an application can prevent a subsequent MLM_UNDO message from changing the state of an MLE field.

MLE Text Formatting

An application can retrieve the number of lines of text in an MLE field by using the MLM_QUERYLINECOUNT message and can retrieve the number of characters in the MLE field by using the MLM_QUERYTEXTLENGTH message. The amount of text and, subsequently, the number of lines to be entered in an MLE field depend on the text limit. An application sets the text limit by using the MLM_SETTEXTLIMIT message and determines the current limit by using the MLM_QUERYTEXTLIMIT message. The user cannot set the text limit. If the user types to the text limit, the MLE field control beeps and ignores any subsequent keystrokes. If the application attempts to add text beyond the limit, the MLE field control truncates the text.

An application can control the length of each line in an MLE field by enabling word wrapping. When word wrapping is enabled, the MLE field control automatically breaks any line that is longer than the width of the MLE field. An application can set word wrapping by using the MLM_SETWRAP message, and it can determine whether the MLE field control is wrapping text by using the MLM_QUERYWRAP message. Word wrapping is disabled by default unless the application specifies the MLS_WORDWRAP style when creating the MLE field control.

An application can set tab stops for an MLE control by using the MLM_SETTABSTOP message. Tab stops specify the maximum width of a tab character. When the user or an application inserts a tab character, the MLE field control expands the character so that it fills the space between the cursor position and the next tab stop. The MLM_SETTABSTOP message sets the distance (in pels) between tab stops, and the MLE field control provides as many tab stops as necessary, no matter how long the line gets. An application can retrieve the distance between tab stops using the MLM_QUERYTABSTOP message.

An application can use the MLM_SETFORMATRECT message to set the *format rectangle* (MLE field). The format rectangle is used to set the horizontal and vertical limits for text. The MLE control sends a notification message to the parent window of the MLE field if text exceeds either of those limits. An application typically uses the format rectangle to provide its own word wrapping or other special text processing. An application can retrieve the current format rectangle by using the MLM_QUERYFORMATRECT message.

An application can prevent the user's editing of the MLE field by setting the MLS_READONLY style in WinCreateWindow or in the MLE statement in the resource-definition file. The application also can set and query the read-only state by using the MLM_SETREADONLY and MLM_QUERYREADONLY messages, respectively.

An application can set the colors and font for an MLE field by using the MLM_SETTEXTCOLOR, MLM_SETBACKCOLOR, and MLM_SETFONT messages. These messages affect all text in the MLE field. An MLE field cannot contain a mixture of fonts and colors. An application can retrieve the current values for the colors and font by using the MLM_QUERYTEXTCOLOR, MLM_QUERYBACKCOLOR, and MLM_QUERYFONT messages.

To prevent scrolling within the MLE when the MLS_READONLY style bit is set, use the MLM_DISABLEREFRESH message. The keyboard and mouse input can be enabled using the MLM_ENABLEREFRESH message.

MLE Text Import and Export Operations

An application can copy text to and from an MLE field by importing and exporting. To import text to an MLE field, an application can use the MLM_IMPORT message, which copies text from a buffer to the MLE field. To export text from an MLE field, the application can use the MLM_EXPORT message, which copies text from the MLE field to a buffer. The application uses the MLM_SETIMPORTEXPORT message to set the import and export buffers.

An application can import and export text in a variety of formats. A text format, set with the MLM_FORMAT message, identifies which characters are used for the end-of-line characters. An MLE field can have the text formats that are listed in Table 3-2.

Table 3-2. MLE Text Format

Format Name	Description
MLFIE_CFTEXT	Exported lines end with a carriage return/newline character pair (*0x0D, 0x0A*). Imported lines must end with a newline character, carriage return/newline character pair, or newline/carriage return character pair.
MLFIE_NOTRANS	Imported and exported lines end with a newline character (*0x0A*).
MLFIE_WINFMT	For exported lines, the carriage return/newline character pair marks a *hard* linebreak (a break entered by the user). Two carriage-return characters and a newline character (*0x0D, 0x0D, 0x0A*) mark a *soft* linebreak (a break inserted during word wrapping and not entered by the user). For imported lines, the extra carriage-return in soft linebreak characters is ignored.

The text format can affect the number of characters in a selection. To ensure that the export buffer is large enough to hold exported text, an application can send the MLM_QUERYFORMATLINELENGTH message. The application can send the

MLM_QUERYFORMATTEXTLENGTH message to determine the number of bytes in the text to be exported.

Each time an application inserts text in an MLE field, the MLE field control automatically refreshes (repaints) the display by drawing the new text. When an application copies large amounts of text to an MLE field, refreshing can be quite time-consuming, so the application should disable the refresh state. The application disables the refresh state by sending the MLM_DISABLEREFRESH message. After copying all the text, the application can restore the refresh state by sending the MLM_ENABLEREFRESH message.

MLE Cut, Copy, and Paste Operations

The user can cut, copy, and paste text in an MLE field by using the Shift+Delete, Ctrl+Insert, and Shift+Insert key combinations, respectively. An application—either by itself or in response to the user—can cut, copy, and paste text by using the MLM_CUT, MLM_COPY, and MLM_PASTE messages. The MLM_CUT and MLM_COPY messages copy the selected text to the clipboard. The MLM_CUT message also deletes the text from the MLE field; MLM_COPY does not. The MLM_PASTE message copies the text from the clipboard to the current position in the MLE field, replacing any existing text with the copied text. An application can delete the selected text without copying it to the clipboard by using the MLM_CLEAR message.

An application also can copy the selected text from an MLE field to a buffer by using the MLM_QUERYSELTEXT message. This message does not affect the contents of the clipboard.

MLE Search and Replace Operations

An application can search for a specified string within MLE field text by using the MLM_SEARCH message, which searches for the string. The MLE field control returns TRUE if the string is found. The cursor does not move to the string unless the message specifies the MLFSEARCH_SELECTMATCH option.

An application also can use the MLM_SEARCH message to replace one string with another. If the message specifies the MLFSEARCH_CHANGEALL option, the MLE field control replaces all occurrences of the search string with the replacement string. Both the search string and the replacement string must be specified in an MLE_SEARCHDATA data structure passed with the message.

MLE Colors

MLE supports indexed colors (solid) only; it does not support dithered (RGB) colors. If an RGB color is specified for the MLE, it is changed to the closest solid color representation.

Using Multiple-Line Entry Field Controls

This section explains how to create an MLE field control by using WinCreateWindow and by specifying the MLE statement in a dialog template in a resource-definition file.

Creating an MLE

The sample code illustrated in Figure 3-1 shows how to create an MLE by using WinCreateWindow.

```
#define MLE_WINDOW_ID 2

HWND hwndParent;
HWND hwndMLE;

hwndMLE = WinCreateWindow(
            hwndParent,    /* Parent window     */
            WC_MLE,        /* Window class      */
            "Test",        /* Initial text      */

            WS_VISIBLE |   /* Window style      */
            MLS_BORDER,    /* Window style      */
            100, 100,      /* x and y positions */
            100, 100,      /* Width and height  */
            hwndParent,    /* Owner window      */
            HWND_TOP,      /* Top of z-order    */
            MLE_WINDOW_ID, /* Identifier        */
            NULL,          /* Control data      */
            NULL);         /* Presparam         */
```

Figure 3-1. Sample Code for Creating an MLE Using WinCreateWindow

It also is common to create an MLE field control by using an MLE statement in a dialog-window template in a resource file, as shown in the code fragment illustrated in Figure 3-2.

```
MLE  "",
     IDD_MLETEXT,
     110, 10, 50, 100,
     WS_VISIBLE |
     MLS_BORDER |
     MLS_WORDWRAP
```

Figure 3-2. Sample Code for Creating an MLE Using an MLE

The predefined class for an MLE control is WC_MLE. If you do not specify a style for the MLE control, the default styles used are MLS_BORDER, WS_GROUP, and WS_TABSTOP.

Importing and Exporting MLE Text

Importing and exporting MLE text takes place through a buffer. An *import* operation copies text from the buffer to the MLE field; an *export* operation copies text from the MLE to the buffer. Before an application can import or export MLE text, it must send an MLM_SETIMPORTEXPORT message to the MLE field control, specifying the address and size of the buffer.

The maximum size of import/export buffer is 64K. Once the data is into the buffer, the data is manipulated (verified for carriage returns, line feeds and so forth), and is finally placed in the MLE's memory.

Importing MLE Text

To import text, an application sends the MLM_IMPORT message to the MLE field control. This message requires two parameters: *plOffset* and *cbCopy*. The *plOffset* parameter is a pointer to a variable that specifies the position in the MLE field where the text from the buffer is to be placed. The position is an *offset* from the beginning of the MLE text, that is, the number of characters from the beginning of the MLE text. If *plOffset* points to a variable that equals -1, the MLE field control places the text starting at the current cursor position. On return, this variable contains the offset to the first character beyond the imported text. The *cbCopy* parameter of the MLM_IMPORT message points to a variable that specifies the number of bytes to import.

The following criterias apply when importing MLE text:

- If the text ends by a line feed (LF), the import logic generates a blank line.
- If the text ends by a carriage return (CR), MLE prevents a line break (LB) but flags the condition.
- If the *plOffset* field points to the current cursor position (-1) and the import text contains a LF:
  - If the MLE text is imported before the text being edited, then the cursor does not move and the text being edited is shifted down to make room for the text being imported.
  - If the MLE text is imported after the text being edited, then the cursor does not move and the text being imported is inserted starting at the current cursor position.
- If the *plOffset* field points to the current cursor position (-1) and the import text does not contain a LF:
  - If the MLE text is imported before the text being edited, then the cursor does not move and the text being edited is shifted to the right to make room for the text being imported.
  - If the MLE text is imported after the text being edited, then the cursor does not move and the text being imported is inserted starting at the current cursor position.

Exporting MLE Text

Before using the MLM_EXPORT message the number of characters to export needs to be determined. The MLM_QUERYFORMATTEXTLENGTH message is used to determine the number of characters to be copied from the MLE to the buffer (including LF and CR) and to allocate the room in the buffer. MLM_EXPORT is then used to export the MLE text into the buffer.

Note: The MLM_QUERYTEXTLENGTH message does not consider the CR and LF characters as the MLM_QUERYFORMATTEXTLENGTH message does.

The sample code illustrated in Figure 3-3 reads text from a file to a buffer, then imports the text to an MLE field.

```
HWND  hwndMle;
CHAR  szMleBuf[512];
IPT   lOffset = 0;
PSZ   pszTextFile;
HFILE hf;
ULONG cbCopied;
ULONG ulAction;
ULONG cbBytesRead;

/* Obtain a file name from the user */

/* Open the file */

DosOpen(pszTextFile,
        &hf,
        &ulAction,
        0,
        FILE_NORMAL,
        FILE_OPEN |
        FILE_CREATE,
        OPEN_ACCESS_READONLY |
        OPEN_SHARE_DENYNONE,
        NULL);

/* Zero-fill the buffer using memset, a C run-time function */
memset(szMleBuf, 0, sizeof(szMleBuf));
```

Figure 3-3 (Part 1 of 2). Sample Code for Importing and Exporting Text

```
/* Set the MLE import-export buffer */
WinSendMsg(hwndMle,
           MLM_SETIMPORTEXPORT,
           MPFROMP(szMleBuf),
           MPFROMSHORT((USHORT) sizeof(szMleBuf)));

/*********************************************************************/
/*  Read the text from the file to the buffer,                      */
/*  then import it to the MLE.                                      */
/*********************************************************************/

do {
     DosRead(hf,
             szMleBuf,
             sizeof(szMleBuf),
             &cbBytesRead);

     cbCopied = (ULONG) WinSendMsg(hwndMle,
                        MLM_IMPORT,
                        MPFROMP( &lOffset),
                        MPFROMP(&cbBytesRead));
   } while (cbCopied);

/* Close the file */
DosClose(hf);
```

Figure 3-3 (Part 2 of 2). Sample Code for Importing and Exporting Text

To export MLE text, an application sends the MLM_EXPORT message to the MLE control. Like MLM_IMPORT, the MLM_EXPORT message takes the *plOffset* and *cbCopy* parameters. The *plOffset* parameter is a pointer to a variable that specifies the offset to the first character to export. A value of -1 specifies the current cursor position. On return, the variable contains the offset to the first character in the MLE field not copied to the buffer. The *cbCopy* parameter is a pointer to a variable that specifies the number of bytes to export. On return, this variable equals 0 if the number of characters actually copied does not exceed the number specified to be copied. The sample code illustrated in Figure 3-4 on page 3-10 shows how to export text from an MLE field, then store the text in a file.

```
HWND  hwndMle;
CHAR  szMleBuf[512];
IPT   lOffset = 0;
PSZ   pszTextFile;
HFILE hf;

ULONG cbCopied;
ULONG ulAction;
ULONG cbBytesWritten;
ULONG cbCopy;

/* Zero-fill the buffer using memset, a C run-time function */
memset(szMleBuf, 0, sizeof(szMleBuf));

/* Set the MLE import-export buffer */
WinSendMsg(hwndMle,
           MLM_SETIMPORTEXPORT,
           MPFROMP(szMleBuf),
           MPFROMSHORT ((USHORT) sizeof(szMleBuf)));

   .
   .
   .

/* Obtain a filename from the user */
   .
   .
   .

/* Open the file */
DosOpen(pszTextFile,
        &hf,
        &ulAction,
        0,
        FILE_NORMAL,
        FILE_OPEN |
        FILE_CREATE,
        OPEN_ACCESS_WRITEONLY |
        OPEN_SHARE_DENYNONE,
        NULL);
```

Figure 3-4 (Part 1 of 2). Sample Code for Exporting Text from an MLE Field

```
/* Find out how much text is in the MLE */
cbCopy = (ULONG) WinSendMsg(hwndMle,
                            MLM_QUERYFORMATTEXTLENGTH,
                            MPFROMLONG(lOffset),
                            MPFROMLONG((-1)));

/* Copy the MLE text to the buffer */
cbCopied = (ULONG) WinSendMsg(hwndMle,
                              MLM_EXPORT,
                              MPFROMP(&lOffset),
                              MPFROMP(&cbCopy));

/* Write the contents of the buffer to the file */
DosWrite(hf,
         szMleBuf,
         sizeof(szMleBuf),
         &cbBytesWritten);

/* Close the file */
DosClose(hf);
```

Figure 3-4 (Part 2 of 2). Sample Code for Exporting Text from an MLE Field

Searching MLE Text

An application uses the MLM_SEARCH message and the MLE_SEARCHDATA data structure to search for strings in MLE text. The first parameter of the MLM_SEARCH message is an array of flags that specify the style of the search. The application can set the MLFSEARCH_CASESENSITIVE flag if a case-sensitive search is required. If the application sets the MLFSEARCH_SELECTMATCH flag, the MLE field control highlights a matching string and, if necessary, scrolls the string into view. An application can use the MLFSEARCH_CHANGEALL flag to replace every occurrence of the string with the string specified in the *pchReplace* member of the MLE_SEARCHDATA data structure.

The second parameter of the MLM_SEARCH message is a pointer to an MLE_SEARCHDATA data structure that contains information required to perform the search operation. This data structure includes a pointer to the string and, if the MLFSEARCH_CHANGEALL flag is set in the MLM_SEARCH message, a pointer to the replacement string. The *iptStart* and *iptStop* members specify the starting and ending positions of the search. These positions are specified as offsets from the beginning of the MLE field. A value of -1 in the *iptStart* member causes the search to start at the current cursor position. A negative value in the *iptStop* member causes the search to end at the end of the MLE field. If a matching string is found, the MLE field control returns the length of the string in the *cchFound* member.

The following code fragment uses an entry field to obtain a search string from the user, then searches an MLE field for an occurrence of the string. The search begins at the current cursor position and ends at the end of the MLE text. When the

MLFSEARCH_SELECTMATCH flag is specified, the MLE field control highlights a matching string and scrolls it into view.

The sample code illustrated in Figure 3-5 shows how to search MLE text.

```
#define IDD_SEARCHFIELD  101

HWND hwnd;
HWND hwndEntryFld;
HWND hwndMle;
MLE_SEARCHDATA mlesrch;
CHAR szSearchString[64];

/* Obtain the handle of the entry field containing the search string */
hwndEntryFld = WinWindowFromID(hwnd, IDD_SEARCHFIELD);

/* Obtain the search string from the entry field */
WinQueryWindowText(hwndEntryFld,
                   sizeof(szSearchString),
                   szSearchString);

/* Fill the MLE_SEARCHDATA data structure                          */
mlesrch.cb          = sizeof(mlesrch);  /* Structure size          */
mlesrch.pchFind     = szSearchString;   /* Search string           */
mlesrch.pchReplace = NULL;              /* No replacement string   */
mlesrch.cchFind     = 0;                /* Not used                */
mlesrch.cchReplace = 0;                 /* Not used                */
mlesrch.iptStart    = -1;               /* Start at cursor position */
mlesrch.iptStop     = -1;               /* Stop at end of file     */

/* Start the search operation */
WinSendMsg(hwndMle,
           MLM_SEARCH,
           MPFROMLONG(MLFSEARCH_SELECTMATCH),
           MPFROMP(&mlesrch));
```

Figure 3-5. Sample Code for Searching MLE Text

Related Window Messages Received by an MLE Field Control

This section covers the window messages received by a mutliple-line entry field control.

MLM_CHARFROMLINE

This message returns the first insertion point on a given line.

Parameters

param1

lLineNum (LONG)
Line number of interest.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

iptFirst (IPT)

First insertion point on line.

MLM_CLEAR

This message clears the current selection.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulClear (ULONG)
Number of bytes deleted, counted in CF_TEXT format.

MLM_COPY

This message copies the current selection to the clipboard.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

MLM_CUT

This message copies the text that forms the current selection to the clipboard and then deletes it from the MLE control.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

MLM_DELETE

This message deletes text.

Parameters

param1

iptBegin (IPT)
Starting point of deletion.

param2

ulDel (ULONG)
Number of bytes to delete.

Returns

ulSuccess (ULONG)
Number of bytes successfully deleted.

MLM_DISABLEREFRESH

This message disables screen refresh.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

MLM_ENABLEREFRESH

This message enables screen refresh.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

0 Reserved value, 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_EXPORT

This message exports text to a buffer.

Parameters

param1

pBegin (PIPT)
Starting point.

Updated to follow the last character exported.

param2

pCopy (PULONG)
Number of bytes being exported.

Decremented by the number of bytes actually exported.

Returns

ulSuccess (ULONG)
Number of bytes successfully exported.

MLM_FORMAT

This message sets the format to be used for buffer importing and exporting.

Parameters

param1

usFormat (USHORT)
Format to be used for import and export.

MLFIE_CFTEXT	Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.
MLFIE_NOTRANS	Uses LF for line delineation, and guarantees that any text imported into the MLE in this format can be recovered in exactly the same form on export.
MLFIE_WINFMT	(Windows MLE format.) On import, recognizes CR LF as denoting hard line-breaks, and ignores the sequence CR CR LF. On export, uses CR LF to denote a hard line-break and CR CR LF to denote a soft line-break caused by word-wrapping.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

usFormat (USHORT)
Previous format value.

MLM_IMPORT

This message imports text from a buffer.

Parameters

param1

pBegin (PIPT)
Insertion point.

Updated to insertion point following last insert.

param2

ulCopy (ULONG)
Number of bytes in buffer.

Returns

ulSuccess (ULONG)
Number of bytes successfully inserted.

MLM_INSERT

This message deletes the current selection and replaces it with a text string.

Parameters

param1

pchText (PCHAR)
Null-terminated text string.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulCount (ULONG)

Number of bytes actually inserted.

MLM_LINEFROMCHAR

This message returns the line number corresponding to a given insertion point.

Parameters

param1

iptFirst (IPT)
Insertion point of interest.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

lLineNum (LONG)
Line number of insertion point.

MLM_PASTE

This message replaces the text that forms the current selection, with text from the clipboard.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulCopy (ULONG)
Number of bytes transferred, counted in. CF_TEXT format.

MLM_QUERYBACKCOLOR

This message queries the background color.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

lColor (LONG)
Text color.

MLM_QUERYCHANGED

This message queries the changed flag.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Current changed status.

TRUE	Text has changed since the last time that the change flag was cleared.
FALSE	Text has not changed since the last time that the change flag was cleared.

MLM_QUERYFIRSTCHAR

This message queries the first visible character.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

iptFVC (IPT)
First visible character.

MLM_QUERYFONT

This message queries which font is in use.

Parameters

param1

pFattrs (PFATTRS)
Font attribute structure.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
System font indicator.

TRUE	The system font is in use.
FALSE	The system font is not in use.

MLM_QUERYFORMATLINELENGTH

This message returns the number of bytes to end of line after formatting has been applied.

Parameters

param1

iptStart (IPT)
Insertion point to count from.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

iptLine (IPT)
Count of bytes to end of line.

MLM_QUERYFORMATRECT

This message queries the format dimensions and mode.

Parameters

param1

pFormatRect (PPOINTL)
Format dimensions.

The size of the current limiting dimensions.

param2

flFlags (ULONG)
Flags governing interpretation of dimensions.

An array of MLFFMTRECT_* flags defined under the *flFlags* field of the MLM_SETFORMATRECT message.

Returns

ulReserved (ULONG)
Reserved value.

MLM_QUERYFORMATTEXTLENGTH

This message returns the length of a specified range of characters after the current formatting has been applied.

Parameters

param1

iptStart (IPT)
Insertion point to start from.

param2

ulScan (ULONG)
Number of characters to convert to bytes.

0xFFFFFFFF	Convert until end of line
other	Convert specified number of characters.

Returns

ulText (ULONG)
Count of bytes in text after formatting.

MLM_QUERYIMPORTEXPORT

This message queries the current transfer buffer.

Parameters

param1

Buff (PVOID *)
Transfer buffer.

param2

pulLength (PULONG)
Size of transfer buffer in bytes.

Returns

rc (ULONG)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

MLM_QUERYLINECOUNT

This message queries the number of lines of text.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulLines (ULONG)
The number of lines of text.

MLM_QUERYLINELENGTH

This message returns the number of bytes between a given insertion point and the end of line.

Parameters

param1

iptStart (IPT)
Insertion point to count from.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

iptLine (IPT)
Count of bytes to end of line.

MLM_QUERYREADONLY

This message queries the read-only mode.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Current read-only status.

TRUE	Read-only mode is set.
FALSE	Read-only mode is cleared.

MLM_QUERYSEL

This message returns the location of the selection.

Parameters

param1

usQueryMode (USHORT)
Query Mode.

MLFQS_MINMAXSEL	Return both minimum and maximum points of selection in a format compatible with the EM_QUERYSEL message.
MLFQS_MINSEL	Return minimum insertion point of selection.
MLFQS_MAXSEL	Return maximum insertion point of selection.
MLFQS_ANCHORSEL	Return anchor point of selection.
MLFQS_CURSORSEL	Return cursor point of selection.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ReturnCode

sMinSel (SHORT)
Minimum insertion point of selection.

This value is rounded down to 65 535, if necessary.

ReturnCode contains *sMinSel* and *sMaxSel* for a *usQueryMode* of MLFQS_MINMAXSEL.

sMaxSel (SHORT)
Maximum insertion point of selection.

This value is rounded down to 65 535 if necessary.

ReturnCode contains *sMinSel* and *sMaxSel* for a *usQueryMode* of MLFQS_MINMAXSEL.

ipt (IPT)
Requested insertion point.

ReturnCode contains *ipt* for a *usQueryMode* of MLFQS_MINSEL, MLFQS_MAXSEL, MLFQS_ANCHORSEL, or MLFQS_CURSORSEL.

MLM_QUERYSELTEXT

This message copies the currently selected text into a buffer.

Parameters

param1

pchBuff (PCHAR)
Character buffer for text string.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulCount (ULONG)
Number of bytes to put into text string.

MLM_QUERYTABSTOP

This message queries the &pel. interval at which tab stops are placed.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

pixTabset (PIX)
Tab width in &pel.s.

< 0	An error occurred.
Other	The &pel. interval at which tab stops are placed.

MLM_QUERYTEXTCOLOR

This message queries the text color.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

lColor (LONG)
Text color.

MLM_QUERYTEXTLENGTH

This message returns the number of characters in the text.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

iptText (IPT)
Count of text in bytes.

MLM_QUERYTEXTLIMIT

This message queries the maximum number of bytes that a multi-line entry field control can contain.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

lSize (LONG)
Maximum number of bytes allowed in the MLE.

MLM_QUERYUNDO

This message queries the undo or redo operations that are possible.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ReturnCode

usOperation (USHORT)
Operation that can be undone or redone.

0	An undo or redo operation is not possible.
WM_CHAR	A WM_CHAR message, or messages for a simple string of keystrokes, can be undone or redone.
MLM_SETFONT	A MLM_SETFONT message can be undone or redone.
MLM_SETTEXTCOLOR	A MLM_SETTEXTCOLOR message can be undone or redone for both background and foreground color.
MLM_CUT	A MLM_CUT message can be undone or redone.
MLM_PASTE	A MLM_PASTE message can be undone or redone.
MLM_CLEAR	A MLM_CLEAR message can be undone or redone.

rc (BOOL)
Undo or redo indicator.

TRUE	An undo is possible.
FALSE	A redo is possible.

MLM_QUERYWRAP

This message queries the wrap flag.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Wrap flag.

TRUE	Word-wrap enabled
FALSE	Word-wrap disabled.

MLM_RESETUNDO

This message resets the undo state to indicate that no undo operations are possible.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ReturnCode

usOperation (USHORT)
Operation that can be undone or redone.

0	An undo or redo operation is not possible.
WM_CHAR	A WM_CHAR message, or messages for a simple string of keystrokes, can be undone or redone.
MLM_SETFONT	A MLM_SETFONT message can be undone or redone.
MLM_SETTEXTCOLOR	A MLM_SETTEXTCOLOR message can be undone or redone for both background and foreground color.
MLM_CUT	A MLM_CUT message can be undone or redone.
MLM_PASTE	A MLM_PASTE message can be undone or redone.
MLM_CLEAR	A MLM_CLEAR message can be undone or redone.

rc (BOOL)
Undo or redo indicator.

TRUE	An undo is possible.
FALSE	A redo is possible.

MLM_SEARCH

This message searches for a specified text string.

Parameters

param1

ulStyle (ULONG)
Style flags.

MLFSEARCH_CASESENSITIVE	If set, only exact matches are considered a successful match. If not set, any case-combination of the correct characters in the correct sequence is considered a successful match.
MLFSEARCH_SELECTMATCH	If set, the MLE selects the text and scrolls it into view when found, just as if the application had sent an MLM_SETSEL message. This is not done if MLFSEARCH_CHANGEALL is also indicated.
MLFSEARCH_CHANGEALL	Using the MLE_SEARCHDATA structure specified in *pse*, all occurrences of *pchFind* are found, searching from *iptStart* to *iptStop*, and replacing them with *pchReplace*. If this style is selected, the *cchFound* field has no meaning, and the *iptStart* value points to the place where the search stopped, or is the same as *iptStop* because the search has not been stopped at any of the found strings. The current cursor location is not moved. However, any existing selection is deselected.

param2

pse (PMLE_SEARCHDATA)
Search specification structure.

Returns

rc (BOOL)
Success indicator.

TRUE	The search was successful.
FALSE	The search was unsuccessful.

MLM_SETBACKCOLOR

This message sets the background color.

Parameters

param1

lColor (LONG)
Color.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

lOldColor (LONG)
Color previously used.

MLM_SETCHANGED

This message sets or clears the changed flag.

Parameters

param1

usChangedNew (USHORT)
Value to set changed flag to.

TRUE	Changed flag set.
FALSE	Changed flag cleared.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Changed status before message was processed.

TRUE	Text has changed since the last time that the change flag was cleared.
FALSE	Text has not changed since the last time that the change flag was cleared.

MLM_SETFIRSTCHAR

This message sets the first visible character.

Parameters

param1

iptFVC (IPT)
Insertion point to place in top left-hand corner.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_SETFONT

This message sets a font.

Parameters

param1

pFattrs (PFATTRS)
Font attribute structure.

NULL	The system font is set.
other	The specified font is set.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	The font was successfully set.
FALSE	An error occurred.

MLM_SETFORMATRECT

This message sets the format dimensions and mode.

Parameters

param1

pFormatRect (PPOINTL)
New format dimensions.

NULL	A null value sets both dimensions to the current window size.
other	The structure is a pair of LONGs designating the diagonally-opposite corner of the rectangle, assuming 0,0 for the first. Therefore, they are the width and height in &pel.s of the format rectangle. These dimensions are used as the word-wrap and text-size limiting boundaries. Negative values for either dimension cause the MLE to substitute the current window size (the MLE window rectangle minus margins). If the rectangle specified has either, or both, of the limits set, and the size is inadequate to contain the text, *rc* is set to FALSE and the rectangle dimensions are replaced with the overflow amounts.

param2

flFlags (ULONG)
Flags governing interpretation of dimensions.

MLFFMTRECT_MATCHWINDOW	The dimensions of the format rectangle are always to be kept the same as the window size minus the margins. This causes the MLE implicitly to do a MLM_SETFORMATRECT each time the window is resized, and effectively causes any other dimensions to be ignored. Resizing of the window can cause this setting to be automatically negated (see MLN_OVERFLOW).
MLFFMTRECT_LIMITHORZ	The width of any line in the MLE cannot exceed the given horizontal dimension. If word-wrap is on, this limit has no effect. Word-wrap can result in trailing blanks beyond the right limit. These do not cause an overflow notification.
MLFFMTRECT_LIMITVERT	The vertical height of the total text, as displayed, is limited to that which fits totally within the vertical dimension of the format rectangle.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_SETIMPORTEXPORT

This message sets the current transfer buffer.

Parameters

param1

pBuff (PCHAR)
Transfer buffer.

param2

ulLength (ULONG)
Size of transfer buffer in bytes.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_SETREADONLY

This message sets or clears read-only mode.

Parameters

param1

usReadOnly (USHORT)
New read-only value.

TRUE	Read-only mode set.
FALSE	Read-only mode cleared.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Previous read-only value.

TRUE	Read-only mode was set.
FALSE	Read-only mode was cleared.

MLM_SETSEL

This message sets a selection.

Parameters

param1

iptAnchor (IPT)
Insertion point for new anchor point.

param2

iptCursor (IPT)
Insertion point for new cursor point.

Returns

rc (BOOL)
Success indicator.

TRUE	Selection successfully set
FALSE	An error occurred.

MLM_SETTABSTOP

This message sets the &pel. interval at which tab stops are placed.

Parameters

param1

pixTab (PIX)
&Pel. interval for tab stops.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

pixTabset (PIX)
Success indicator.

< 0 An error occurred.
Other The value to which the width was set.

MLM_SETTEXTCOLOR

This message sets the text color.

Parameters

param1

lColor (LONG)
Color.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

lOldColor (LONG)
Color previously used.

MLM_SETTEXTLIMIT

This message sets the maximum number of bytes that a multi-line entry field control can contain.

Parameters

param1

lSize (LONG)
Maximum number of characters in MLFIE_NOTRANS format.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulFit (ULONG)
Success indicator.

0	Successful completion. Current text fits within the new limit.
Other	The number of bytes by which the current text exceeds the proposed limit. The limit is not changed.

MLM_SETWRAP

This message sets the wrap flag.

Parameters

param1

usWrap (USHORT)
New value for wrap flag.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_UNDO

This message performs any available undo operation.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (USHORT)
: Success indicator.

TRUE	An undo operation was performed.
FALSE	No undo operation was performed.

Related Window Messages Sent by an MLE Field Control

This section covers the window messages sent by a mutliple-line entry field control.

WM_BUTTON1DBLCLK

This message occurs when the operator presses button 1 of the pointing device twice within a specified time, as detailed below.

Parameters

param1

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

fsHitTestres (USHORT)
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see WM_HITTEST.

fsflags (USHORT)
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.

Returns

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1DOWN

This message occurs when the operator presses pointer button one.

Parameters

param1

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

fsHitTestres (USHORT)
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see WM_HITTEST.

fsflags (USHORT)
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.

Returns

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1UP

This message occurs when the operator releases button 1 of the pointing device.

Parameters

param1

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

fsHitTestres (USHORT)
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see WM_HITTEST.

fsflags (USHORT)
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.

Returns

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_CHAR

This message is sent when an operator presses a key.

Parameters

param1

fsflags (USHORT)
Keyboard control codes.

KC_CHAR	Indicates that *usch* value is valid.
KC_SCANCODE	Indicates that *ucscancode* is valid. Generally, this is set in all WM_CHAR messages generated from actual operator input. However, if the message has been generated by an application that has issued the WinSetHook function to filter keystrokes, or posted to the application queue, this may not be set.
KC_VIRTUALKEY	Indicates that *usvk* is valid. Normally *usvk* should be given precedence when processing the message. **Note:** For those using hooks, when this bit is set, KC_SCANCODE should usually be set as well.
KC_KEYUP	The event is a key-up transition; otherwise it is a down transition.
KC_PREVDOWN	The key has been previously down; otherwise it has been previously up.
KC_DEADKEY	The character code is a dead key. The application is responsible for displaying the glyph for the dead key without advancing the cursor.
KC_COMPOSITE	The character code is formed by combining the current key with the previous dead key.
KC_INVALIDCOMP	The character code is not a valid combination with the preceding dead key. The application is responsible for advancing the cursor past the dead-key glyph and then, if the current character is not a space, sounding the alarm and displaying the new character code.
KC_LONEKEY	Indicates if the key is pressed and released without any other keys being pressed or released between the time the key goes down and up.
KC_SHIFT	The SHIFT state is active when key press or release occurred.
KC_ALT	The ALT state is active when key press or release occurred.

KC_CTRL	The CTRL state was active when key press or release occurred.

ucrepeat (UCHAR)
Repeat count.

ucscancode (UCHAR)
Hardware scan code.

A keyboard-generated value that identifies the keyboard event. This is the raw scan code, not the translated scan code.

param2

usch (USHORT)
Character code.

The character value translation of the keyboard event resulting from the current code page that would apply if the CTRL or ALT keys were not depressed.

usvk (USHORT)
Virtual key codes.

A virtual key value translation of the keyboard event resulting from the virtual key code table. The low-order byte contains the **vk** value, and the high-order byte is always set to zero by the standard translate table.

0 This value applies if *fsflags* does not contain KC_VIRTUALKEY.

Returns

rc (BOOL)
Processed indicator.

TRUE Message processed
FALSE Message ignored.

WM_ENABLE

This message notifies a windows of a change to its enable state.

Parameters

param1

usnewenabledstate (USHORT)
New enabled state indicator.

TRUE The window was set to the enabled state.
FALSE The window was set to the disabled state.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_MOUSEMOVE

This message occurs when the pointing device pointer moves.

Parameters

param1

sxMouse (SHORT)
&Pdev. x-coordinate.

syMouse (SHORT)
&Pdev. y-coordinate.

param2

uswHitTest (USHORT)
Message result.

Zero	A pointing device capture is currently in progress
Other	The result of the WM_HITTEST message.

fsflags (USHORT)
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed

Returns

rc (BOOL)
Processed indicator.

TRUE	The window procedure did process the message.
FALSE	The window procedure did not process the message.

WM_QUERYWINDOWPARAMS

This message occurs when an application queries the window parameters.

Parameters

param1

pwndparams (PWNDPARAMS)
Window parameter structure.

This points to a window parameter structure; see "WNDPARAMS" on page 3-58.

The valid values of *fsStatus* are WPM_CCHTEXT, WPM_TEXT, WPM_CBCTLDATA, and WPM_CTLDATA.

The flags in *fsStatus* are cleared as each item is processed. If the call is successful, *fsStatus* is 0. If any item has not been processed, the flag for that item is still set.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the window parameters.

Parameters

param1

pwndparams (PWNDPARAMS)
Window parameter structure.

This points to a window parameter structure; see "WNDPARAMS" on page 3-58.

The valid values of *fsStatus* are WPM_TEXT and WPM_CTLDATA.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

Related Notification Message

This section covers the notification message that is related to multiple-line entry field controls.

WM_CONTROL (in Multiline Entry Fields)

For the cause of this message, see "WM_CONTROL" on page 5-15.

Parameters

param1

usid (USHORT)
Control window identity.

usnotifycode (USHORT)
Notify code.

MLN_TEXTOVERFLOW
: A key stroke causes the amount of text to exceed the limit on the number of bytes of data (refer to MLM_SETTEXTLIMIT). The parameter contains the number of bytes of data which would not fit within the current text limit. For character key strokes this can be 1 or 2 (DBCS). For Shift+Ins (paste) it can be any amount up to the paste limit.

  The default *rc* of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

  An *rc* of TRUE implies that corrective action has been taken (such as deleting existing text or raising the limit) and the WM_CHAR (in Multiline Entry Fields) should be reprocessed as if just entered.

MLN_PIXHORZOVERFLOW
: A key stroke causes the size of the display bit map to exceed the horizontal limit of the format rectangle (refer to MLM_SETFORMATRECT). The parameter contains the number of &pel.s that would not fit within the current text limit.

  The default *rc* of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

  An *rc* of TRUE implies that corrective action has been taken (such as changing to a smaller font or raising the limit) and the WM_CHAR (in Multiline Entry Fields) should be reprocessed as if just entered.

MLN_PIXVERTOVERFLOW

A key stroke causes the size of the display bit map to exceed the vertical limit of the format rectangle (refer to MLM_SETFORMATRECT). The parameter contains the number of &pel.s that would not fit within the current text limit.

The default *rc* of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An *rc* of TRUE implies that corrective action has been taken (such as changing to a smaller font or raising the limit) and the WM_CHAR (in Multiline Entry Fields) should be reprocessed as if just entered.

MLN_OVERFLOW

An action other than entry of a key stroke causes a condition involving the text limit or format rectangle limit, such that either the limit becomes inadequate to contain the text or the text exceeds the limit.

This can be caused by:

MLM_SETWRAP
MLM_SETTABSTOP
MLM_SETFONT
MLM_IMPORT
MLM_PASTE
MLM_CUT
MLM_UNDO
MLM_DELETE
WM_SIZE.

MLN_HSCROLL

Indicates that the MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.

MLN_VSCROLL

Indicates that the MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.

MLN_CHANGE

Signals that the text has changed. This notification is sent whenever any text change occurs.

MLN_UNDOOVERFLOW

Signals that the text change operation, which could normally be undone, cannot be undone because the amount of text involved exceeds the undo capability. This includes text entry, deletion, cutting, and pasting.

MLN_CLPBDFAIL	Signals that a clipboard operation failed.
MLN_MEMERROR	Signals that the required storage cannot be obtained. The action that results in the increased storage requirement fails.
MLN_SETFOCUS	Sent whenever the MLE window receives the input focus.
MLN_KILLFOCUS	Sent whenever the MLE window loses the input focus.
MLN_MARGIN	Whenever the user moves the mouse into the left, right top, or bottom margins, this message is sent to the owner of the window. If the owner returns an *rc* of TRUE, the mouse move is assumed to have been processed by the owner and no further action need be taken. If the owner returns an *rc* of FALSE, the MLE performs a default action appropriate to each different mouse action. The exceptions to this are all mouse messages that occur after a button-down inside the margin, until and including the matching button-up. Conceptually the drag (button-down until button-up) is a single macro event. Therefore, if FALSE is returned for a button-down event, no further margin notifications are given until after the drag has ended (button-up). **Note:** If the application receives a notification of button-down in the margin and processes it, it must capture the mouse until the button-up event.
MLN_SEARCHPAUSE	This notification is sent periodically by the MLE, while an MLM_SEARCH message is being processed, to give an application the opportunity to stop excessively long searches, and to provide search progress information. The owner window can respond either with TRUE or FALSE. FALSE causes the MLE to continue searching; TRUE causes the MLE to stop the search immediately. For further information, see MLM_SEARCH

param2

ulOver (ULONG)
Number of bytes that do not fit.

param2 contains *ulOver* for a *usnotifycode* of MLN_TEXTOVERFLOW.

pixOver (PIX)
Linear distance of overflow in &pel.s.

param2 contains *pixOver* for a *usnotifycode* of MLN_PIXHORZOVERFLOW or MLN_PIXVERTOVERFLOW.

pErrInfo (POVERFLOW)
Overflow error information structure.

param2 contains *pErrInfo* for a *usnotifycode* of MLN_OVERFLOW.

The *afErrInd* field of the MLEOVERFLOW structure can take one or more of the following values:

MLFEFR_RESIZE	The window is resized, and the format rectangle is tied to the window size and limited either horizontally, vertically, or both. The implicit change of the format rectangle to the new size does not contain the text. The format rectangle is made static at the previous size, and the MLESFR_MATCHWINDOW style is turned off until set again by the application. This is done in response to a WM_SIZE message, and therefore the multi-line entry field does not forward the return value from this notification message.
MLFEFR_TABSTOP	A tab stop location change is requested, and the text is limited either horizontally, vertically, or both. Changing the tab stops causes the text to exceed the limit. The tab stop change is rejected.
MLFEFR_FONT	A font change is requested, and the text is limited either horizontally, vertically, or both. Changing the font causes the text to exceed the limit. The font change is rejected.
MLFEFR_WORDWRAP	The word-wrap state is requested to be changed, and the text is limited either horizontally, vertically, or both. Wrapping the text differently exceeds the limit, and the request is rejected. This happens in situations where the horizontal limit is not set, there are lines exceeding it, and word-wrap is being changed from off to on, such that it creates soft line breaks resulting in increased vertical size. This happens if word-wrap is being changed from on to off, and there is at least one line created by a soft line-break, such that when that line-break is removed, the full line (up to the hard line break) exceeds the horizontal limit.

MLFEFR_TEXT	Text is changed by MLM_IMPORT, MLM_PASTE, MLM_CUT, MLM_UNDO, or MLM_DELETE, and the text is limited either horizontally, vertically, or both within the format rectangle. The change causes the text to exceed the format rectangle in a dimension that is limited. For example, Delete and EOL joins text from two lines into one line long enough to exceed the horizontal limit.
MLFETL_TEXTBYTES	Text is changed by MLM_IMPORT MLM_PASTE, or MLM_UNDO, and the text is limited to a maximum number of bytes. The change causes the text to exceed that maximum.

ulErrInd (ULONG)
Clipboard fail flag.

param2 contains *ulErrInd* for a *usnotifycode* of MLN_CLPBDFAIL.

MLFCPBD_TOOMUCHTEXT	Text amount exceeds clipboard capacity
MLFCPBD_CLPBDERROR	A clipboard error occurred.

pmrg (PMARGSTRUCT)
Margin structure.

param2 contains *pmrg* for a *usnotifycode* of MLN_MARGIN.

The left and right margins are defined as going all the way to the top and bottom such that the top and bottom margins are contained between them. Therefore, the corners are included in the sides.

usMouMsg contains the mouse message that signals the event.

iptNear contains the insertion point of the nearest point in the text. For situations where the nearest location is beyond the end of a line, the insertion point for the end of the line is returned. (The EOL character is considered to be beyond the end of the line.)

iptSearchedTo (IPT)
Current insertion point of search.

param2 contains *iptSearchedTo* for a *usnotifycode* of MLN_SEARCHPAUSE.

ulReserved (ULONG)
Reserved value, should be 0.

param2 contains *ulReserved* for a *usnotifycode* of MLN_HSCROLL, MLN_VSCROLL, MLN_CHANGE, MLN_UNDOOVERFLOW, MLN_MEMERROR, MLN_SETFOCUS, or MLN_KILLFOCUS.

Returns

ReturnCode

rc (BOOL)
Action taken by application.

ReturnCode contains *rc* for a *usnotifycode* of MLN_TEXTOVERFLOW, MLN_PIXHORZOVERFLOW, MLN_PIXVERTOVERFLOW, MLN_MARGIN, or MLN_SEARCHPAUSE.

TRUE — The multiline entry field control assumes that appropriate action has been taken by the application. Appropriate action depends on the MLN_* notification code, and is documented under the *usnotifycode* field.

FALSE — The multiline entry field control assumes that the application has ignored this WM_CONTROL (in Multiline Entry Fields) message, and takes action appropriate to the MLN_* notification code, as documented under the *usnotifycode* field.

ulReserved (ULONG)
Reserved value, should be 0.

ReturnCode contains *ulReserved* for a *usnotifycode* of MLN_OVERFLOW, MLN_HSCROLL, MLN_VSCROLL, MLN_CHANGE, MLN_UNDOOVERFLOW, MLN_CLPBDFAIL, MLN_MEMERROR, MLN_SETFOCUS, or MLN_KILLFOCUS.

Related Data Structures

This section covers the data structures that are related to multiple-line entry field controls.

MLECTLDATA

Multiline entry-field (MLE) control data structure.

Syntax

```
typedef struct _MLECTLDATA {
USHORT       cbCtlData;
USHORT       afIEFormat;
ULONG        cchText;
IPT          iptAnchor;
IPT          iptCursor;
LONG         cxFormat;
LONG         cyFormat;
ULONG        afFormatFlags;
 } MLECTLDATA;

typedef MLECTLDATA *PMLECTLDATA;
```

Fields

cbCtlData (USHORT)
Length of control data in bytes.

afIEFormat (USHORT)
Import/export format.

This sets the initial import/export format. Setting this value via control data is considered identical to setting it through the MLM_FORMAT message. The same constants apply here. The default is MLE_CFTEXT.

cchText (ULONG)
Text limit.

The maximum amount of text allowed in the MLE. This value is interpreted identically to the parameter of MLM_SETTEXTLIMIT. A negative value indicates that the length is considered unbounded.

iptAnchor (IPT)
Selection anchor point.

iptCursor (IPT)
Selection cursor point.

The *iptAnchor* and *iptCursor* parameters identify the beginning and ending points, respectively, of the selection. These values may range from 0 through the length of the text. The default is 0,0 and can be indicated by entering 0,0.

cxFormat (LONG)
Formatting-rectangle width in &pel.s.

cyFormat (LONG)
Formatting-rectangle height in &pel.s.

The *cxFormat* and *cyFormat* parameters identify the dimensions in &pel.s of the formatting rectangle, as can be set by the MLM_SETFORMATRECT message. These values are considered identical to the two fields in the format rectangle structure referenced in that message, and the interpretation of the values in these fields is governed by the *afFormatFlags* field.

The default is the window size in both dimensions, and can be indicated by 0 values.

afFormatFlags (ULONG)
Format flags.

These flags govern the interpretation of the *cxFormat* and *cyFormat* fields, just as in the MLM_SETFORMATRECT message. The flag values defined there are also valid in this field. The default is unlimited in both directions, and is of varying size to match the window size.

MLEMARGSTRUCT

Multiline entry-field margin information.

Syntax

```
typedef struct _MLEMARGSTRUCT {
USHORT       afMargins;
USHORT       usMouMsg;
IPT          iptNear;
 } MLEMARGSTRUCT;

typedef MLEMARGSTRUCT *PMARGSTRUCT;
```

Fields

afMargins (USHORT)
Margin in which the event occurred.

The left and right margins are defined as including the corners at the top and bottom, and the top and bottom margins are contained between them. Therefore, the corners are included in the sides.

MLFMARGIN_LEFT
MLFMARGIN_RIGHT
MLFMARGIN_TOP
MLFMARGIN_BOTTOM

usMouMsg (USHORT)
Message identity of the original mouse event.

iptNear (IPT)
Insertion point nearest to the margin event.

MLEOVERFLOW

Overflow error structure for multiline entry field.

Syntax

```
typedef struct _MLEOVERFLOW {
ULONG       afErrInd;
LONG        nBytesOver;
LONG        pixHorzOver;
LONG        pixVertOver;
 } MLEOVERFLOW;

typedef MLEOVERFLOW *POVERFLOW;
```

Fields

afErrInd (ULONG)
One or more EFR_* flags.

nBytesOver (LONG)
Number of bytes over the limit.

pixHorzOver (LONG)
Number of &pel.s over the horizontal limit.

pixVertOver (LONG)
Number of &pel.s over the vertical limit.

MLE_SEARCHDATA

Search structure for multiline entry field.

Syntax

```
typedef struct _SEARCH {
USHORT       cb;
PCHAR        pchFind;
PCHAR        pchReplace;
SHORT        cchFind;
SHORT        cchReplace;
IPT          iptStart;
IPT          iptStop;
USHORT       cchFound;
 } MLE_SEARCHDATA;

typedef MLE_SEARCHDATA *PMLE_SEARCHDATA;
```

Fields

cb (USHORT)
Size of structure.

pchFind (PCHAR)
String to search for.

pchReplace (PCHAR)
String to replace with.

cchFind (SHORT)
Length of *pchFind* string.

cchReplace (SHORT)
Length of *pchReplace* string.

iptStart (IPT)
Point at which to start search, or point where string was found.

Non-negative	Point at which to start search.
Negative	Start search from current cursor location.

iptStop (IPT)
Point at which to stop search.

Non-negative	Point at which to stop search.
Negative	Stop search at end of text.

cchFound (USHORT)
Length of string found at *iptStart*.

WNDPARAMS

Window parameters.

Syntax

```
typedef struct _WNDPARAMS {
ULONG      fsStatus;
ULONG      cchText;
PSZ        pszText;
ULONG      cbPresParams;
PVOID      pPresParams;
ULONG      cbCtlData;
PVOID      pCtlData;
 } WNDPARAMS;

typedef WNDPARAMS *PWNDPARAMS;
```

Fields

fsStatus (ULONG)
Window parameter selection.

Identifies the window parameters that are to be set or queried:

WPM_CBCTLDATA	Window control data length
WPM_CCHTEXT	Window text length
WPM_CTLDATA	Window control data
WPM_PRESPARAMS	Presentation parameters
WPM_TEXT	Window text.

cchText (ULONG)
Length of window text.

pszText (PSZ)
Window text.

cbPresParams (ULONG)
Length of presentation parameters.

pPresParams (PVOID)
Presentation parameters.

cbCtlData (ULONG)
Length of window class specific data.

pCtlData (PVOID)
Window class specific data.

Summary

Following are tables that describe the OS/2 window messages, notification message, notification codes, and data structures used with multiple-line entry field control:

Table 3-3 (Page 1 of 2). Window Messages Received by an MLE

Message Name	Description
MLM_CHARFROMLINE	Returns the first insertion point on a given line.
MLM_CLEAR	Clears the current selection.
MLM_COPY	Copies the current selection to the clipboard.
MLM_CUT	Copies the text that forms the current selection to the clipboard, then deletes the text from the MLE field control.
MLM_DELETE	Deletes text.
MLM_DISABLEREFRESH	Disables screen refresh.
MLM_ENABLEREFRESH	Enables screen refresh.
MLM_EXPORT	Exports text to a buffer.
MLM_FORMAT	Sets the format to be used for buffer importing and exporting.
MLM_IMPORT	Imports text from a buffer.
MLM_INSERT	Deletes the current selection and replaces it with a text string.
MLM_LINEFROMCHAR	Returns the line number corresponding to a given insertion point.
MLM_PASTE	Replaces the text that forms the current selection with text from the clipboard.
MLM_QUERYBACKCOLOR	Queries the background color.
MLM_QUERYCHANGED	Queries the changed flag.
MLM_QUERYFIRSTCHAR	Queries the first visible character.
MLM_QUERYFONT	Queries which font is in use.
MLM_QUERYFORMATLINELENGTH	Returns the number of bytes to end of line after formatting is applied.
MLM_QUERYFORMATRECT	Queries the format dimensions and mode.
MLM_QUERYFORMATTEXTLENGTH	Returns the length of a specified range of characters after the current formatting is applied.
MLM_QUERYIMPORTEXPORT	Queries the current transfer buffer.
MLM_QUERYLINECOUNT	Queries the number of lines of text.
MLM_QUERYLINELENGTH	Returns the number of bytes between a given insertion point and the end of line.
MLM_QUERYREADONLY	Queries the read-only mode.
MLM_QUERYSEL	Returns the location of the selection.

Table 3-3 (Page 2 of 2). Window Messages Received by an MLE

Message Name	Description
MLM_QUERYSELTEXT	Copies the currently selected text into a buffer.
MLM_QUERYTABSTOP	Queries the pel interval at which tab stops are placed.
MLM_QUERYTEXTCOLOR	Queries the text color.
MLM_QUERYTEXTLENGTH	Returns the number of characters in the text.
MLM_QUERYTEXTLIMIT	Queries the maximum number of bytes that a multiple-line entry field control can contain.
MLM_QUERYUNDO	Queries the possible undo or redo operations.
MLM_QUERYWRAP	Queries the wrap flag.
MLM_RESETUNDO	Resets the undo state to indicate the no undo operations are possible.
MLM_SEARCH	Searches for a specified text string.
MLM_SETBACKCOLOR	Sets the background color.
MLM_SETCHANGED	Sets or clears the changed flag.
MLM_SETFIRSTCHAR	Sets the first visible character.
MLM_SETFONT	Sets a font.
MLM_SETFORMATRECT	Sets the format dimensions and mode.
MLM_SETIMPORTEXPORT	Sets the current transfer buffer.
MLM_SETREADONLY	Sets or clears read-only mode.
MLM_SETSEL	Sets a selection.
MLM_SETTABSTOP	Sets the pel interval at which tab stops are placed.
MLM_SETTEXTCOLOR	Sets the text color.
MLM_SETTEXTLIMIT	Sets the maximum number of bytes that a multiple-line entry field control can contain.
MLM_SETWRAP	Sets the wrap flag.
MLM_UNDO	Performs any available undo operations.

Table 3-4 (Page 1 of 2). Window Messages Sent by an MLE

Message Name	Description
WM_BUTTON1DBLCLK	Occurs when the user presses mouse button 1 twice within a specified time.
WM_BUTTON1DOWN	Occurs when the user presses mouse button 1.
WM_BUTTON1UP	Occurs when the user releases mouse button 1.
WM_CHAR	Sent when the user presses a key.
WM_ENABLE	Sets the state of the MLE field.
WM_MOUSEMOVE	Occurs when the pointing device pointer moves.

Table 3-4 (Page 2 of 2). Window Messages Sent by an MLE	
Message Name	**Description**
WM_QUERYWINDOWPARAMS	Occurs when an application queries the entry field control window parameters.
WM_SETWINDOWPARAMS	Occurs when an application sets or changes the entry field control window parameters.

Table 3-5. MLE Notification Message	
Message Name	**Description**
WM_CONTROL	Occurs when an MLE field control has a significant event to notify to its owner.

Table 3-6. MLE Notification Codes	
Code Name	**Description**
MLN_CHANGE	Indicates that the contents of the MLE field have changed.
MLN_CLPBDFAIL	Indicates that a clipboard operation failed.
MLN_HSCROLL	Indicates that the MLE text is about to scroll horizontally.
MLN_KILLFOCUS	Indicates that the MLE field lost the input focus.
MLN_MARGIN	Indicates that the mouse moved across the MLE field margin.
MLN_MEMERROR	Indicates that the MLE field control cannot allocate enough memory to perform the requested operation.
MLN_OVERFLOW	Indicates that the specified MLE operation would overflow the field's text limit or the format rectangle.
MLN_PIXHORZOVERFLOW	Indicates that the user entered more text than could fit horizontally in the MLE field.
MLN_PIXVERTOVERFLOW	Indicates that the user entered more text than could fit vertically in the MLE field.
MLN_SEARCHPAUSE	Indicates that the MLE field control paused during a search operation initiated by an MLM_SEARCH message.
MLN_SETFOCUS	Indicates that the MLE field received the input focus.
MLN_TEXTOVERFLOW	Indicates that the user or application attempted to exceed the text limit of the MLE field.
MLN_UNDOOVERFLOW	Indicates that the MLE field control cannot undo a text change because the undo operation involves too much text.
MLN_VSCROLL	Indicates that the MLE text is about to scroll vertically.

Table 3-7. MLE Data Structures

Data Structure Name	Description
MLECTLDATA	MLE field control data structure.
MLEMARGSTRUCT	MLE field margin information data structure.
MLEOVERFLOW	MLE field overflow error data structure.
MLE_SEARCHDATA	MLE field search data structure.
WNDPARAMS	Window parameters data structure.

Chapter 4. Spin Button Controls

A *spin button* control (WC_SPINBUTTON window class) is a visual component that gives users quick access to a finite set of data by letting them select from a scrollable ring of choices. Because the user can see only one item at a time, a spin button should be used only with data that is intuitively related, such as a list of the months of the year, or an alphabetic list of cities or states. This chapter explains when and how to use spin buttons in PM applications.

About Spin Button Controls

A *spin button* consists of at least one spin field that is a single-line entry (SLE) field, and up and down arrows that are stacked on top of one another. These arrows are positioned to the right of the SLE field. Figure 4-1 shows an example of spin button.

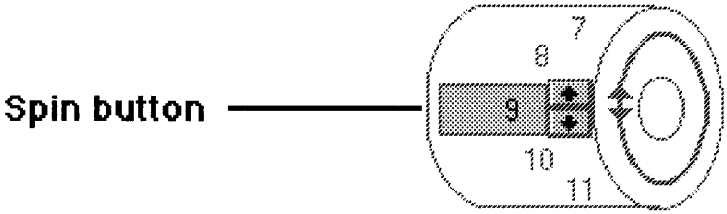

Figure 4-1. Spin Button Example

You can create multi-field spin buttons for those applications in which users must select more than one value. For example, in setting a date, the spin button control can provide individual fields for setting the month, day, and year. The first spin field in the spin button could contain a list of months; the second, a list of numbers; and the third, a list of years.

The application uses a multi-field spin button by creating one master component that contains a spin field and the spin arrows, and servant components that contain only spin fields. The spin buttons are created at component initialization. The servant components are passed a handle to the master component in a message. When a servant spin field has the focus, it is spun by the arrows in the master component.

The list of values in a spin button entry field can be an array of data or a list of consecutive integers, defined by an upper and a lower limit.

Using Spin Button Controls

This section describes how to create a spin button control.

Creating a Spin Button

A spin button is created as a public window class by using WinCreateWindow, with a class style of WC_SPINBUTTON and a window style of WS_VISIBLE. These are joined with any of the spin button style flags by using a logical OR (|). The spin button style flags let you specify:

- Character input restrictions (*none, numeric, read-only*)
- Presentation of the data in the spin field (*left-justified, right-justified, centered*)
- Presence or absence of a border around the spin field
- Spin speed
- Zero-padding of numeric spin fields.

The placement and width of the spin button component are specified as parameters in WinCreateWindow.

The upper and lower limits of numeric fields, the value array pointer for arrays of strings, and the initial value in the spin field are all set by messages sent from the application to the component.

You can destroy the spin button component window using WinDestroyWindow when finished. The component handle that was returned when the spin button was created is the input parameter to WinDestroyWindow. The sample code illustrated in Figure 4-2 shows an example of how to create a spin button.

```
ULONG      ulSpinStyle;              /* Spin Button style           */
HWND       hwndSpin;                 /* Spin Button window handle   */

/*******************************************************************/
/* Set the SPBS_* style flags.                                     */
/*******************************************************************/
ulSpinStyle = SPBS_MASTER       |    /* Spin button has its own     */
                                     /* buttons,                    */
              SPBS_NUMERICONLY |     /* and it only holds numbers   */
              SPBS_JUSTRIGHT    |    /* that are right justified,   */
              SPBS_FASTSPIN;         /* and it spins faster as      */
                                     /* the arrows are held down    */
```

Figure 4-2 (Part 1 of 2). Sample Code for Creating a Spin Button

```
/**********************************************************************/
/* Create the Spin Button control window.                             */
/* The handle of the window is returned in hwndSpin.                  */
/**********************************************************************/
hwndSpin = WinCreateWindow (
                     hwndClient,      /* Parent window handle         */
                     WC_SPINBUTTON,   /* Spin Button window class name */
                     (PSZ)NULL,       /* No window text               */
                     ulSpinStyle,     /* Spin Button styles variable  */

                     (LONG)10,        /* X coordinate                 */
                     (LONG)10,        /* Y coordinate                 */
                     (LONG)150,       /* Window width                 */
                     (LONG)50,        /* Window height                */
                     hwndClient,      /* Owner window handle          */
                     HWND_TOP,        /* Sibling window handle        */
                     ID_SPINBUTTON,   /* Spin Button control window ID */
                     (PVOID)NULL      /* No control data structure    */
                     (PVOID)NULL);    /* No presentation parameters   */

/**********************************************************************/
/* Set the limits of the Spin Button control, since it has a style    */
/* of SPBS_NUMERICONLY.                                               */
/**********************************************************************/
WinSendMsg (hwndSpin,                 /* Spin Button window handle    */
            SPBM_SETLIMITS,           /* Set limits message           */
            (MPARAM)1000,             /* Spin Button maximum setting  */
            (MPARAM)0);               /* Spin Button minimum setting  */

/**********************************************************************/
/* Set the initial value of the Spin Button.                          */
/**********************************************************************/
WinSendMsg (hwndSpin,                 /* Spin Button window handle    */
            SPBM_SETCURRENTVALUE,     /* Set current value message    */
            (MPARAM)100,              /* Spin Button initial value    */
            (MPARAM)NULL);            /* Reserved value               */

/**********************************************************************/
/* Because all items have been set, make the control visible.         */
/**********************************************************************/
WinShowWindow (hwndSpin,              /* Spin Button window handle    */
               TRUE);                 /* Make the window visible      */
```

Figure 4-2 (Part 2 of 2). Sample Code for Creating a Spin Button

Graphical User Interface Support for Spin Button Controls

Users can interact with the spin button using either the keyboard or a pointing device, such as a mouse, as follows:

- Using the select button (button 1) on the pointing device, users first give focus to the spin field they want to change, and then click on either the Up Arrow or Down Arrow until the value they want is displayed in the spin field.
- Using a keyboard, users press the:
  - Up Arrow and Down Arrow keys to see the choices
  - Left Arrow and Right Arrow keys to move the cursor left and right within a spin field
  - Home and End keys to move the cursor to the first and last characters in a spin field
  - Tab and Shift+Tab keys to move the input focus from one field to another in multi-field spin buttons.

Users can view the values in a spin field one at a time, or they can rapidly scroll a list by keeping either the Up or Down Arrow keys pressed. When a spin button is not read-only, users can advance quickly to the value they want to set in a spin field by typing over the value currently displayed.

Related Window Messages

This section covers the window messages that are related to spin button controls.

SPBM_OVERRIDESETLIMITS

This message causes the component to set or reset numeric limits.

Parameters

param1

lUpLimit (LONG)
Upper limit.

param2

lLowLimit (LONG)
Lower limit.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

SPBM_QUERYLIMITS

This message enables an application to query the limits of a numeric spin field.

Parameters

param1

plUpLimit (PLONG)
Pointer to a LONG that will receive the returned upper limit.

param2

plLowLimit (PLONG)
Pointer to a LONG that will receive the returned lower limit.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_QUERYVALUE

This message causes the component to show the value in the spin field.

Parameters

param1

pStorage (PVOID)
Place for returned value.

A place for the returned value. This value is either the address of a string or the address of a long variable.

If the *usBufSize* is 0, *param1* is assumed to be an address of a long variable.

If *param1* is Other, it is assumed to be an address of a string.

NULL	Causes the spin button to process the reset or update as specified, but it will not try to return a value to the application.
Other	The address where the value is returned.

param2

usBufSize (USHORT)
Buffer size.

If *usBufSize* is too small to return all of the text, the spin button returns as much of the text as it can.

0	The spin button assumes that *param1* is the address of a long variable. If the data in the spin button is spinning between an upper and lower limit, the current value is passed back in the variable. If the data in the spin button is in an array, the index of the current array value (or last valid value) is passed back in the variable.
Other	The spin button assumes that *param1* is the address of a string. The information passed back in the string is dependent upon the flags in the *usValue* parameter.

usValue (USHORT)
Update/reset value.

Controls how the spin field is updated.

SPBQ_UPDATEIFVALID	Update the contents of the spin field if the value is valid. This is the default. Specifying this flag on a query will *not* update the contents of the spin field if it is *exactly* the same as an item in the spin button list. If an item in the list is Monday, specifying SPBQ_UPDATEIFVALID updates the spin field contents when MONDAY, monday, or mONDAY are typed, but not when Monday is typed. This prevents recursion if the application checks for the validity each time a SPBN_CHANGE message is sent from the component.
SPBQ_ALWAYSUPDATE	Update the contents of the spin field if the value is valid. Reset the contents of the spin field to the last valid value if the field contains data that is not valid. If the spin button is spinning numbers between an upper and a lower limit, and the content of the spin field is a valid number that is out of range, the spin button does not reset itself to the last valid value. It sets the current position at the upper limit when the out-of-range number specified is above the upper limit. It sets the current position at the lower limit when the out-of-range number is below the lower limit. When the current value is changed, the return of the query message is still FALSE.
SPBQ_DONOTUPDATE	Do not update the contents of the spin field, even if the value is valid.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

SPBM_SETARRAY

This message causes the component to set or reset the array of data.

Parameters

param1

pStrl (PSZ)
Pointer to the new array of values.

param2

usItems (USHORT)
Number of items in the array.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SETCURRENTVALUE

This message causes the component to set or reset the current numeric value or array index.

Parameters

param1

lValue (LONG)
Array value or index.

Current value or index of array.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SETLIMITS

This message causes the component to set or reset numeric limits.

Parameters

param1

lUpLimit (LONG)
Upper limit.

param2

lLowLimit (LONG)
Lower limit.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SETMASTER

This message causes the component to identify its master.

Parameters

param1

hwnd (HWND)
Handle of master component.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SETTEXTLIMIT

This message sets the maximum number of characters allowed in a spin field.

Parameters

param1

usLimit (USHORT)
Character limit.

Number of characters to allow.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SPINDOWN

This message causes the component to show the previous value (spin backward).

Parameters

param1

ulItem (ULONG)
Number of values to spin down.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SPINUP

This message causes the component to show the next value (spin forward).

Parameters

param1

ulItem (ULONG)
Number of values to spin up.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Related Notification Message

This section covers the notification message that is related to spin button controls.

WM_CONTROL (in Spin Button Controls)

For the cause of this message, see "WM_CONTROL" on page 5-15.

Parameters

param1

id (USHORT)
Identity of the spin button component window.

notifycode (USHORT)
Notification code.

SPBN_UPARROW	Tells the application that the Up Arrow was clicked on, or the Up Arrow key was pressed.
SPBN_DOWNARROW	Tells the application that the Down Arrow was clicked on, or the Down Arrow key was pressed.
SPBN_SETFOCUS	Tells the application which spin field was selected.
SPBN_KILLFOCUS	Tells the application when the spin field loses focus.
SPBN_ENDSPIN	Tells the application that the user released the select button or one of the arrow keys while spinning a button.
SPBN_CHANGE	Tells the application that the contents of the spin field changed.

param2

hwnd (HWND)
Window handle.

The interpretation of this handle is dependent upon the following notification codes:

- SPBN_UPARROW, SPBN_DOWNARROW, and SPBN_ENDSPIN.

  The *param2* parameter is the handle to the currently selected spin field in a particular master-servant setup. If either the Up or Down Arrow is clicked on and none of a spin button's servants are currently selected, the master will return a handle to itself.

- SPBN_SETFOCUS

  The *param2* parameter is the handle of the currently selected spin field.

  This message tells the application which spin field is selected.

- SPBN_KILLFOCUS

  The *param2* parameter is NULLHANDLE if the spin field loses focus or no spin field is currently selected.

  This message tells the application when a spin field loses focus.

  Note: Both SPBN_KILLFOCUS and SPBN_SETFOCUS are set independently. You must check this message only when the application does not specify a master-servant relationship.

- SPBN_CHANGE

  The *param2* parameter is the handle of the spin button in which the spin field text changed.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

Related Data Structure

This section covers the data structure that is related to spin button controls.

SPBCDATA

Spin Button control data structure.

Syntax

```
typedef struct _SPBCDATA {
ULONG      cbSize;
ULONG      ulTextLimit;
LONG       lLowerLimit;
LONG       lUpperLimit;
ULONG      idMasterSpb;
PVOID      pHWXCtlData;
 } SPBCDATA;

typedef SPBCDATA *PSPBCDATA;
```

The SPBCDATA structure is used in WinCreateWindow's *pCtlData* parameter.

When using this structure the SPBM_SETLIMITS, SPBM_SETTEXTLIMIT, and SPBM_SETMASTER messages do not need to be specified.

- *ulTextLimit* and *lLowerLimit* replace SPBM_SETLIMITS.
- *lUpperLimit* replaces SPBM_SETTEXTLIMIT.
- *idMasterSpb* replaces SPBM_SETMASTER.

Fields

cbSize (ULONG)
Size of control block.

ulTextLimit (ULONG)
Entryfield text limit.

lLowerLimit (LONG)
Spin lower limit (numeric only).

lUpperLimit (LONG)
Spin upper limit (numeric only).

idMasterSpb (ULONG)
ID of the servant's master spinbutton.

pHWXCtlData (PVOID)
Reserved for Pen *CtlData*.

Summary

Following are tables that describe the OS/2 window messages, notification message, notification codes, and data structure used with spin button controls:

Table 4-1. Spin Button Control Window Messages

Message Name	Description
SPBM_OVERRIDESETLIMITS	Causes the component to set or reset numeric limits.
SPBM_QUERYLIMITS	Enables an application to query the limits of a numeric spin field.
SPBM_QUERYVALUE	Causes the component to show the value in the spin field.
SPBM_SETARRAY	Causes the component to set or reset the array of data.
SPBM_SETCURRENTVALUE	Causes the component to set or reset the current numeric value or array index.
SPBM_SETLIMITS	Causes the component to set or reset numeric limits.
SPBM_SETMASTER	Causes the component to identify its master.
SPBM_SETTEXTLIMIT	Sets the maximum number of characters allowed in a spin field.
SPBM_SPINDOWN	Causes the component to show the previous value (spin backward).
SPBM_SPINUP	Causes the component to show the next value (spin forward).

Table 4-2. Spin Button Control Notification Message

Message Name	Description
WM_CONTROL	Occurs when the spin button control has a significant event to notify to its owner.

Table 4-3. Spin Button Control Notification Codes

Code Name	Description
SPBN_CHANGE	Sent when the contents of the spin field change.
SPBN_DOWNARROW	Sent when the Down Arrow button is clicked on or the Down Arrow key is pressed.
SPBN_ENDSPIN	Sent when the user releases the select button or one of the arrow keys while spinning a button.
SPBN_KILLFOCUS	Sent when the spin field loses the focus.
SPBN_SETFOCUS	Sent when the spin field is selected.
SPBN_UPARROW	Sent when the Up Arrow button is clicked on or the Up Arrow key is pressed.

Table 4-4. Spin Button Control Data Structure	
Data Structure Name	**Description**
SPBCDATA	Spin button data structure.

Chapter 5. Static Controls

A *static* control is a simple text field, bit map, or icon that an application can use to label, enclose, or separate other control windows. This chapter describes how to create and use static controls in PM applications.

About Static Controls

Unlike the other types of control windows, a static control does not accept user input or send notification messages to its owner. The primary advantage of a static control is that it provides a label or graphic that requires little attention from an application. At most, an application might change the text or position of a static control.

Keyboard Focus

A static control never accepts the keyboard focus. When a static control receives a WM_SETFOCUS message, or when a user clicks the static control, the system advances the focus to the next sibling window that is not a static control. If the control has no siblings, the system gives the focus to the owner of the static control.

Static Control Handle

Every static control is associated with a 32-bit data field. A static control with the SS_BITMAP or SS_ICON style uses this field to store the handle of the bit map or icon that it displays. An application can obtain that handle by sending the SM_QUERYHANDLE message to the control. An application can replace the bit map or icon by sending the SM_SETHANDLE message to the control, specifying a valid icon or bit map handle. Changing the handle causes the system to redraw the control.

For a non-icon or non-bit map static control, the data field is available for application-defined data and has no effect on the appearance of the control.

An application can retrieve the data field of a static control window by calling WinWindowFromID, using the handle of the owner and the window identifier of the static control. The static control window identifier is specified in either the dialog-window template or WinCreateWindow.

Static Control Styles

A static control has style bits that determine whether the control displays text, draws a simple box containing text, displays an icon or a bit map, or shows a framed or unframed colored box. Applications can specify a combination of the styles listed in Table 5-1 on page 5-2 for a static control.

Table 5-1. Static Control Styles	
Style Name	**Description**
SS_BITMAP	Draws a bit map. The bit map resource must be provided in the resource-definition file. To include the bit map in a dialog window, the resource identifier must be specified in the *text* parameter of the CONTROL statement in the resource definition file. To include the bit map in a non-dialog window, the ASCII representation of the identifier must be specified in the *pszName* parameter of WinCreateWindow, that is, the first byte of the *pszName* parameter must be the cross-hatch character (#), and the remaining text must be the ASCII representation of the identifier, for example, #125.
SS_BKGNDFRAME	Creates a box whose frame has the background color.
SS_BKGNDRECT	Creates a rectangle filled with the background color.
SS_FGNDFRAME	Creates a box whose frame has the foreground color.
SS_FGNDRECT	Creates a rectangle filled with the foreground color.
SS_GROUPBOX	Creates a box whose upper-right corner contains control text. This style is useful for enclosing groups of radio buttons or check boxes in a box.
SS_HALFTONEFRAME	Creates a box whose frame has halftone shading.
SS_HALFTONERECT	Creates a box filled with halftone shading.
SS_ICON	Draws an icon. The resource identifier for the icon resource is determined the same way as the SS_BITMAP style. The icon resource must be in the resource-definition file.
SS_SYSICON	Draws a system-pointer icon. The resource identifier for the system-pointer resource is determined the same way as the SS_BITMAP style. To display this system pointer, the system calls WinQuerySysPointer with the specified identifier.
SS_TEXT	Creates a box with formatted text. An application can combine various formatting options with this style to produce formatted text in the boundaries of the control. The formatting flags are the same as those used for WinDrawText.

Default Static Control Performance

The messages specifically handled by the predefined static control class (WC_STATIC) are listed in Table 5-2 on page 5-3.

Table 5-2. Static Control Messages Handled by the WC_STATIC Class

Message Name	Description
SM_QUERYHANDLE	Returns the handle associated with the static control window.
SM_SETHANDLE	Sets the handle associated with the static control and invalidates the control window, forcing it to be redrawn.
WM_ADJUSTWINDOWPOS	Adjusts the SWP data structure so that the new window size matches the bit map, icon, or system-pointer dimensions associated with the static control.
WM_CREATE	Sets the text for a static-text control. Loads the bit map or icon resource for the bit map or icon static control. Returns TRUE if the resource cannot be loaded.
WM_DESTROY	Frees the text for a static-text control. Destroys the bit map or icon for a bit map or icon static control. The icon for a system-pointer static control is not destroyed because it belongs to the system.
WM_ENABLE	Invalidates the entire static control window, forcing it to be redrawn.
WM_HITTEST	Returns the value HT_TRANSPARENT for the following static-control styles: • SS_BKGNDFRAME • SS_BKGNDRECT • SS_FGNDFRAME • SS_FGNDRECT • SS_GROUPBOX • SS_HALFTONEFRAME • SS_HALFTONERECT. For other styles, this message returns the result of WinDefWindowProc.
WM_MATCHMNEMONIC	Returns TRUE if the mnemonic passed in the *mp1* parameter matches the mnemonic in the control-window text.
WM_MOUSEMOVE	Sets the mouse pointer to the arrow pointer and returns TRUE.
WM_PAINT	Draws the static control based on its style attributes.
WM_QUERYDLGCODE	Returns the predefined constant DLGC_STATIC.
WM_QUERYWINDOWPARAMS	Returns the requested window parameters.
WM_SETFOCUS	Sets the focus to the next sibling window that can accept the focus; or if no such sibling exists, sets the focus to the parent window.
WM_SETWINDOWPARAMS	Allows the text to be set (static-text controls only).

Using Static Controls

This section explains how to perform the following tasks:

- Include a static control in a dialog window
- Include a static control in a client window.

Including a Static Control in a Dialog Window

To include a static control in a dialog window, you must define the control in a dialog-window template in a resource-definition file. The resource-definition file illustrated in Figure 5-1 creates a dialog window that contains a static-text control and three static-icon controls.

```
DLGTEMPLATE IDD_TOOLDLG LOADONCALL MOVEABLE DISCARDABLE
  BEGIN
    DIALOG "",
           IDD_TOOLDLG,
           114, 53, 161, 127,
           FS_NOBYTEALIGN |
           FS_DLGBORDER   |
           WS_VISIBLE     |
           WS_SAVEBITS

    BEGIN
      CTEXT "Select a tool",
            IDS_TEXT,
            49, 110, 56, 8,
            SS_TEXT   |
            DT_CENTER |
            DT_TOP    |
            WS_GROUP  |
            WS_VISIBLE

      AUTORADIOBUTTON "Paintbrush",
                      IDB_BRUSH,
                      63, 87, 61, 10,
                      WS_TABSTOP |
                      WS_GROUP   |
                      WS_VISIBLE
```

Figure 5-1 (Part 1 of 3). Sample Code Using a Static Control in a Dialog Window

```
AUTORADIOBUTTON "Scissors",
                IDB_SCISSORS,
                63, 64, 60, 10,
                WS_TABSTOP |
                WS_VISIBLE

AUTORADIOBUTTON "Eraser",
                IDB_ERASER,
                65, 39, 43, 10,
                WS_TABSTOP |
                WS_VISIBLE

ICON IDI_BRUSH,
     IDI_BRUSHICON,
     33, 84, 22, 16,
     WS_GROUP |
     WS_VISIBLE

ICON IDI_SCISSORS,
     IDI_SCISSORSICON,
     33, 60, 22, 16,
     WS_GROUP |
     WS_VISIBLE

ICON IDI_ERASER,
     IDI_ERASERICON,
     33, 36, 22, 16,
     WS_GROUP |
     WS_VISIBLE

PUSHBUTTON "OK",
           DID_OK,
           10, 12, 38, 13,
           WS_TABSTOP |
           WS_GROUP   |
           WS_VISIBLE
```

Figure 5-1 (Part 2 of 3). Sample Code Using a Static Control in a Dialog Window

```
     PUSHBUTTON "Cancel",
                DID_CANCEL,
                59, 12, 38, 13,
                BS_DEFAULT |
                WS_TABSTOP |
                WS_GROUP   |
                WS_VISIBLE

     PUSHBUTTON "Help",
                IDB_HELP,
                111, 13, 38, 13,
                BS_HELP    |
                WS_TABSTOP |
                WS_GROUP   |
                WS_VISIBLE
   END
 END

ICON IDI_BRUSH    brush.ico
ICON IDI_SCISSORS scissr.ico
ICON IDI_ERASER   eraser.ico
```

Figure 5-1 (Part 3 of 3). Sample Code Using a Static Control in a Dialog Window

Including a Static Control in a Client Window

An application can include a static control in a non-dialog window by calling WinCreateWindow with the window class WC_STATIC. The *flStyle* parameter to WinCreateWindow defines the appearance of the control.

The sample code illustrated in Figure 5-2 on page 5-7 creates a static-text control whose size and position are based on the size of the client window and the metrics for the current font.

```
#define ID_TITLE 5

HWND hwnd,hwndStatic,hwndClient;
HPS hps;
RECTL rcl;
FONTMETRICS fm;
ULONG ulTitleLen;
CHAR szTitle[] = "Static Text Controls";

/* Obtain the size of the client window */
WinQueryWindowRect(hwnd, &rcl);

/* Obtain a presentation space handle and */
/* the metrics for the current font       */
hps = WinBeginPaint (hwnd, (HPS) NULL, (PRECTL) NULL);
GpiQueryFontMetrics(hps, sizeof(FONTMETRICS), &fm);

/* Obtain the size of the static control text string */
ulTitleLen = (ULONG) strlen(szTitle);

/* Create the static control. Base the size and  */
/* position on the size of the client window and */
/* the metrics of the current font.              */

hwndStatic = WinCreateWindow(
               hwndClient,                    /* Parent window        */
               WC_STATIC,                     /* Window class         */
               szTitle,                       /* Window text          */
               WS_VISIBLE |                   /* Make it visible      */
               SS_TEXT    |                   /* Static-text control  */
               DT_VCENTER |                   /* Center text vert.    */
               DT_CENTER,                     /* Center text horiz.   */
```

Figure 5-2 (Part 1 of 2). Sample Code Using a Static Control in a Client Window

```
            ((rcl.xRight / 2) -
            (ulTitleLen / 2) * fm.lEmInc), /* x position             */
            rcl.yTop - fm.lEmHeight * 2,   /* y position             */
            fm.lEmInc * ulTitleLen,        /* Width                  */
            fm.lEmHeight * 2,              /* Height                 */
            hwndClient,                    /* Owner window           */
            HWND_TOP,                      /* Top of z-order         */
            ID_TITLE,                      /* Window identifier      */
            NULL,                          /* Control data           */
            NULL);                         /* Presentation parameters*/

WinEndPaint(hps);
```

Figure 5-2 (Part 2 of 2). Sample Code Using a Static Control in a Client Window

If your application creates a static control with the SS_ICON or SS_BITMAP style, make sure that the resource identifier specified in the *pszName* parameter corresponds to an icon or a bit map resource in the resource-definition file. If there is no resource, the application cannot create the static control.

Related Functions

This section covers the functions that are related to static controls.

WinQuerySysPointer

This function returns the system-pointer handle.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

HPOINTER WinQuerySysPointer (HWND hwndDeskTop, LONG lIdentifier, BOOL fCopy)

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

lIdentifier (LONG) – input
System-pointer identifier.

SPTR_ARROW	Arrow pointer
SPTR_TEXT	Text I-beam pointer
SPTR_WAIT	Hourglass pointer
SPTR_SIZE	Size pointer
SPTR_MOVE	Move pointer
SPTR_SIZENWSE	Downward-sloping, double-headed arrow pointer
SPTR_SIZENESW	Upward-sloping, double-headed arrow pointer
SPTR_SIZEWE	Horizontal, double-headed arrow pointer
SPTR_SIZENS	Vertical, double-headed arrow pointer
SPTR_APPICON	Standard application icon pointer
SPTR_ICONINFORMATION	Information icon pointer
SPTR_ICONQUESICON	Question mark icon pointer
SPTR_ICONERROR	Exclamation mark icon pointer
SPTR_ICONWARNING	Warning icon pointer
SPTR_ILLEGAL	Illegal operation icon pointer
SPTR_FILE	Single file icon pointer
SPTR_MULTFILE	Multiple files icon pointer
SPTR_FOLDER	Folder icon pointer
SPTR_PROGRAM	Application program icon pointer

fCopy (BOOL) – input
Copy indicator.

TRUE	Create a copy of the default system pointer and return its handle. Specify this value if the system pointer is to be modified. The application should destroy the copy of the pointer created. This can be done by using the WinDestroyPointer function.
FALSE	Return the handle of the current system pointer.

Returns

hptrPointer (HPOINTER) – returns
Pointer handle.

WinSetWindowPos

This function allows the general positioning of a window.

Note: Messages may be received from other processes or threads during the processing of this function.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetWindowPos (HWND hwnd, HWND hwndInsertBehind, LONG x,
                      LONG y, LONG cx, LONG cy, ULONG fl)
```

Parameters

hwnd (HWND) – input
Window handle.

hwndInsertBehind (HWND) – input
Relative window-placement order.

HWND_TOP	Place *hwnd* on top of all siblings
HWND_BOTTOM	Place *hwnd* behind all siblings
Other	Identifies the sibling window behind which *hwnd* is to be placed.

x (LONG) – input
Window position, x-coordinate.

y (LONG) – input
Window position, y-coordinate.

cx (LONG) – input
Window size.

cy (LONG) – input
Window size.

fl (ULONG) – input
Window-positioning options.

SWP_SIZE	Change the window size.
SWP_MOVE	Change the window x,y position.
SWP_ZORDER	Change the relative window placement.
SWP_SHOW	Show the window.
SWP_HIDE	Hide the window.
SWP_NOREDRAW	Changes are not redrawn.
SWP_NOADJUST	Do not send a WM_ADJUSTWINDOWPOS message before moving or sizing.
SWP_ACTIVATE	Activate the *hwnd* window if it is a frame window. This indicator has no effect on other windows. The frame window is made the topmost window, unless SWP_ZORDER is specified also in which instance the *hwndInsertBehind* window is used.
SWP_DEACTIVATE	Deactivate the *hwnd* window if it is a frame window. This indicator has no effect on other windows. The frame window is made the bottommost window, unless SWP_ZORDER is specified, in which instance the *hwndInsertBehind* window is used.
SWP_MINIMIZE	Minimize the window. This indicator has no effect if the window is in a minimized state, and is also mutually exclusive with SWP_MAXIMIZE and SWP_RESTORE.
SWP_MAXIMIZE	Maximize the window. This indicator has no effect if the window is in a maximized state, and is also mutually exclusive with SWP_MINIMIZE and SWP_RESTORE.

SWP_RESTORE	Restore the window. This indicator has no effect if the window is in its normal state, and is also mutually exclusive with SWP_MINIMIZE and SWP_MAXIMIZE. The position and size of the window in its normal state is remembered in its window words when it is first maximized or minimized, although these values can be altered by use of the WinSetWindowUShort function. The window is restored to the position and size remembered in its window words, unless the SWP_MOVE or SWP_SIZE indicators are set. These indicators cause the position and size values specified in this function to be used.

Returns

rc (BOOL) – returns
Repositioning indicator.

TRUE	Window successfully repositioned
FALSE	Window not successfully repositioned.

WinSetWindowText

This function sets the window text for a specified window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetWindowText (HWND hwnd, PSZ pszString)
```

Parameters

hwnd (HWND) – input
Window handle.

pszString (PSZ) – input
Window text.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Text updated
FALSE	Error occurred.

WinWindowFromID

This function returns the handle of the child window with the specified identity.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinWindowFromID (HWND hwndParent, ULONG id)
```

Parameters

hwndParent (HWND) – input
Parent-window handle.

id (ULONG) – input
Identity of the child window.

Returns

hwnd (HWND) – returns
Window handle.

NULLHANDLE	No child window of the specified identity exists
Other	Child-window handle.

Related Window Messages

This section covers the window messages that are related to static controls.

SM_QUERYHANDLE

This message returns the icon or bit-map handle of a static control.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

hbmHandle (HBITMAP)
Icon or bit-map handle of the static control.

NULLHANDLE	No icon or bit-map handle of the static control exists, or an error occurred.
Other	Icon or bit-map handle of the static control.

SM_SETHANDLE

This message sets the icon or bit-map handle of a static control.

Parameters

param1

hbmHandle (HBITMAP)
Icon or bit-map handle of a static control.

This is an icon handle when sent to a control with a style of SS_ICON or SS_SYSICON, and a bit-map handle when sent to a control with a style of SS_BITMAP.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

hbmHandle (HBITMAP)

Icon or bit-map handle of the static control.

NULLHANDLE	No icon or bit-map handle of the static control exists, or an error occurred.
Other	Icon or bit-map handle of the static control.

WM_CONTROL

This message occurs when a control has a significant event to notify to its owner.

Parameters

param1

id (USHORT)

Control-window identity.

This is either the *id* parameter of the WinCreateWindow function or the identity of an item in a dialog template.

usnotifycode (USHORT)

Notify code.

The meaning of the notify code depends on the type of the control. For details, refer to the section describing that control.

param2

ulcontrolspec (ULONG)

Control-specific information.

The meaning of the control-specific information depends on the type of the control. For details, refer to the section describing that control.

Returns

ulReserved (ULONG)

Reserved value, should be 0.

WM_MATCHMNEMONIC

This message is sent by the dialog box to a control window to determine whether a typed character matches a mnemonic in its window text.

Parameters

param1

usmatch (USHORT)
Match character.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Match indicator.

TRUE	Mnemonic found
FALSE	Mnemonic not found, or an error occurred.

WM_PRESPARAMCHANGED

This message is sent when a presentation parameter is set or removed dynamically from a window instance using the WinSetPresParam or WinRemovePresParam functions. It is also sent to all windows owned by the window whose presentation parameter was changed.

Parameters

param1

idAttrType (ULONG)
Presentation parameter attribute identity.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_QUERYCONVERTPOS

This message is sent by an application to determine whether it is appropriate to begin conversion of DBCS characters.

Parameters

param1

pCursorPos (PRECTL)
Cursor position.

If *usCode* = QCP_CONVERT, *pCursorPos* should be updated to contain the position of the cursor in the window receiving this message. The position is specified as a rectangle in screen coordinates.

If *usCode* = QCP_NOCONVERT, *pCursorPos* should not be updated.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

usCode (USHORT)
Conversion code.

QCP_CONVERT	Conversion may be performed for the window with the input focus, *pCursorPos* has been updated to contain the position of the cursor.
QCP_NOCONVERT	Conversion should not be performed, the window with the input focus cannot receive DBCS characters, *pCursorPos* has not been updated.

Summary

Following are tables that describe the functions and window messages used with static controls:

Table 5-3. Static Control Functions

Function Name	Description
WinQuerySysPointer	Returns the system pointer handle.
WinSetWindowPos	Allows the general positioning of a window.
WinSetWindowText	Sets the window text for a specified window.
WinWindowFromID	Returns the handle of the child window with the specified identity.

Table 5-4. Static Control Window Messages

Message Name	Description
SM_QUERYHANDLE	Returns the icon or bit map handle of a static control.
SM_SETHANDLE	Sets the icon or bit map handle of a static control.
WM_CONTROL	Occurs when a control has a significant event to notify to its owner.
WM_MATCHMNEMONIC	Sent by the dialog box to a control window to determine whether a typed character matches a mnemonic in its window text.
WM_PRESPARAMCHANGED	Sent when a presentation parameter is set or removed dynamically from a window instance using the WinSetPresParam or WinRemovePresParam functions. It is also sent to all windows owned by the window whose presentation parameter was changed.
WM_QUERYCONVERTPOS	Sent by an application to determine whether it is appropriate to begin conversion of DBCS characters.
WM_QUERYWINDOWPARAMS	Occurs when an application queries the static control window procedure window parameters.
WM_SETWINDOWPARAMS	Occurs when an application sets or changes the static control window procedure window parameters.

Chapter 6. Slider Controls

A slider control is a visual component that displays a range of values and allows a user to set, display, or modify a value by moving a slider arm. There are two types of sliders:

- The *linear slider* is represented as a shaft along which the slider arm can be moved by the user to set a value.
- The *circular slider* is represented as a dial with the slider arm shown as the radius of the dial.

This chapter explains how to use each of these slider controls in PM applications.

About Slider Controls

This section covers linear and circular slider controls. Linear sliders are used to set values that have familiar increments. Circular sliders, although different in appearance from linear sliders, provide much the same function. Both types of sliders can be used in a window to create a user interface.

Linear Sliders

Typically, linear sliders are used to easily set values that have familiar increments, such as feet, inches, degrees, and decibels. They also can be used for other purposes when immediate feedback is required, such as blending colors or showing a task's percentage of completion. For example, an application might let a user mix and match color shades by moving a slider arm, or a read-only slider could show how much of a task is complete by filling in the slider shaft as the task progresses. These are just a few examples of the ways in which sliders can be used. Figure 6-1 shows an example of a slider used to set a decibel value.

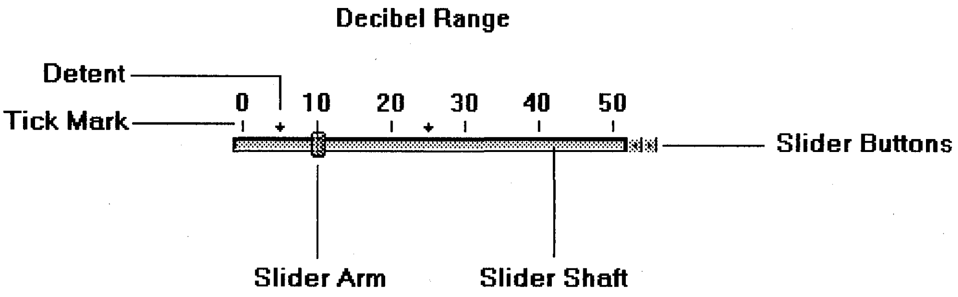

Figure 6-1. Linear Slider Example

The *slider arm* shows the value currently set by its position on the *slider shaft*. The user selects slider values by changing the location of the slider arm.

A *tick mark* indicates an incremental value in a slider scale. A *detent*, which is similar to a tick mark, also represents a value on the scale. However, a detent can be placed anywhere along the slider scale, rather than only in specific increments, and can be selected.

The appearance of a slider and the user interaction with a slider are similar to that of a scroll bar. However, these two controls are not interchangeable because each has a unique purpose. A scroll bar scrolls information into view that is outside a window's work area, while the slider is used to set, display, or modify that information, whether it is in or out of the work area.

Although linear sliders usually use values that have familiar increments, text also can be used. However, if the text is too long it can overlap the text displayed on the next tick mark or detent. Also, if the text on the far edge markers is too long, some of the text will not be displayed on screen. To prevent this use one of the following:

- Smaller font
- Shorter text values
- Static controls.

The slider can be customized to meet varying application requirements, while providing a user interface component that can be used easily to develop products that conform to the Systems Application Architecture (SAA) Common User Access (CUA) user interface guidelines. The application can specify different scales, sizes, and orientations for its sliders, but the underlying function of the control remains the same. For a complete description of CUA sliders, refer to the *SAA CUA Guide to User Interface Design* and the *SAA CUA Advanced Interface Design Reference*.

Linear Slider Styles

Slider control styles are set when a slider window is created. Table 6-1 describes linear slider control styles. If no styles are specified, defaults are used as indicated in the table.

Table 6-1 (Page 1 of 3). Linear Slider Control Styles

Style Name	Description
SLS_BOTTOM	Positions the slider at the bottom of the slider window. Valid only for horizontal sliders.
SLS_BUTTONSBOTTOM	Specifies that the optional slider buttons are to be used and places them at the bottom of the slider shaft. The buttons move the slider arm by one position, up or down, in the direction selected. Valid only for vertical sliders.
SLS_BUTTONSLEFT	Specifies that the optional slider buttons are to be used and places them to the left of the slider shaft. The buttons move the slider arm by one position, left or right, in the direction selected. Valid only for horizontal sliders.
SLS_BUTTONSRIGHT	Specifies that the optional slider buttons are to be used and places them to the right of the slider shaft. The buttons move the slider arm by one position, left or right, in the direction selected. Valid only for horizontal sliders.

Table 6-1 (Page 2 of 3). Linear Slider Control Styles

Style Name	Description
SLS_BUTTONSTOP	Specifies that the optional slider buttons are to be used and places them at the top of the slider shaft. The buttons move the slider arm by one position, up or down, in the direction selected. Valid only for vertical sliders.
SLS_CENTER	Centers the slider within the slider window. This is the default position of the slider.
SLS_HOMEBOTTOM	Specifies the slider arm's home position. The bottom of the slider is used as the base value for incrementing. Valid only for vertical sliders.
SLS_HOMELEFT	Specifies the slider arm's home position. The left edge is used as the base value for incrementing. This is the default for horizontal sliders and is valid only for horizontal sliders.
SLS_HOMERIGHT	Specifies the slider arm's home position. The right edge is used as the base value for incrementing. Valid only for horizontal sliders.
SLS_HOMETOP	Specifies the slider arm's home position. The top of the slider is used as the base value for incrementing. Valid only for vertical sliders.
SLS_HORIZONTAL	Positions the slider horizontally. The slider arm can move left and right on the slider shaft. A scale can be placed on top of the slider shaft, below the slider shaft, or in both places. This is the default orientation of the slider.
SLS_LEFT	Positions the slider at the left edge of the slider window. Valid only for vertical sliders.
SLS_OWNERDRAW	Notifies the application whenever the slider shaft, the ribbon strip, the slider arm, and the slider background are to be drawn.
SLS_PRIMARYSCALE1	Determines the location of the scale on the slider shaft by using increment and spacing specified for scale 1 as the incremental value for positioning the slider arm. Scale 1 is displayed above the slider shaft of a horizontal slider and to the right of the slider shaft of a vertical slider. This is the default for a slider.
SLS_PRIMARYSCALE2	Determines the location of the scale on the slider shaft by using increment and spacing specified for scale 2 as the incremental value for positioning the slider arm. Scale 2 is displayed below the slider shaft of a horizontal slider and to the left of the slider shaft of a vertical slider.
SLS_READONLY	Creates a read-only slider, which presents information to the user but allows no interaction with the user.
SLS_RIBBONSTRIP	Fills, as the slider arm moves, the slider shaft between the home position and the slider arm with a color value different from slider shaft color, similar to mercury in a thermometer.

Table 6-1 (Page 3 of 3): Linear Slider Control Styles	
Style Name	**Description**
SLS_RIGHT	Positions the slider at the right edge of the slider window. Valid only for vertical sliders.
SLS_SNAPTOINCREMENT	Causes the slider arm, when positioned between two values, to be positioned to the nearest value and redrawn at that position.
SLS_TOP	Positions the slider at the top of the slider window. Valid only for horizontal sliders.
SLS_VERTICAL	Positions the slider vertically. The slider arm can move up and down the slider shaft. A scale can be placed on the left side of the slider shaft, on the right side of the slider shaft, or in both places.

More on Linear Slider Styles

This section summarizes information in the table and provides additional information on some of the styles.

Slider Orientation

The slider's orientation is determined by specifying SLS_HORIZONTAL or SLS_VERTICAL. The default orientation is horizontal, with the slider arm moving left and right on the shaft.

Slider Positioning

The slider's positioning within the slider window is determined by specifying SLS_CENTER, SLS_BOTTOM, SLS_TOP, SLS_LEFT, or SLS_RIGHT. The default positioning is centered in the slider window.

Slider Scale Location

The location of the scale on the slider shaft is determined by specifying SLS_PRIMARYSCALE1 or SLS_PRIMARYSCALE2. The default is to use the increment and spacing specified for scale 1 as the incremental value for positioning the slider arm. Scale 1 is displayed above the slider shaft of a horizontal slider and to the right of the slider shaft of a vertical slider.

Slider Arm Home Position

The slider arm's home position is determined by specifying SLS_HOMELEFT, SLS_HOMERIGHT, SLS_HOMEBOTTOM, or SLS_HOMETOP. The default is SLS_HOMELEFT for horizontal sliders and SLS_HOMEBOTTOM for vertical sliders.

Slider Buttons Location

The location of the slider buttons, if used, is determined by specifying SLS_BUTTONSLEFT, SLS_BUTTONSRIGHT, SLS_BUTTONSBOTTOM, or SLS_BUTTONSTOP. If you do not specify one of these styles, or if conflicting styles are specified, slider buttons are not included in the slider control.

Other Styles

If SLS_SNAPTOINCREMENT is specified and the slider arm is moved to a position between two values on the slider scale, it is positioned on the nearest value and redrawn at that position. If this style is not specified, the slider arm remains at the position to which it is moved.

SLS_READONLY creates a read-only slider. This means the user cannot interact with the slider. It is simply used as to present a quantity to the user, such as the percentage of completion of an ongoing task. Visual differences for a read-only slider include a narrow slider arm, no slider buttons, and no detents.

The SLS_RIBBONSTRIP style allows movement of the slider arm to cause the slider shaft between home position and the slider arm to be filled with a color value that is different from the slider shaft color, similar to mercury in a thermometer.

If SLS_OWNERDRAW is specified, the application is notified whenever the slider shaft, the ribbon strip, the slider arm, and the slider background are to be drawn.

Circular Sliders

The circular slider, although different in appearance from the linear slider, provides much the same function. The circular slider control provides an analog user interface and emulates the controls of stereo and video equipment. Because the circular slider takes up less space on the screen, it may be more practical to use in cases where you might want to have several controls in the same window. You may want to use both types of sliders in a window to create a user interface that makes good use of available space and provides a familiar appearance to the user. Figure 6-2 shows an example of a circular slider used to set the volume.

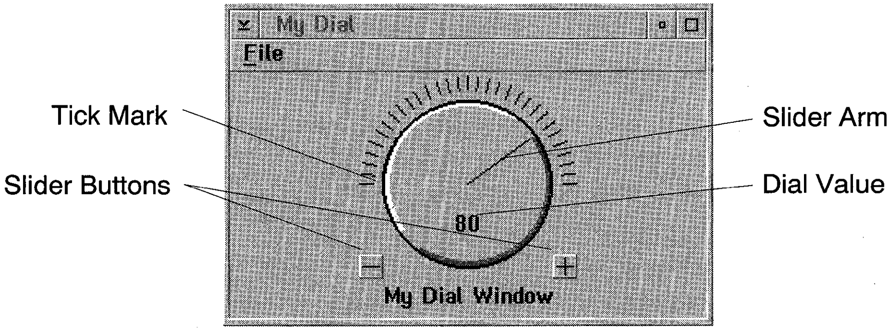

Figure 6-2. Circular Slider Example

The slider arm shows the value currently set by its position on the slider dial. Figure 6-2 shows the slider arm as the radius on the dial. The slider arm can also be represented as a small circular thumb on the dial rather than a line. The user selects slider values by changing the location of the slider arm on the dial. Outside the perimeter of the dial is a circular scale with tick marks representing incremental values the slider arm can point to. Its values can be tracked by pointing to any area on the dial and pressing the select button while moving the mouse on the desktop.

The circular slider can have a set of buttons, one to the left and the other to the right of the scroll range, similar to the buttons found on the linear slider, that can be used to modify the value shown on the slider.

The minus sign on the left button and plus sign on the right button can be replaced with other symbols. For example, you might want to use a left arrow and a right arrow instead of the minus and plus signs.

Another option of the circular slider is to have a window, in the center of the dial, that displays the value of the dial in scrollable numeric text.

The appearance and functionality of the circular slider are controlled by the circular slider control styles specified. These style bits are summarized in the section that follows.

Circular Slider Styles

Circular slider control style bits control the appearance and behavior of the slider. Table 6-2 describes circular slider control styles.

Table 6-2. Circular Slider Control Styles

Style Name	Description
CSS_360	Extends the scroll range of the dial to 360 degrees. When this style is set, CSS_NONUMBER and CSS_NOBUTTON styles automatically are set.
CSS_CIRCULARVALUE	Draws a circular thumb, rather than a line, for the value indicator.
CSS_MIDPOINT	Enlarges the mid-point and end-point tick marks.
CSS_NOBUTTON	Prevents the display of the + and – value buttons.
CSS_NONUMBER	Prevents the display of a scrollable numeric value on the dial indicating the dial's value.
CSS_NOTEXT	Prevents the display of a window title beneath the dial.
CSS_POINTSELECT	Enables tracking of the dial's value with the mouse.
CSS_PROPORTIONALTICKS	Enables the length of the tick marks to be calculated as a percentage of the dial's radius.

More on Circular Slider Styles

This section provides information on some of the styles.

Circular Slider Buttons

The circular slider has a set of buttons, one to the left and the other to the right of the scroll range. These buttons are similar to the buttons found on the linear slider. When selected, they modify the value of the circular slider in opposing ways. If you decide you do not want to display these buttons as part of your circular slider, specify the style CSS_NOBUTTON.

The bit maps that show a minus sign (–) on the left button and a plus sign (+) on the right button can be replaced. For example, you might want to use a left arrow (←) and a right arrow (→). To set the bit map data for the replacement bit maps, the owner window must send the CSM_SETBITMAPDATA control message. The optimal size for the button bit maps is 10x10 pels.

Window Title

Centered beneath the dial of the circular slider is a rectangular text field that contains the title of the window. To prevent the display of this feature, specify CSS_NOTEXT.

Dial Value Window

Another option of the circular slider is a window, in the center of the dial, that displays the value of the dial in scrollable numeric text. To prevent the display of this window, specify CSS_NONUMBER.

360-Degree Scale

You can choose a scale of 360 degrees for your slider with CSS_360. Setting this style causes the CSS_NOBUTTON and CSS_NONUMBER styles to be set automatically. The CSS_NONUMBER style prevents the value indicator from corrupting the dial value. A 360-degree circular slider is displayed without the bit maps for the plus and minus buttons.

Tracking Modes for Direct Manipulation

There are two tracking modes used for direct manipulation of the circular slider: scrolling the dial and point selection.

The default tracking behavior is scrolling, where the dial position of the slider is changed gradually. For example, by holding down the select mouse button while the pointer is positioned on the indicator line of the dial, you can move the mouse and cause the dial to rotate. This gradual changing of the dial value might be desirable for a volume control.

If the CSS_POINTSELECT style is specified for the circular slider, the position of the dial is changed immediately to a point on the scale that has been selected by the mouse. Point selection allows value changes to occur immediately.

Sizing Tick Marks

When a low-resolution display is being used, or in situations where the circular slider must be made very small, the length of the tick marks can be sized proportionately to allow effective sizing of the circular slider. Specifying the CSS_PROPORTIONALTICKS style causes the length of the tick marks to be calculated as a percentage of the dial's radius. This style does not adversely affect the size of the push buttons and bit-map graphics an application might provide.

Using Slider Controls

This section explains how to use sliders in your PM applications. It covers:

- Creating a linear slider
- Retrieving data for selected slider values
- Creating a circular slider.

Code samples are provided.

Creating a Linear Slider

Before the slider is created, a temporary SLDCDATA data structure is allocated, and variables are specified for the slider control window handle and slider style. The SLDCDATA data structure is allocated so that the scale increments and spacing of the slider can be specified.

The slider style variable enables the application to specify style bits, SLS_* values, that are used to customize the slider.

You create a slider by using the WC_SLIDER window class name in the *ClassName* parameter of WinCreateWindow call. The handle of the slider control window is returned in the slider window variable.

After the slider is created, but before it is made visible, the application can set other slider control characteristics, such as:

- Size and placement of tick marks
- Text above one or more tick marks
- One or more detents
- Initial slider arm position.

The settings in the preceding list are just a few that an application can specify. Slider control messages are used to specify these settings.

Figure 6-1 on page 6-1 shows how the linear slider created by the sample code in Figure 6-3 on page 6-9 would appear, except for the Decibel Range text string. The code that inserts this static text string is separate from the code used to create a slider window and, therefore, is not included here. The main components of the slider are labeled.

```
SLDCDATA sldcData;                 /* SLDCDATA data structure       */
CHAR     szTickText{5};            /* Text strings variable         */
USHORT   idx;                      /* Counter for setting text      */
                                   /* strings                       */
HWND     hwndSlider;               /* Slider window handle          */
ULONG    ulSliderStyle;            /* Slider styles                 */

/*********************************************************************/
/* Initialize the parameters in the data structure.                 */
/*********************************************************************/
sldcData.cbSize = sizeof(SLDCDATA); /* Size of SLDCDATA structure    */
sldcData.usScale1Increments = 6;   /* Number of increments          */
sldcData.usScale1Spacing = 0;      /* Use 0 to have slider calculate */
                                   /* spacing                       */

/*********************************************************************/
/* Set the SLS_* style flags to the default values, plus slider     */
/* buttons right.                                                   */
/*********************************************************************/
ulSliderStyle = SLS_HORIZONTAL    | /* Slider is horizontal          */
                SLS_CENTER        | /* Slider shaft centered in      */
                                    /* slider window                 */
                SLS_HOMELEFT      | /* Home position is left edge of */
                                    /* slider                        */
                SLS_PRIMARYSCALE1 | /* Scale is displayed above      */
                                    /* slider shaft                  */
                SLS_BUTTONSRIGHT;   /* Slider buttons at right end of */
                                    /* slider                        */

/*********************************************************************/
/* Create the slider control window.                                */
/* The handle of the window is returned in hwndSlider.              */
/*********************************************************************/
hwndSlider = WinCreateWindow(
               hwndClient,         /* Parent window handle          */
               WC_SLIDER,          /* Slider window class name      */
               (PSZ)NULL,          /* No window text                */
               ulSliderStyle,      /* Slider styles variable        */
```

Figure 6-3 (Part 1 of 3). Sample Code for Creating a Slider

```
            (SHORT)10,           /* X coordinate                   */
            (SHORT)10,           /* Y coordinate                   */
            (SHORT)150,          /* Window width                   */
            (SHORT)80,           /* Window height                  */
            hwndClient,          /* Owner window handle            */
            HWND_TOP,            /* Sibling window handle          */
            ID_SLIDER,           /* Slider control window ID       */
            &sldcData,           /* Control data structure         */
            (PVOID)NULL);        /* No presentation parameters     */

/******************************************************************/
/* Set tick marks at several places on the slider shaft using the */
/* primary scale.                                                 */
/******************************************************************/
WinSendMsg(hwndSlider,               /* Slider window handle           */
           SLM_SETTICKSIZE,          /* Message for setting tick mark  */
                                     /* size.                          */
           MPFROM2SHORT(
             SMA_SETALLTICKS,        /* Attribute for setting all tick */
                                     /* marks to the same size         */
             6),                     /* Draw tick marks 6 pixels long  */
           NULL);                    /* Reserved value                 */

/******************************************************************/
/* Set text above the tick marks.                                 */
/******************************************************************/
for (idx = 0; idx <= 5; idx++)       /* Count from 0 to 5              */
{
  itoa(10*idx, szTickText, 10);      /* Set text at increments of 10   */

  WinSendMsg(hwndSlider,             /* Slider window handle           */
           SLM_SETSCALETEXT,         /* Message for setting text on a  */
                                     /* slider scale                   */
           MPFROMSHORT(idx),         /* Text string counter            */
           MPFROMPSZ(szTickText));   /* Text to put on slider scale    */
}

/******************************************************************/
/* Set detents between two of the tick marks on the slider shaft. */
/******************************************************************/
WinSendMsg(hwndSlider,               /* Slider window handle           */
           SLM_ADDDETENT,            /* Message for adding detents to  */
                                     /* a slider scale                 */
           MPFROMSHORT(5),           /* Put a detent 5 pixels from home*/
           NULL);                    /* Reserved value                 */
```

Figure 6-3 (Part 2 of 3). Sample Code for Creating a Slider

```
WinSendMsg(hwndSlider,              /* Slider window handle          */
           SLM_ADDDETENT,           /* Message for adding detents to */
                                    /* slider scale                  */
           MPFROMSHORT(25),         /* Put a detent 25 pixels from   */
                                    /* home                          */
           NULL);                   /* Reserved value                */

/*****************************************************************/
/* Set the slider arm position to the 1st increment on the scale.    */
/*****************************************************************/
WinSendMsg(hwndSlider,              /* Slider window handle          */
           SLM_SETSLIDERINFO,       /* Message for setting slider    */
                                    /* attributes                    */
           MPFROM2SHORT(
             SMA_SLIDERARMPOSITION, /* Modify slider arm position    */
             SMA_INCREMENTVALUE),   /* Use an increment value        */
           MPFROMSHORT(1));         /* Value to use is 1st           */
                                    /* increment                     */

/*****************************************************************/
/* Since all items have been set, make the control visible.          */
/*****************************************************************/
WinShowWindow(hwndSlider,           /* Slider window handle          */
              TRUE);                /* Make the window visible       */
```

Figure 6-3 (Part 3 of 3). Sample Code for Creating a Slider

Retrieving Data for Selected Slider Values

To retrieve data represented by a slider value, specify a variable for the current position of the slider arm. Then, use the SLM_QUERYSLIDERINFO message to retrieve information about the current slider arm position in increment coordinates. The code fragment in Figure 6-4 on page 6-12 shows how to retrieve data for a selected slider value.

```
ULONG  ulValue;                         /* Variable in which to store     */
                                        /* current slider arm position    */

/*****************************************************************/
/* Get the information about the current slider arm position in       */
/* incremental coordinates.                                            */
/*****************************************************************/
ulValue = (ULONG)WinSendMsg(
  hwndSlider,                           /* Slider window handle           */
  SLM_QUERYSLIDERINFO,                  /* Message for querying slider    */
                                        /* attributes                     */
  MPFROM2SHORT(
    SMA_SLIDERARMPOSITION,              /* Get increment at which slider  */
    SMA_INCREMENTVALUE),                /* arm is located                 */
  NULL);                                /* Reserved value                 */
```

Figure 6-4. Sample Code for Retrieving a Slider Value

Creating a Circular Slider

The circular slider PM window class WC_CIRCULARSLIDER is similar to the window class of a linear slider or a scroll bar. This window class must be registered with WinRegisterCircularSlider before you can create a circular slider. A circular slider can be created by a CONTROL statement in a dialog resource, as shown in Figure 6-5.

```
CONTROL "~Balance",
        ID_BALANCECS,
        10, 50, 60, 60,
        WC_CIRCULARSLIDER,
        WS_TABSTOP |
        WS_VISIBLE |
        CSS_POINTSELECT
```

Figure 6-5. Circular Slider CONTROL in a Dialog Resource

A circular slider also can be created by specifying the WC_CIRCULARSLIDER window class name as a parameter of the WinCreateWindow call, as shown in the sample code illustrated in Figure 6-6 on page 6-13.

```
hwndCS = WinCreateWindow (hwndClient,          /* Parent handle  */
                          WC_CIRCULARSLIDER,   /* Class name     */
                          "~Balance",          /* Window text    */
                          WS_VISIBLE |
                          WS_TABSTOP |
                          CSS_POINTSELECT,
                          0,0,0,0,             /* Coordinates    */
                          hwndClient,          /* Owner handle   */
                          HWND_TOP,            /* Z-order        */
                          ID_BALANCECS,        /* Window ID      */
                          NULL,                /* Control data   */
                          NULL);               /* Presparam      */
```

Figure 6-6. Sample Code Using WinCreateWindow to Create a Circular Slider

Circular Slider Sample

The sample code illustrated in Figure 6-7 shows a complete example for adding a circular slider.

```
#define  INCL_WIN

#include <os2.h>
#include "circle.h"

/* Procedure Prototype */
MRESULT EXPENTRY MyWindowProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2);
MRESULT EXPENTRY MainProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2);

/* Global Variables */
HAB      hab;
HMQ      hmq;
QMSG     qmsg;
HWND     hwndFrame;
ULONG    flCreate;
HWND     hwndClient;
```

Figure 6-7 (Part 1 of 5). Sample Code for Adding a Circular Slider

```
INT main(VOID)
{
  /* Convert system pointer into hourglass pointer */
  WinSetPointer(HWND_DESKTOP,
    WinQuerySysPointer(HWND_DESKTOP,SPTR_WAIT,FALSE));

  hab = WinInitialize(0);
  hmq = WinCreateMsgQueue(hab,0);

  WinRegisterClass(hab,"Client",MainProc,CS_SIZEREDRAW,0);

  flCreate = FCF_SYSMENU       |
             FCF_SIZEBORDER    |
             FCF_TITLEBAR      |
             FCF_MENU          |
             FCF_MINMAX        |
             FCF_SHELLPOSITION |
             FCF_TASKLIST;

  hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                 WS_VISIBLE,
                                 &flCreate,
                                 "Client",
                                 "My Dial",
                                 0L, 0,
                                 MAIN_FRAME,
                                 &hwndClient);

  /* Convert system pointer into arrow pointer */
  WinSetPointer(HWND_DESKTOP,
    WinQuerySysPointer(HWND_DESKTOP,SPTR_ARROW,FALSE));

  while (WinGetMsg(hab,&qmsg,0,0,0))WinDispatchMsg(hab,&qmsg);

  WinDestroyWindow(hwndFrame);
  WinDestroyMsgQueue(hmq);
  WinTerminate(hab);

  /* Beep when done */
  DosBeep(750,500);
  return(0);
}
```

Figure 6-7 (Part 2 of 5). Sample Code for Adding a Circular Slider

```
MRESULT EXPENTRY MainProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
    HPS     hps;
    static  HWND hwndCirc;
    SWP     swp;
    switch(msg)

    {
        case WM_CLOSE:
            WinPostMsg(hwnd,WM_QUIT,0L,0L);
            return ((MRESULT)NULL);

        case WM_COMMAND:
            /* Exit option was selected in the menu bar */
            switch(SHORT1FROMMP(mp1))
            {
                case IDM_FILEEXIT:
                    WinPostMsg(hwnd,WM_QUIT,0L,0L);
                    return ((MRESULT)NULL);
            }
            return ((MRESULT)NULL);

        case WM_CONTROL:
            /* Process circular slider notification messages */
            if (SHORT1FROMMP(mp1) == ID_DIAL)
            {
              switch (SHORT2FROMMP(mp1))
              {
               /* Notification codes can be specified here */
              }
            }
            /* Default processing for other control window ids */
            return (WinDefWindowProc(hwnd,msg,mp1,mp2));
```

Figure 6-7 (Part 3 of 5). Sample Code for Adding a Circular Slider

```
case WM_CREATE:
    /* Create circular slider control */
    hwndCirc = WinCreateWindow(hwnd,
                WC_CIRCULARSLIDER,
                "My Dial Window",
                WS_VISIBLE,
                0, 0, 0, 0,          /* Position & size      */
                hwnd,                /* Client window        */
                HWND_TOP,
                ID_DIAL,
                NULL,NULL);

    /* Specify range of values for circular slider */
    WinSendMsg (hwndCirc,
                CSM_SETRANGE,
                MPFROMLONG(0L),
                MPFROMLONG(100L));

    /* Specify scroll & tick mark increments */
    WinSendMsg (hwndCirc,
                CSM_SETINCREMENT,
                MPFROMLONG(10L),
                MPFROMLONG(2L));

    /* Set initial value */
    WinSendMsg (hwndCirc,
                CSM_SETVALUE,
                MPFROMLONG(80L),
                NULL);

    return (MRESULT)FALSE;
```

Figure 6-7 (Part 4 of 5). Sample Code for Adding a Circular Slider

```
        case WM_SIZE:
            /* The frame window has changed in size */
            /* Recalculate size of circular slider  */
            WinQueryWindowPos(hwnd,&swp);
            WinSetWindowPos(hwndCirc,
                            HWND_TOP,
                            0, 0,
                            swp.cx,
                            swp.cy,
                            SWP_MOVE |
                            SWP_SIZE);
            return (MRESULT)NULL;

        case WM_PAINT:
            hps = WinBeginPaint(hwnd,0,NULL);
            WinEndPaint(hps);
            return (MRESULT)NULL;

        default:
            return (WinDefWindowProc(hwnd,msg,mp1,mp2));
    }
}
```

Figure 6-7 (Part 5 of 5). Sample Code for Adding a Circular Slider

Graphical User Interface Support for Slider Controls

This section describes the support the slider control provides for graphical user interfaces (GUIs). Except where noted, this support conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

Since slider values all are mutually exclusive, only one of them can be selected at a time. Therefore, the only type of selection supported by the slider control is *single selection*.

Note: If more than one slider window is open, selecting values in one slider window has no effect on the values selected in any other slider window. For linear sliders, a black square is drawn in the center of the slider arm to show which slider control window has the focus.

An initial value is selected when the slider control is first displayed. If the application does not provide the initial selection for a linear slider (using the SLM_SETSLIDERINFO message) to position the slider arm, the value at the home position is selected automatically. The home position is the end of the slider that contains the lowest value on the scale.

Slider Navigation Techniques

The slider control supports the use of pointing devices and the keyboard for selecting values.

Pointing Device Support

A user can select slider values with a pointing device. The CUA guidelines defines mouse button 1 (the select button) as the button for selecting values, and button 2 (the drag button) for dragging the slider arm to a value. These definitions also apply to the same buttons on any other pointing device, such as a joystick.

The select button and drag button can be used in conjunction with the following slider components to select slider values:

- Slider arm

  Moving the pointer over the slider arm, then pressing and holding the select or drag buttons while moving the pointer, causes the slider arm to move in the direction the pointer is moving. When the button is released, the value closest to the slider arm position becomes the selected value.

- Slider shaft

  Clicking the select button when the pointer is over the slider shaft causes the slider arm to move one increment in the direction of the pointer. For linear sliders, increments are determined by the initial values passed for the primary scale specified (SLS_PRIMARYSCALE1 or SLS_PRIMARYSCALE2) when the slider is created.

  Clicking the drag button when the pointer is over the slider shaft causes the slider arm to move to the pointer's location.

- Slider buttons

  Clicking the select button when the pointer is over a slider button causes the slider arm to move one increment in the direction the arrow on the slider button is pointing.

Slider buttons are optional. If used, two slider buttons are available to the user. The arrows on top of the slider buttons point to opposite ends of the slider. Both slider buttons are positioned at the same end of the slider.

For linear sliders, slider buttons are enabled by specifying the appropriate SLS_* value when the slider control window is created. For horizontal sliders, you can specify either SLS_BUTTONSLEFT or SLS_BUTTONSRIGHT. For vertical sliders, you can specify either SLS_BUTTONSBOTTOM or SLS_BUTTONSTOP. The default is no slider buttons. If more than one of these style bits is specified, no slider buttons are enabled.

- Detents

  A detent is similar to a tick mark on a linear slider scale because it represents a value on the scale. However, unlike a tick mark, a detent can be placed anywhere along the slider scale instead of in specific increments.

  A detent can be selected by moving the pointer over it and pressing the select button on the pointing device. When this happens, the slider arm moves to the position on the slider shaft indicated by the detent.

Keyboard Support

A user can select a value by using the navigation keys to move the slider arm to the value or by typing a value in an entry field, if one is provided by the application, to change the slider arm position. The following list describes these methods of selecting slider values:

- Values can be selected using the Up, Down, Left, and Right Arrow keys to move the slider arm one increment at a time. The Up and Down Arrow keys are enabled for vertical sliders, and the Right and Left Arrow keys are enabled for horizontal sliders. If no tick mark exists on the scale in the requested direction, the slider arm does not move.

  If an Arrow key is pressed in conjunction with the Shift key, the slider arm moves to the next detent instead of the next tick mark. If no detent exists on the scale in the requested direction, the slider arm does not move.

- The Home and End keys can be used to select the lowest and highest values, respectively, in the scale. If the Ctrl key is pressed in combination with the Home or End keys, the result is the same as pressing only the Home or End keys.

- The application can provide an optional entry field for the slider control. The entry field is a separate control, but it can work in conjunction with the slider control.

  If the application provides an entry field for the slider control window, it must be implemented as follows:

  - The user must be allowed to type a value into the entry field.
  - If the typed value is within the range of the slider scale, the slider arm moves to that value as soon as the value is typed.
  - No other action, such as pressing the Enter key, is required.

Related Functions

This section covers the functions that are related to slider controls.

WinCreateWindow

This function creates a new window of class *pszClass* and returns *hwnd*.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinCreateWindow  (HWND hwndParent, PSZ pszClass, PSZ pszName,
                       ULONG flStyle, LONG x, LONG y, LONG cx, LONG cy,
                       HWND hwndOwner, HWND hwndInsertBehind,
                       ULONG id, PVOID pCtlData, PVOID pPresParams)
```

Parameters

hwndParent (HWND) – input
Parent-window handle.

pszClass (PSZ) – input
Registered-class name.

pszName (PSZ) – input
Window text.

flStyle (ULONG) – input
Window style.

x (LONG) – input
x-coordinate of window position.

y (LONG) – input
y-coordinate of window position.

cx (LONG) – input
Width of window, in window coordinates.

cy (LONG) – input
Height of window, in window coordinates.

hwndOwner (HWND) – input
Owner-window handle.

hwndInsertBehind (HWND) – input
Sibling-window handle.

id (ULONG) – input
Window identifier.

pCtlData (PVOID) – input
Pointer to control data.

pPresParams (PVOID) – input
Presentation parameters.

Returns

hwnd (HWND) – returns
Window handle.

NULLHANDLE	Error occurred
Other	Window handle.

WinSendMsg

This function sends a message with identity *ulMsgid* to *hwnd*, passing *mpParam1* and *mpParam2* as the parameters to the window.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>

MRESULT WinSendMsg  (HWND hwnd, ULONG ulMsgid, MPARAM mpParam1,
                     MPARAM mpParam2)
```

Parameters

hwnd (HWND) – input
Window handle.

ulMsgid (ULONG) – input
Message identity.

mpParam1 (MPARAM) – input
Parameter 1.

mpParam2 (MPARAM) – input
Parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

WinShowWindow

This function sets the visibility state of a window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
```

BOOL WinShowWindow (HWND hwnd, BOOL fNewVisibility)

Parameters

hwnd (HWND) – input
Window handle.

fNewVisibility (BOOL) – input
New visibility state.

TRUE	Set window state visible
FALSE	Set window state invisible.

Returns

rc (BOOL) – returns
Visibility changed indicator.

TRUE	Window visibility successfully changed
FALSE	Window visibility not successfully changed.

Related Window Messages

This section covers the window messages that are related to linear and circular slider controls.

CSM_QUERYINCREMENT

This message queries the increments used to scroll the value and draw the tick marks.

Parameters

param1

ScrollIncre (PUSHORT)
The increment value added or subtracted for the value of the control when scrolling.

param2

TickIncr (PUSHORT)
The increment value used to draw the tick marks.

Returns

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Errors occurred.

CSM_QUERYRADIUS

This message queries the current radius of the circular slider.

Parameters

param1

uRadius (PUSHORT)
The radius of the circular slider.

param2

ulReserved (ULONG)
Reserved value.

Returns

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_QUERYRANGE

This message queries the value range of the control.

Parameters

param1

pLow (PSHORT)
The low range value.

param2

pHigh (PSHORT)
The high range value.

Returns

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_QUERYVALUE

This message queries the value of the control.

Parameters

param1

pValue (PSHORT)
The value of the control.

param2

ulReserved (ULONG)
Reserved value.

Returns

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_SETBITMAPDATA

This message is used to change the bit maps for the plus and minus buttons. For example, you might want to use left or right arrows. The optimal size for these bit maps is 10 x 10 pels.

Parameters

param1

pCSBitmapData (PCSBITMAPDATA)
The structure defining button bit maps.

param2

ulReserved (ULONG)
Reserved value.

Returns

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_SETINCREMENT

This message sets the scroll and tick mark increments of the control.

Parameters

param1

usScrollIncr (USHORT)
Scroll increment.

This is the number by which the current value is incremented or decremented when one of the circular slider control button is selected.

param2

usTickIncr (USHORT)
Tick mark increment.

This represents the number of tick marks to "skip" before drawing tick marks around the circular slider.

Returns

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_SETRANGE

This message sets the range of values which the control sends to the application via CSN_TRACKING and CSN_CHANGE messages.

Parameters

param1

Low (SHORT)
The minimum value of the circular slider.

param2

High (SHORT)
The maximum value of the circular slider.

Returns

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_SETVALUE

This message sets the current value of the circular slider control.

Parameters

param1

Value (SHORT)
The new value to which to set the circular slider.

param2

ulReserved (ULONG)
Reserved value.

Returns

rc (ULONG)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

SLM_ADDDETENT

This message places a detent along the slider shaft at the position specified on the primary scale. A detent is an indicator that represents a predefined value for a quantity. It does not have to correspond to an increment of the slider.

Parameters

param1

usDetentPos (USHORT)
Detent position.

Number of pixels the detent is positioned from home.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulDetentId (ULONG)
Detent ID.

SLM_QUERYDETENTPOS

This message queries for the current position of a detent.

Parameters

param1

ulDetentId (ULONG)
Detent ID.

Unique detent identifier, which indicates the position to be returned.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ReturnCode

usDetentPos (USHORT)
Detent position.

Number of pixels the detent is positioned from home.

>= 0	Number of pixels the detent is positioned from home.
SLDERR_INVALID_PARAMETERS	An error occurred. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS.

fDetentLocation (USHORT)
Scale.

The scale along which the detent is located. One of the following:

SMA_SCALE1	Detent position is along scale 1.
SMA_SCALE2	Detent position is along scale 2.

SLM_QUERYSCALETEXT

This message queries for the text associated with a tick mark for the primary scale and copies that text into a buffer.

Parameters

param1

usTickNum (USHORT)
Tick location.

Tick location to query for the text.

usBufLen (USHORT)
Buffer length.

Length of the buffer to copy the text into. The buffer size should include space for the null termination character.

param2

pTickText (PSZ)
Pointer to the buffer into which to place the text string for the tick mark.

Returns

sTextLen (SHORT)
Count of bytes.

>= 0	Length of the text string, excluding the null termination character.
SLDERR_INVALID_PARAMETERS	An error occurred. The WinGetLastError function may return the following errors: • PMERR_INVALID_PARAMETERS • PMERR_PARAMETER_OUT_OF_RANGE.

SLM_QUERYSLIDERINFO

This message queries the current position or dimensions of a key component of the slider. The information returned and its format depends on the type of information requested.

Parameters

param1

usInfoType (USHORT)
Information attribute.

Attribute that identifies the requested information. It can be one of the following:

SMA_SHAFTDIMENSIONS	Queries for the length and breadth of the slider shaft.
SMA_SHAFTPOSITION	Queries for the x-, y-position of the lower-left corner of the slider shaft.
SMA_SLIDERARMDIMENSIONS	Queries for the length and breadth of the slider arm.
SMA_SLIDERARMPOSITION	Queries for the position of the slider arm. The position can be returned either as an increment position or a range value.

usArmPosType (USHORT)
Format attribute.

Attribute that identifies the format in which the information should be returned if the slider arm position is requested. This value is ignored for all other queries and is one of the following:

SMA_RANGEVALUE	The value returned represents the number of pixels between the home position and the current arm position in the low order byte. The high order byte represents the pixel count of the entire range of the slider control.
SMA_INCREMENTVALUE	The value returned represents an increment position using the primary scale.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulInfo (ULONG)
Return information.

SLM_QUERYTICKPOS

This message queries for the current position of a tick mark for the primary scale. This represents where the tick mark would be located. The tick mark does not have to have a size (that is, to be visible) to use this message.

Parameters

param1

usTickNum (USHORT)
Tick mark location.

Specifies the tick mark location to query for the position.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ReturnCode

xTickPos (USHORT)
X-coordinate.

X-coordinate of the point that represents the position of the tick mark. It is the starting position of the tick mark and represents the end of the tick mark closest to the slider shaft.

yTickPos (USHORT)
Y-coordinate.

Y-coordinate of the point that represents the position of the tick mark. It is the starting position of the tick mark and represents the end of the tick mark closest to the slider shaft.

If NULL is returned in either parameter, an error occurred. The WinGetLastError function may return the following error:

PMERR_PARAMETER_OUT_OF_RANGE.

SLM_QUERYTICKSIZE

This message queries for the size of a tick mark for the primary scale. All tick marks default to a size of 0 (invisible) if not set by the application with the SLM_SETTICKSIZE message.

Parameters

param1

usTickNum (USHORT)
Tick mark location.

Specifies the tick mark location to query for the size.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

usTickSize (USHORT)
Tick mark length.

SLM_REMOVEDETENT

This message removes a previously specified detent. A detent is an indicator that represents a predefined value for a quantity and does not have to correspond to an increment of the slider.

Parameters

param1

ulDetentId (ULONG)
Detent ID.

Unique detent identifier for the detent that is to be removed from the slider.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE Detent was successfully removed.

FALSE An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

SLM_SETSCALETEXT

This message sets text above a tick mark for the primary scale. A tick mark does not have to be visible to have text set above it. The text is centered on the tick mark.

Parameters

param1

usTickNum (USHORT)
Tick mark location.

Specifies the tick mark location that is to have the text placed with it.

param2

pTickText (PSZ)
Pointer to the text that is to be drawn at the position specified.

If this value is NULL, no text is drawn.

Returns

rc (BOOL)
Success indicator.

TRUE Text was successfully added to the scale.

FALSE An error occurred. The WinGetLastError function may return the following errors:

- PMERR_HEAP_MAX_SIZE_REACHED
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_SETSLIDERINFO

This message sets the current position or dimensions of a key component of the slider. The component to be changed is indicated by one parameter and the new value is placed in the other.

Parameters

param1

usInfoType (USHORT)
Component attribute.

Identifies the slider component that is to be modified. Specify one of the following:

SMA_SHAFTDIMENSIONS	Sets the width (for vertical sliders) or height (for horizontal sliders) of the slider shaft.
SMA_SHAFTPOSITION	Sets the x-, y-position of the lower-left corner of the slider shaft in the slider window.
SMA_SLIDERARMDIMENSIONS	Sets the width and height of the slider arm.
SMA_SLIDERARMPOSITION	Sets the position of the slider arm. This value can be specified either as an increment position or a range value.

usArmPosType (USHORT)
Format attribute.

Identifies the format in which the information should be interpreted by the slider if setting the slider arm position is requested. This value is a reserved field for other set requests. The format is one of the following:

SMA_RANGEVALUE	Number of pixels between the home position and the current arm position.
SMA_INCREMENTVALUE	Increment position using the primary scale.

parm2

ulInfo (ULONG)
New value.

New value to change the slider component to. The format of the information depends on the component being changed and is indicated by the SMA_* message attribute or attributes that are set.

- If the SMA_SHAFTDIMENSIONS attribute is set, the *ulInfo* parameter is as follows:

  usShaftBreadth (USHORT)
  Width (for vertical sliders) or height (for horizontal sliders) the slider shaft should be set to, in pixels. This is the breadth the shaft should be.

- If the SMA_SHAFTPOSITION attribute is set, the *ulInfo* parameter is as follows:

  xShaftCoord (USHORT)
  X-coordinate to set the position of the shaft to within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the shaft.

  yShaftCoord (USHORT)
  Y-coordinate to set the position of the shaft to within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the shaft.

- If the SMA_SLIDERARMDIMENSIONS attribute is set, the *ulInfo* parameter is as follows:

  usArmLength (USHORT)
  Length of the slider arm, in pixels. This is the width of the arm for horizontal sliders and the height of the arm for vertical sliders.

  usArmBreadth (USHORT)
  Breadth of the slider arm, in pixels. This is the height of the arm for horizontal sliders and the width of the arm for vertical sliders.

- If the SMA_SLIDERARMPOSITION and SMA_RANGEVALUE attributes are set, the *ulInfo* parameter is as follows:

  usArmPos (USHORT)
  Number of pixels to be set from home to the slider arm.

- If the SMA_SLIDERARMPOSITION and SMA_INCREMENTVALUE attributes are set, the *ulInfo* parameter is as follows:

  usIncrementPos (USHORT)
  Increment value which corresponds to the position the slider arm should be set to.

Returns

rc (BOOL)
Success indicator.

TRUE Slider component was successfully set.

FALSE An error occurred. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_SETTICKSIZE

This message sets the size of a tick mark for the primary scale. All tick marks are initially set to a size of 0 (invisible). Each tick mark along a scale can be set to the size desired.

Parameters

param1

usTickNum (USHORT)
Tick mark location.

Tick mark location whose size is to be changed. If the SMA_SETALLTICKS attribute is specified for this parameter, all tick marks on the primary scale are set to the size specified.

usTickSize (USHORT)
Tick mark length.

Length of the tick mark, in pixels. If set to 0, the tick mark will not be drawn.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE — Tick mark position was successfully set.

FALSE — An error occurred. The WinGetLastError function may return the following errors:

- PMERR_HEAP_MAX_SIZE_REACHED
- PMERR_PARAMETER_OUT_OF_RANGE.

WM_PRESPARAMCHANGED (in Slider Controls)

For the cause of this message, see WM_PRESPARAMCHANGED.

Parameters

param1

attrtype (ULONG)
Attribute type.

Presentation parameter attribute identity. The following presentation parameters are initialized by the slider control. The initial value of each is shown in the following list:

PP_FOREGROUNDCOLOR or PP_FOREGROUNDCOLORINDEX
Item foreground color; used when displaying text and bit maps. This color is initialized to SYSCLR_WINDOWTEXT.

PP_BACKGROUNDCOLOR or PP_BACKGROUNDCOLORINDEX
Slider background color; used for entire control as the background. This color is initialized to SYSCLR_WINDOW.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, must be 0.

WM_QUERYWINDOWPARAMS (in Slider Controls)

For the cause of this message, see WM_QUERYWINDOWPARAMS.

Parameters

param1

pwndparams (PWNDPARAMS)
Pointer to a WNDPARAMS window parameter structure.

This structure contains:

status (USHORT)
Window parameter selection.

Identifies the window parameters that are to be set or queried. Valid values for the slider control are:

WPM_CBCTLDATA	Window control data length.
WPM_CTLDATA	Window control data.

The flags in the *status* field are cleared as each item is processed. If the call is successful, the *status* field is 0. If any item has not been processed, the flag for that item is still set.

length (USHORT)
Length of the window text.

text (PSZ)
Window text.

presparamslength (USHORT)
Length of presentation parameters.

presparams (PVOID)
Presentation parameters.

ctldatalength (USHORT)
Length of window class-specific data.

ctldata (PVOID)
Window class-specific data.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WM_SETWINDOWPARAMS (in Slider Controls)

For the cause of this message, see WM_SETWINDOWPARAMS.

Parameters

param1

pwndparams (PWNDPARAMS)
Pointer to a WNDPARAMS window parameter structure.

This structure contains:

status (USHORT)
Window parameter selection.

Identifies the window parameters that are to be set or queried. The valid value for the slider control is:

WPM_CTLDATA Window control data.

The flags in the *status* field are cleared as each item is processed. If the call is successful, the *status* field is 0. If any item has not been processed, the flag for that item is still set.

length (USHORT)
Length of the window text.

text (PSZ)
Window text.

presparamslength (USHORT)
Length of presentation parameters.

presparams (PVOID)
Presentation parameters.

ctldatalength (USHORT)
Length of window class-specific data.

ctldata (PVOID)
Window class-specific data.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

Related Notification Messages

This section covers the notification messages that are related to linear and circular slider controls.

WM_CONTROL (in Circular Slider Controls)

This message occurs when a control has a significant event to notify to its owner.

Parameters

param1

usID (USHORT)
Control-window identity.

The identity of the circular slider that generated the notification.

usnotifycode (USHORT)
Notification code.

The notification codes that indicate what action has occurred.

CSN_SETFOCUS	This code returns a Boolean indicating whether the circular slider control sending the notification message is gaining or losing the focus. *param2* contains TRUE if the control is gaining the focus.
CSN_CHANGED	This code is sent to notify the application that the circular slider value has been changed. *param2* contains the new value of the circular slider.
CSN_TRACKING	This code is sent to notify the application that the circular slider is being tracked by the mouse. *param2* contain the inter-media value of the circular slider. Inter-media values are not necessarily contiguous.
CSN_QUERYBACKGROUNDCOLOR	This code gives the application the opportunity to set the background color of the circular slider. CLR_* or SYSCLR_* values can be returned for the background color. *param2* is NULL.

param2

ulnotifyspec (ULONG)
Notify control-specific information.

Returns

ulReserved (ULONG)
Reserved value.

WM_CONTROL (in Slider Controls)

For the cause of this message, see WM_CONTROL.

Parameters

param1

id (USHORT)
Slider control identity.

notifycode (USHORT)
Notification code.

The slider control uses these notification codes:

SLN_CHANGE	The slider arm position has changed.
SLN_KILLFOCUS	The slider control is losing the focus.
SLN_SETFOCUS	The slider control is receiving the focus.
SLN_SLIDERTRACK	The slider arm is being dragged, but has not been released.

param2

notifyinfo (ULONG)
Control-specific information.

When the value of the *notifycode* parameter is SLN_CHANGE or SLN_SLIDERTRACK, this value is the new arm position, expressed as the number of pixels from the home position.

Otherwise, this value is the window handle (HWND) of the slider control.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_CONTROLPOINTER

This message is sent to a owner window of a control when the pointing device pointer moves over the control window, allowing the owner to set the pointing device pointer.

Parameters

param1

usidCtl (USHORT)
Control identifier.

param2

hptrNew (HPOINTER)
Handle of the pointing device pointer that the control is to use.

Returns

hptrRet (HPOINTER)
Returned pointing device-pointer handle that is then used by the control.

WM_DRAWITEM

This notification is sent to the owner of a control each time an item is to be drawn.

Parameters

param1

idIdentity (USHORT)
Window identifier.

The window identity of the control sending this notification message.

param2

ulcontrolspec (ULONG)
Control-specific information.

The meaning of the control-specific information depends on the type of control. For details of each control type, refer to the appropriate section.

Returns

rc (BOOL)

Item-drawn indicator.

TRUE	The owner has drawn the item, and so the control does not draw it.
FALSE	If the item contains text and the owner does not draw the item, the owner returns this value and the control draws the item.

Related Data Structures

This section covers the data structures that are related to linear and circular slider controls.

CSBITMAPDATA

This is the bit-map data structure for the circular slider buttons.

Syntax

```
typedef struct _CSBITMAPDATA {
HBITMAP       hbmLeftUp;
HBITMAP       hbmLeftDown;
HBITMAP       hmbRightUp;
HBITMAP       hbmRightDown;
 } CSBITMAPDATA;

typedef CSBITMAPDATA *PCSBITMAPDATA;
```

Fields

hbmLeftUp (HBITMAP)
Handle to the "up" position bit map for the button on the left.

hbmLeftDown (HBITMAP)
Handle to "down" position bit map for the button on the left.

hmbRightUp (HBITMAP)
Handle to the "up" position bit map for the button on the right.

hbmRightDown (HBITMAP)
Handle to the "down" position bit map for the button on the right.

SLDCDATA

Slider control data structure.

Syntax

```
typedef struct _SLDCDATA {
ULONG        cbSize;
USHORT       usScale1Increments;
USHORT       usScale1Spacing;
USHORT       usScale2Increments;
USHORT       usScale2Spacing;
 } SLDCDATA;

typedef SLDCDATA *PSLDCDATA;
```

Fields

cbSize (ULONG)
Data length.

Length of the control data in bytes.

usScale1Increments (USHORT)
Scale increments.

The number of increments to set for the slider control. This number represents the range of values that can be selected within the slider when the SLS_PRIMARYSCALE1 style bit is specified.

usScale1Spacing (USHORT)
Scale spacing.

The spacing between increments, expressed in pixels. It represents the unit that is the smallest division of the scale when the SLS_PRIMARYSCALE1 style bit is specified. If 0 is specified, the slider automatically calculates the spacing based on the window size and the number of increments specified.

usScale2Increments (USHORT)
Alternate scale increments.

An alternate number of increments to set for the slider control. This number represents the range of values that can be selected within the slider when the SLS_PRIMARYSCALE2 style bit is specified.

usScale2Spacing (USHORT)
Alternate scale spacing.

An alternate spacing between increments, expressed in pixels. It represents the unit that is the smallest division of the scale when the SLS_PRIMARYSCALE2 style bit is specified. If 0 is specified, the slider automatically calculates the spacing based on the window size and the number of increments specified.

Summary

Following are tables that describe the OS/2 functions, window messages, notification messages, notification codes, and data structure used with (linear and circular) slider controls:

Table 6-3. Linear Slider Control Functions

Function Name	Description
WinCreateWindow	Creates a window.
WinSendMsg	Sends a message with identity *Msgid* to *hwnd*.
WinShowWindow	Sets the visibility state of a window.

Table 6-4. Linear Slider Control Window Messages

Message Name	Description
SLM_ADDDETENT	Places a detent along the slider shaft at the position specified on the primary scale.
SLM_QUERYDETENTPOS	Queries for the current position of a detent.
SLM_QUERYSCALETEXT	Queries for the text associated with a tick mark for the primary scale and copies that text into a buffer.
SLM_QUERYSLIDERINFO	Queries the current position or dimensions of a key component of the slider.
SLM_QUERYTICKPOS	Queries for the current position of a tick mark for the primary scale.
SLM_QUERYTICKSIZE	Queries for the size of a tick mark for the primary scale.
SLM_REMOVEDETENT	Removes a previously specified detent.
SLM_SETSCALETEXT	Sets text above a tick mark for the primary scale.
SLM_SETSLIDERINFO	Sets the current position or dimensions of a key component of the slider.
SLM_SETTICKSIZE	Sets the size of a tick mark for the primary scale.
WM_CHAR	Occurs when the user presses a key.
WM_PRESPARAMCHANGED	Sent when a presentation parameter is set or removed dynamically from a window instance.
WM_QUERYWINDOWPARAMS	Occurs when an application queries the window parameters.
WM_SETWINDOWPARAMS	Occurs when an application sets or changes the window parameters.

Table 6-5. Linear Slider Control Notification Messages

Message Name	Description
WM_CONTROL	Occurs when the linear slider control has a significant event to notify to its owner.
WM_CONTROLPOINTER	Sent to the owner window of the linear slider control when the pointing device pointer moves over the slider control window, enabling the owner window to set the pointer.
WM_DRAWITEM	Sent to the owner of the slider control each time an item is to be drawn.

Table 6-6. Linear Slider Control Notification Codes

Code Name	Description
SLN_CHANGE	Sent when the slider arm position has changed.
SLN_KILLFOCUS	Sent when the slider control is losing the focus.
SLN_SETFOCUS	Sent when the slider control is receiving the focus.
SLN_SLIDERTRACK	Sent when the slider arm is being dragged, but it has not been released.

Table 6-7. Linear Slider Control Data Structure

Data Structure Name	Description
SLDCDATA	Slider control data structure.

Table 6-8 (Page 1 of 2). Circular Slider Control Window Messages

Message Name	Description
CSM_QUERYINCREMENT	Queries the increments used to scroll the value and to draw the tick marks.
CSM_QUERYRADIUS	Queries the current radius of the circular slider.
CSM_QUERYRANGE	Queries the range of values for the circular slider scale.
CSM_QUERYVALUE	Queries the current value of the circular slider.
CSM_SETBITMAPDATA	Replaces the bit maps used for the plus and minus buttons. The optimal size for these bit maps is 10x10 pels.
CSM_SETINCREMENT	Sets the scroll and tick-mark increments of the circular slider.
CSM_SETRANGE	Sets the range of values for the circular slider scale.
CSM_SETVALUE	Sets the current value of the circular slider.
WM_CHAR	Occurs when the user presses a key. **Note:** The keystrokes processed by a circular slider control are the left and right arrows only.

Table 6-8 (Page 2 of 2). Circular Slider Control Window Messages	
Message Name	**Description**
WM_PRESPARAMCHANGED	Sent when a presentation parameter is set or removed dynamically from a window instance.
WM_QUERYWINDOWPARAMS	Occurs when an application queries the window parameters.
WM_SETWINDOWPARAMS	Occurs when an application sets or changes the window parameters.

Table 6-9. Circular Slider Control Notification Messages	
Message Name	**Description**
WM_CONTROL	Occurs when the circular slider control has a significant event to notify to its owner.
WM_CONTROLPOINTER	Sent to the owner window of the circular slider control when the pointing device pointer moves over the slider control window, enabling the owner window to set the pointer.

Table 6-10. Circular Slider Control Notification Codes	
Code Name	**Description**
CSN_CHANGED	The value of the slider is changed.
CSN_QUERYBACKGROUNDCOLOR	The circular slider is about to be painted.
CSN_SETFOCUS	The circular slider is gaining the input focus.
CSN_TRACKING	The value of the slider is being tracked by the mouse.

Table 6-11. Circular Slider Control Data Structure	
Data Structure Name	**Description**
CSBITMAPDATA	Bit-map data structure for the circular slider control.

Chapter 7. Value Set Controls

A *value set control* (WC_VALUESET window class), like a radio button, is a visual component that enables a user to select one choice from a group of mutually exclusive choices. However, unlike radio buttons, a value set can use graphic images (bit maps or icons), as well as colors, text, and numbers, to represent the items a user can select. This chapter presents the basics about value set controls and tells you how to create and use them in PM applications.

About Value Set Controls

Even though text is supported, the purpose of a value set control is to display choices as graphic images for faster selection. The user can *see* the selections instead of having to take time to read descriptions of the choices. Using graphic images in a value set also lets you conserve space on the display screen. For example, if you want to let a user choose from a variety of patterns, you can present those patterns as value set choices, as shown in Figure 7-1, instead of providing a list of radio buttons with a description of each pattern.

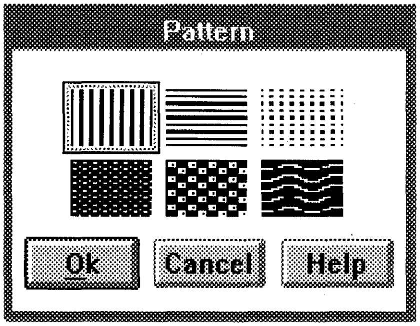

Figure 7-1. Value Set Example

If long strings of data are to be displayed as choices, radio buttons should be used. However, for small sets of numeric or textual information, you can use either a value set or radio buttons.

The value set is customizable to meet varying application requirements, while providing a user interface component that can be used easily to develop products that conform to the Common User Access (CUA) user interface guidelines. The application can specify different types of items, sizes, and orientations for its value sets, but the underlying function of the control remains the same. For a complete description of CUA value sets, refer to the *SAA CUA Guide to User Interface Design* and the *SAA CUA Advanced Interface Design Reference*.

Value Set Styles

Value set control window styles are set when a value set window is created.

- Set one of the following styles when creating a value set control window. You can override these styles by specifying VIA_BITMAP, VIA_ICON, VIA_TEXT, VIA_RGB, or VIA_COLORINDEX attributes for individual value set items.

VS_BITMAP The attribute for each value set item is set to the VIA_BITMAP value set item attribute, which means the value set treats each item as a bit map unless otherwise specified. This is the default. Figure 7-2 provides an example of a value set with bit maps.

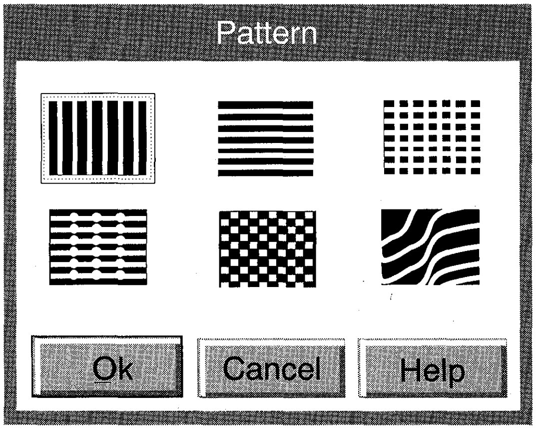

Figure 7-2. Value Set with Bit Maps

VS_COLORINDEX The attribute for each value set item is set to the VIA_COLORINDEX value set item attribute, which means the value set treats each item as an index into the logical color table unless otherwise specified. This style is most often used when the colors currently available are adequate. Figure 7-3 on page 7-3 provides an example of a value set with colors.

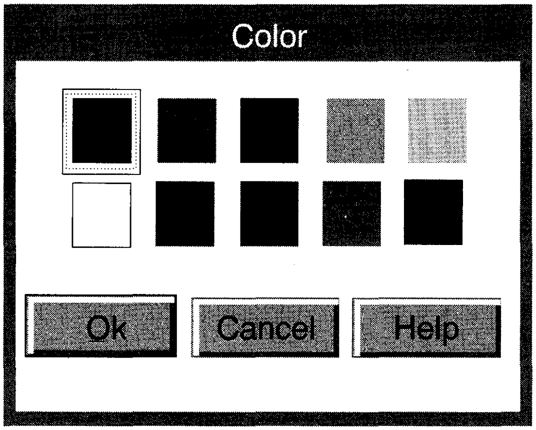

Figure 7-3. Value Set with Colors

VS_ICON The attribute for each value set item is set to the VIA_ICON value set item attribute, which means the value set treats each item as an icon unless otherwise specified. Figure 7-4 provides an example of a value set with icons.

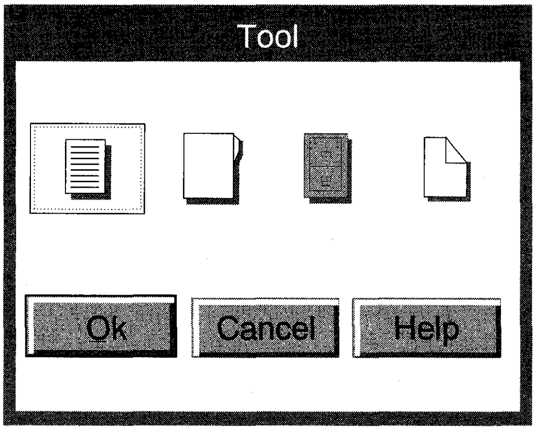

Figure 7-4. Value Set with Icons

VS_RGB The attribute for each value set item is set to the VIA_RGB value set item attribute, which means the value set treats each item as a RGB color value unless otherwise specified. This style is most often used when you need to create new colors. Figure 7-3 provides an example of a value set with colors.

VS_TEXT The attribute for each value set item is set to the VIA_TEXT value set item attribute, which means the value set treats each item as a text string unless otherwise specified. Figure 7-5 on page 7-4 provides an example of a value set with text strings.

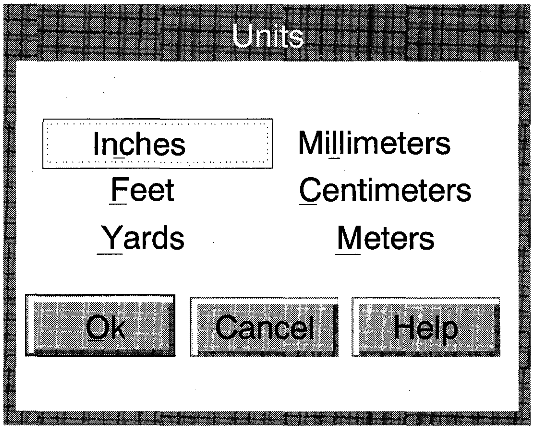

Figure 7-5. Value Set with Text Strings

- Specify one or more of the following optional window styles, if desired, by using an OR operator (|) to combine them with the style specified from the preceding list:

VS_BORDER The value set draws a thin border around itself to delineate the control. Figure 7-6 provides an example of a value set with a border.

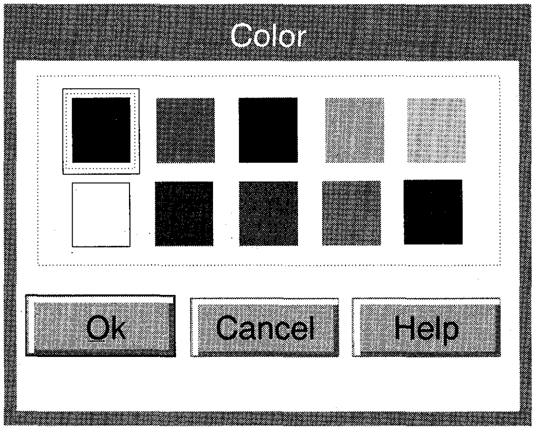

Figure 7-6. Value Set with Border

VS_ITEMBORDER The value set draws a thin border around each item to delineate it from other items.

Note: The VS_ITEMBORDER style is useful for items that are hard to see, such as faint colors or patterns. Figure 7-7 on page 7-5 provides an example of a value set with item borders.

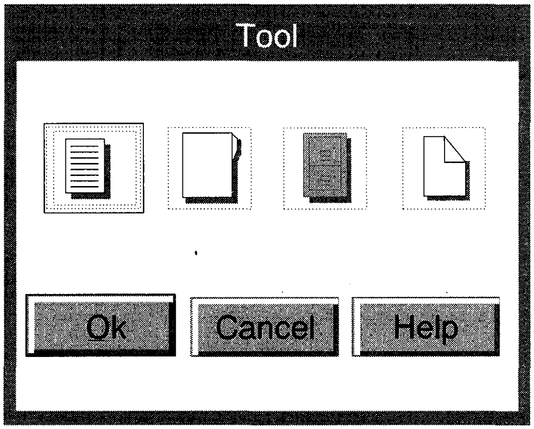

Figure 7-7. Value Set with Item Borders

VS_OWNERDRAW The application is notified whenever the background of the value set window is to be painted.

VS_RIGHTTOLEFT The value set interprets column orientation as right-to-left, instead of the default left-to-right arrangement. This means columns are numbered from right-to-left with the rightmost column being 1 and counting up as you move left. Home is the rightmost column and end is the leftmost column.

There is no visible difference between a value set ordered left-to-right and a value set ordered right-to-left. Therefore, if your application uses multiple value sets, the ordering of the items should be consistent in each value set to avoid confusing the user.

Note: The VS_RIGHTTOLEFT style is used on creation of the control. Changing this style after creation causes unexpected results.

VS_SCALEBITMAPS The value set automatically scales bit maps to the size of the cell. If this style is not used, each bit map is centered in its cell. Also, if the cell is smaller than the bit map, the bit map is clipped to the size of the cell.

Using Value Set Controls

This section provides information that will enable you to create and use a value set control effectively.

Creating a Value Set

You create a value set by using the WC_VALUESET window class name in the *ClassName* parameter of WinCreateWindow call.

Before the value set is created, a temporary *VSCDATA* data structure is allocated so that the number of rows and columns of the value set can be specified.

Also, VS_* values are specified in the *ulValueSetStyle* variable so that the value set can be customized. The sample code illustrated in Figure 7-8 shows the creation of a value set.

```
VSCDATA vscData;                    /* VSCDATA data structure          */
HWND    hwndValueSet;               /* Value set window handle         */
ULONG   ulValueSetStyle;            /* Value set style variable        */

/**********************************************************************/
/* Initialize the parameters in the data structure.                   */
/**********************************************************************/
vscData.cbSize =                    /* Size of value set equals size   */
  sizeof(VSCDATA);                  /* of VSCDATA                      */
vscData.usRowCount = 1;             /* 1 row in the value set          */
vscData.usColumnCount = 3;          /* 3 columns in the value set      */

/**********************************************************************/
/* Set the VS_* style flags to customize the value set.               */
/**********************************************************************/
ulValueSetStyle =
  VS_RGB        |                   /* Use colors for items.           */
  VS_ITEMBORDER |                   /* Put border around each value    */
                                    /* set item.                       */
  VS_BORDER;                        /* Put border around the entire    */
                                    /* value set                       */
```

Figure 7-8 (Part 1 of 3). Sample Code for Creating a Value Set

```
/*********************************************************************/
/* Create the value set control window.                              */
/* The handle of the window is returned in hwndValueSet.             */
/*********************************************************************/
hwndValueSet = WinCreateWindow(
                 hwndClient,          /* Parent window handle          */
                 WC_VALUESET,         /* Value set class name          */
                 (PSZ)NULL,           /* No window text                */
                 ulValueSetStyle,     /* Value set styles              */
                 (SHORT)10,           /* X coordinate                  */
                 (SHORT)10,           /* Y coordinate                  */

                 (SHORT)300,          /* Window width                  */
                 (SHORT)200,          /* Window height                 */
                 hwndClient,          /* Owner window handle           */
                 HWND_TOP,            /* Z-order position              */
                 ID_VALUESET,         /* Value set window ID           */
                 &vscData,            /* Control data structure        */
                 (PVOID)NULL);        /* No presentation parameters    */

/*********************************************************************/
/* Set the color value for each item in each row and column.         */
/*********************************************************************/
WinSendMsg(hwndValueSet,              /* Value set window handle       */
           VM_SETITEM,                /* Message for setting items     */
           MPFROM2SHORT(1,1),         /* Set item in row 1, column 1   */
           MPFROMLONG(0x00FF0000));   /* to the color red.             */

WinSendMsg(hwndValueSet,              /* Value set window handle       */
           VM_SETITEM,                /* Message for setting items     */
           MPFROM2SHORT(1,2),         /* Set item in row 1, column 2   */
           MPFROMLONG(0x0000FF00));   /* to the color green.           */

WinSendMsg(hwndValueSet,              /* Value set window handle       */
           VM_SETITEM,                /* Message for setting items     */
           MPFROM2SHORT(1,3),         /* Set item in row 1, column 3   */
           MPFROMLONG(0x000000FF));   /* to the color blue.            */
```

Figure 7-8 (Part 2 of 3). Sample Code for Creating a Value Set

```
/**********************************************************************/
/* Set the default selection.                                         */
/**********************************************************************/
WinSendMsg(hwndValueSet,            /* Value set window handle       */
           VM_SELECTITEM,           /* Message for selecting items   */
           MPFROM2SHORT(1,2),       /* Item in row 1, column 2       */
           NULL);                   /* Reserved value                */

/**********************************************************************/
/* Since all items have been set in the control,                      */
/* make the control visible.                                          */
/**********************************************************************/
WinShowWindow(hwndValueSet,         /* Value set window handle       */
              TRUE);                /* Make the window visible       */
```

Figure 7-8 (Part 3 of 3). Sample Code for Creating a Value Set

Retrieving Data for Selected Value Set Items

The next step is to be able to retrieve the data represented by a value set item. To do this, variables are specified for combined row and column index values, item attributes, and item information. Then the VM_QUERYSELECTEDITEM, VM_QUERYITEMATTR, and VM_QUERYITEM messages are used to retrieve the index values, attributes, and data. The sample code in Figure 7-9 shows how data for selected value set items is retrieved.

```
ULONG  ulIdx;                       /* Combined row and column       */
                                    /* index value                   */
USHORT usItemAttr;                  /* Item attributes               */
ULONG  ulItemData;                  /* Item data                     */

/**********************************************************************/
/* Get the row and column index values of the item selected by the    */
/* user.  These values are returned in the ulIdx parameter.           */
/**********************************************************************/
ulIdx = (ULONG)WinSendMsg(
  hwndValueSet,                     /* Value set window handle       */
  VM_QUERYSELECTEDITEM,             /* Message for querying          */
                                    /* the selected item             */
  NULL, NULL);                      /* Reserved values               */
```

Figure 7-9 (Part 1 of 2). Sample Code for Retrieving Data for Value Set Items

```
/*********************************************************************/
/* Determine the type of item that was selected. This message is     */
/* only to determine how to interpret item data when a value set     */
/* contains different types of items.                                */
/*********************************************************************/
usItemAttr = (USHORT)WinSendMsg(
  hwndValueSet,                /* Value set window handle            */
  VM_QUERYITEMATTR,            /* Message for querying item attribute */
  MPFROMLONG(ulIdx),           /* Row and column of selected item    */
  NULL);                       /* Reserved value                     */

/*********************************************************************/
/* Get the information about the selected (non-textual) item.        */
/* If you are dealing with text, you need to allocate a buffer       */
/* for the text string.                                              */
/*********************************************************************/
ulItemData = (ULONG)WinSendMsg(
  hwndValueSet,                /* Value set window handle            */
  VM_QUERYITEM,                /* Message for querying an item       */
  MPFROMLONG(ulIdx),           /* Row and column of selected item    */
  NULL);                       /* Set to NULL because the item is not */
                               /* a text item                        */
```

Figure 7-9 (Part 2 of 2). Sample Code for Retrieving Data for Value Set Items

Arranging Value Set Items

The application defines the arrangement of value set items; they can be arranged in one or more rows, columns, or both. Items are placed from left to right in rows and from top to bottom in columns. The application can change the number of rows and columns at any time.

The number of items that can be displayed depends on the number of items that fit into the spaces provided by the defined rows and columns. If the number of items exceeds the number of spaces, the excess items are not displayed.

You can change the composition of a value set by specifying new items. The new items either can be added to the value set or can replace existing items.

Graphical User Interface Support for Value Set Controls

This section describes the support the value set control provides for graphical user interfaces (GUIs). Except where noted, this support conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

The GUI support provided by the value set control consists of Navigating to and selecting value set items.

Value Set Navigation Techniques

Since all value set items are mutually exclusive, only one of them can be selected at a time. Therefore, the only type of selection supported by the value set control is *single selection*.

Note: If more than one value set window is open, navigating to and selecting items in one value set window has no affect on the items displayed in any other value set window.

An initial choice is selected when the value set control is first displayed. If the application does not provide the initial selection by using the VM_SELECTITEM message, the choice in row 1, column 1 is selected automatically.

The value set control supports the use of a pointing device, such as a mouse, and the keyboard for navigating to and selecting items, except for items that are dimmed on the screen. This dimming of items is called *unavailable-state emphasis* and indicates that the items cannot be selected. However, the *selection cursor*, a dotted outline that usually indicates that an item can be selected, can be moved to unavailable items so that a user can press F1 to determine why they cannot be selected. The following sections describe the pointing device and keyboard support for the value set control.

Pointing Device Support

A user can use a pointing device to select value set items. The *SAA CUA Guide to User Interface Design* defines mouse button 1, the *select* button, to be used for selecting items. This definition also applies to the same button on any other pointing device.

An item can be selected by moving the pointer of the pointing device to the item and clicking the select button. When this happens, a black box is drawn around the item to show that it has been selected. The black box is called *selected-state emphasis*. In addition, the selection cursor is drawn inside the black box.

Keyboard Support

The value set control supports *automatic selection*, which means that an available item is selected when the selection cursor is moved to that item. The item is given selected-state emphasis as soon as the selection cursor is moved to it. No further action, such as pressing the spacebar, is required. The same black box and dotted outline are used, for selected-state emphasis and the selection cursor respectively, as when an item is selected with a pointing device.

A user can navigate to and select an item by using either the navigation keys or mnemonic selection to move the selection cursor to the item, as described in the following list:

- Items can be selected using the Up, Down, Left, and Right Arrow keys to move the selection cursor from one item to another.
- The Home and End keys can be used to select the leftmost and rightmost items, respectively, in the current row. If the Ctrl key is pressed in combination with the Home or End key, the item in the top row and the leftmost column, or the item in the bottom row and the rightmost column, respectively, is selected.

Note: The preceding description assumes that the current style of the value set window is left-to-right. However, if the VS_RIGHTTOLEFT style bit is set, the directions described for the Home, End, Ctrl+Home, and Ctrl+End keys in the preceding paragraph are reversed.

- The PgUp key can be used to select the item in the top row that is directly above the current position of the selection cursor. The PgDn key can be used to select the item in the bottom row that is directly below the current position of the selection cursor. If the space in the top or bottom row directly above or below the current cursor position is blank, the cursor moves to the blank space.
- Another keyboard method of selecting items is *mnemonic selection*. A user performs mnemonic selection by pressing a character key that corresponds to an underlined character. Coding a tilde (~) before a text character in the item causes that character to be underlined and activates it as a mnemonic selection character. When this happens, the selection cursor is moved to the item that contains the underlined character, and that item is selected.

Enhancing Value Set Controls Performance and Effectiveness

This section provides dynamic resizing and scrolling to enable you to fine-tune a value set control.

Dynamic Resizing and Scrolling

The value set control supports *dynamic resizing* if the application sends the WM_SIZE message to a value set window. This means that the value set control automatically recalculates the size of the items when either the user or the application changes the size of the value set window.

If the value set window's size is decreased so that the window is not large enough to display all of the items the value set contains, the items are clipped. If scroll bars are desired to allow the clipped information to be scrolled into view, they must be provided by the application.

Related Window Messages

This section covers the window messages that are related to value set controls.

VM_QUERYITEM

This message queries the contents of the item indicated by the values of the *usRow* and *usColumn* fields. The information returned is interpreted based on the attribute of the item.

Parameters

param1

usRow (USHORT)
Row index.

Row index of the item to be queried. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

usColumn (USHORT)
Column index.

Column index of the item to be queried. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

param2

pvsText (PVSTEXT)
Pointer to a VSTEXT data structure or NULL.

If the attribute of the item to query is VIA_TEXT, the value of the *param2* parameter is the same as the value of the *pvsText* field. For all other attributes, the *param2* parameter is reserved and should be set to a NULL value.

See "VSTEXT" on page 7-30 for definitions of this structure's fields as they apply to the VM_QUERYITEM message.

Returns

ulItemId (ULONG)
Item information.

VM_QUERYITEMATTR

This message queries the attribute or attributes of the item indicated by the values of the *usRow* and *usColumn* fields.

Parameters

param1

usRow (USHORT)
Row index.

Row index of the item for which the attribute or attributes are queried. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

usColumn (USHORT)
Column index.

Column index of the item for which the attribute or attributes are queried. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

usItemAttr (USHORT)
Item information.

VM_QUERYMETRICS

This message queries for the current size of each value set item or for the spacing between items. The value returned is either the width and height of one item, or the spacing between items.

Parameters

param1

fMetric (USHORT)
Control metric.

Control metric to be queried with this message. This can be either of the following:

VMA_ITEMSIZE	If this message attribute is set, the width and height of each item (in pixels) are returned in the *usItemWidth* and *usItemHeight* parameters, respectively.
VMA_ITEMSPACING	If this message attribute is set, the horizontal and vertical spacing between items (in pixels) is returned in the *usHorzItemSpacing* parameter and in the *usVertItemSpacing* parameter, respectively.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulMetric (ULONG)
Metric value queried for.

VSERR_INVALID_PARAMETERS
An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

>= 0
This value depends on the VMA_* attribute set in the *param1* parameter.

- If the VMA_ITEMSIZE attribute is set, the following is returned:

  usItemWidth (USHORT)
  Width of one value set item, in pixels.

  usItemHeight (USHORT)
  Height of one value set item, in pixels.

- If the VMA_ITEMSPACING attribute is set, the following is returned:

  usHorzItemSpacing (USHORT)
  : Amount of horizontal space allocated between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically allocated by the value set control. The default space amount is 0.

  usVertItemSpacing (USHORT)
  : Amount of vertical space allocated between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically allocated by the value set control. The default space amount is 0.

VM_QUERYSELECTEDITEM

This message queries for the currently selected value set item indicated by the values of the *usRow* and *usColumn* fields.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ReturnCode

usRow (USHORT)
Row index.

Row index of the currently selected value set item. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

usColumn (USHORT)
Column index.

Column index of the currently selected value set item. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

VM_SELECTITEM

This message selects the value set item indicated by the values of the *usRow* and *usColumn* parameters. When a new item is selected, the previously selected item is deselected.

Parameters

param1

usRow (USHORT)
Row index.

Row index of the value set item to select. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

usColumn (USHORT)
Column index.

Column index of the value set item to select. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE Item was successfully selected.

FALSE An error occurred. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SETITEM

This message specifies the type of information that will be contained by a value set item. This item is indicated by the values of the *usRow* and *usColumn* fields. Each value set item can contain a different type of information. The value set interprets the information set for the item based on the attribute of the item. Value set items that are not set (blank items) are drawn using the background color of the value set.

Parameters

param1

usRow (USHORT)
Row index.

Row index of the value set item for which information is being specified. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

usColumn (USHORT)
Column index.

Column index of the value set item for which information is being specified. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

param2

ulItemId (ULONG)
Item information.

This value depends on the VIA_* attribute set for the item.

- If the VIA_TEXT attribute is specified, the *ulItemId* field is as follows:

  pszItem (PSZ)
  Pointer to a null terminated string containing the text to be placed in the item. If NULL is passed in, the item is blank.

- If the VIA_BITMAP attribute is specified, the *ulItemId* field is as follows:

  hbmItem (HBITMAP)
  Handle to a bit map that is to be drawn in the item indicated by the *param1* parameter. If NULLHANDLE is passed in, the item will be blank.

- If the VIA_ICON attribute is specified, the *ulItemId* field is as follows:

  hptItem (HPOINTER)
  Handle to the icon that is to be drawn in the item indicated by the *param1* parameter. If NULLHANDLE is passed in, the item is blank.

- If the VIA_RGB attribute is specified, the *ulItemId* field is as follows:

  rgbItem (ULONG)
  : Color value to be drawn in the item indicated by the *param1* parameter. If an invalid value is passed in (a value greater than 0x00FFFFFF), the item is blank. Each color value is a 4-byte integer with a value of:

    ```
    (R * 65536) + (G * 256) + B
    ```

    where:

    R Red intensity value
    G Green intensity value
    B Blue intensity value.

- If the VIA_COLORINDEX attribute is specified, the *ulItemId* field is as follows:

  ulColorIndex (ULONG)
  : Index of the color in the logical color table to be drawn in the item indicated by the *param1* parameter.

Returns

rc (BOOL)
Success indicator.

TRUE Item was successfully set.

FALSE An error occurred. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SETITEMATTR

This message sets the attribute or attributes of the item indicated by the values of the *usRow* and *usColumn* parameters.

Parameters

param1

usRow (USHORT)
Row index.

Row index of the value set item for which attributes are being specified. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

usColumn (USHORT)
Column index.

Column index of the value set item for which attributes are being specified. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

param2

usItemAttr (USHORT)
Item attributes.

Attribute or attributes of the item to be set or reset based on the value of the *fSet* field. These attributes can be as follows:

- One of the following attributes can be set:

VIA_BITMAP	If this attribute is set, the item is a bit map. This is the default.
VIA_COLORINDEX	If this attribute is set, the item is an index into the logical color table.
VIA_ICON	If this attribute is set, the item is an icon.
VIA_RGB	If this attribute is set, the item is a color entry.
VIA_TEXT	If this attribute is set, the item is a text string.

- In addition, one or more of the following attributes can be set:

VIA_DISABLED	If this attribute is set, the item cannot be selected and is displayed with unavailable-state emphasis, if possible. Unavailable text items are always displayed with unavailable-state emphasis, according to CUA guidelines; for items displayed as color, bit maps, and icons, it is the application's responsibility to determine the best way to show that these items are unavailable, if possible. The selection cursor can be moved to an unavailable item by using either the keyboard navigation keys or a pointing device. This allows a user to press the F1 key to find out why that item cannot be selected.
VIA_DRAGGABLE	If this attribute is set, the item can be the source of a direct manipulation action.
VIA_DROPONABLE	If this attribute is set, the item can be the target of a direct manipulation action.
VIA_OWNERDRAW	If this attribute is set, a paint notification message is sent whenever this item needs painting.

fSet (USHORT)
Set or reset flag.

TRUE   Set the attribute of the indicated item.

FALSE   Turn off the attribute of the indicated item.

Returns

rc (BOOL)
Success indicator.

TRUE   Attribute or attributes were set successfully.
FALSE   An error occurred. The WinGetLastError function may return the following errors:
- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SETMETRICS

This message sets the size of each item in the value set control, the spacing between items, or both.

Parameters

param1

fMetric (USHORT)
Units of measurement.

Unit or units of measurement that are to be set for the value set control. This can be either of the following:

VMA_ITEMSIZE	If this message attribute is set, the width and height of each item is set using the values of the *usItemWidth* and *usItemHeight* parameters, respectively.
VMA_ITEMSPACING	If this message attribute is set, the horizontal and vertical spacing between each item is set using the values of the *usHorzItemSpacing* and *usVertItemSpacing* parameters, respectively.

param2

ulItemId (ULONG)
Item information.

This value depends on the VMA_* attribute set for the message.

- If the VMA_ITEMSIZE attribute is specified, the *ulItemId* field is as follows:

  usItemWidth (USHORT)
  Width to be set for each value set item, in pixels. The number of pixels specified cannot be less than 2.

  usItemHeight (USHORT)
  Height to be set for each value set item, in pixels. The number of pixels specified cannot be less than 2.

- If the VMA_ITEMSPACING attribute is specified, *ulItemId* field is as follows:

  usHorzItemSpacing (USHORT)
  Amount of horizontal space to be set between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically set by the value set control. The default spacing is 0.

  usVertItemSpacing (USHORT)
  Amount of vertical space to be set between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically set by the value set control. The default spacing is 0.

Returns

rc (BOOL)
Success indicator.

TRUE Item size or spacing was successfully set.

FALSE An error occurred. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

WM_PRESPARAMCHANGED (in Value Set Controls)

For the cause of this message, see WM_PRESPARAMCHANGED.

Parameters

param1

attrtype (ULONG)
Attribute type.

Presentation parameter attribute identity. The following presentation parameters are initialized by the value set control. The initial value of each is shown in the following list:

PP_FOREGROUNDCOLOR or PP_FOREGROUNDCOLORINDEX
Item foreground color; used when displaying text and bit maps. This color is initialized to SYSCLR_WINDOWTEXT.

PP_BACKGROUNDCOLOR or PP_BACKGROUNDCOLORINDEX
Value set background color; used for entire control as the background. This color is initialized to SYSCLR_WINDOW.

PP_HILITEBACKGROUNDCOLOR or PP_HILITEBACKGROUNDCOLORINDEX
Selection color; this is the color used for selected-state and target emphasis. This color is initialized to SYSCLR_HILITEBACKGROUND.

PP_BORDERCOLOR or PP_BORDERCOLORINDEX
Value set and item border color. This color is initialized to SYSCLR_WINDOWFRAME.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_QUERYWINDOWPARAMS (in Value Set Controls)

For the cause of this message, see WM_QUERYWINDOWPARAMS.

Parameters

param1

wndparams (PWNDPARAMS)
Pointer to a WNDPARAMS window parameter structure.

See WNDPARAMS for descriptions of the default fields. For a value set, the valid values for the *fsStatus* field are WPM_CBCTLDATA and WPM_CTLDATA.

The flags in the *fsStatus* field are cleared as each item is processed. If the call is successful, the *fsStatus* field is NULL. If any item has not been processed, the flag for that item is still set.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful operation.
FALSE	Error occurred.

WM_SETWINDOWPARAMS (in Value Set Controls)

For the cause of this message, see WM_SETWINDOWPARAMS.

Parameters

param1

wndparams (PWNDPARAMS)
Pointer to a WNDPARAMS structure.

See WNDPARAMS for descriptions of the fields. For a value set, the valid value of the *fsStatus* field is WPM_CTLDATA.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

WM_SIZE

This message occurs when a window changes its size.

Parameters

param1

scxold (SHORT)
Old horizontal size.

scyold (SHORT)
Old vertical size.

param2

scxnew (SHORT)
New horizontal size.

scynew (SHORT)
New vertical size.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

Related Notification Messages

This section covers the notification messages that are related to value set controls.

WM_CONTROL (in Value Set Controls)

For the cause of this message, see WM_CONTROL.

Parameters

param1

id (USHORT)
Value set control identity.

notifycode (USHORT)
Notify code.

The value set control uses these notification codes:

VN_DRAGLEAVE	The value set receives a DM_DRAGLEAVE message.
VN_DRAGOVER	The value set receives a DM_DRAGOVER message.
VN_DROP	The value set receives a DM_DROP message. The VN_DROP notification code is sent only when an item is dropped on an item that has the VIA_DROPONABLE attribute.
VN_DROPHELP	The value set receives a DM_DROPHELP message.
VN_ENTER	The user presses the Enter key while the value set window has the focus or double-clicks the select button while the pointer is over an item in the value set.
VN_HELP	The value set receives a WM_HELP message.
VN_INITDRAG	The drag button was pressed and the pointer was moved while the pointer was over the value set control. The VN_INITDRAG notification code is sent only for items that have the VIA_DRAGGABLE attribute.
VN_KILLFOCUS	The value set is losing the focus.
VN_SELECT	An item in the value set has been selected and is given selected-state emphasis.
VN_SETFOCUS	The value set receives the focus.

param2

notifyinfo (ULONG)
Control-specific information.

When the value of the *notifycode* parameter is VN_DRAGOVER, VN_DRAGLEAVE, VN_DROP, or VN_DROPHELP, this parameter is a pointer to a VSDRAGINFO structure.

When the value of the *notifycode* parameter is VN_INITDRAG, this parameter is a pointer to a VSDRAGINIT structure.

When the value of the *notifycode* parameter is VN_ENTER, VN_HELP, or VN_SELECT, this parameter contains the row and column of the selection cursor. The low-order word contains the row index, and the high-order word contains the column index.

Otherwise, this parameter is the window handle (HWND) of the value set control.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DRAWITEM (in Value Set Controls)

This notification message is sent to the owner of a value set control each time an item that has the VIA_OWNERDRAW attribute is to be drawn, or when the background of a value set window that has the VS_OWNERDRAW style bit is to be drawn.

Parameters

param1

id (USHORT)
Window identifier.

The window identifier of the value set control sending this notification message.

param2

powneritem (POWNERITEM)
Pointer to an OWNERITEM data structure.

The following list defines the OWNERITEM data structure fields that apply to the value set control. See OWNERITEM for the default field values.

hwnd (HWND)
Value set window handle.

hps (HPS)
Presentation-space handle.

fsState (ULONG)
Value set window style flags. See "Value Set Styles" on page 7-2 for descriptions of these style flags.

fsAttribute (ULONG)
Item attribute flags for the indexed item. See "VM_SETITEMATTR" on page 7-18 for descriptions of these attribute flags.

fsStateOld (ULONG)
Reserved.

fsAttributeOld (ULONG)
Reserved.

rclItem (RECTL)
Item rectangle to be drawn in window coordinates.

idItem (LONG)
Identity of component to be drawn.

VDA_BACKGROUND	Specifies that a part of the value set background is to be drawn.
VDA_SURROUNDING	Specifies that a part of the area surrounding the value set is to be drawn.
VDA_ITEMBACKGROUND	Specifies that the background of an item is to be drawn.
VDA_ITEM	Specifies that an entire item is to be drawn.

hItem (ULONG)
If the value of the **identity** parameter is VDA_ITEMBACKGROUND or VDA_ITEM, this is the current row and column index of the item to be drawn. The low-order word contains the row index, and the high-order word contains the column index. Otherwise, this is reserved.

Returns

rc (BOOL)
Item-drawn indicator.

TRUE	The owner draws the component.
FALSE	If the owner does not draw the component, the owner returns this value and the value set control draws the component.

Related Data Structures

This section covers the data structures that are related to value set controls.

VSCDATA

Structure that contains information about the value set control.

Syntax

```
typedef struct _VSCDATA {
ULONG        cbSize;
USHORT       usRowCount;
USHORT       usColumnCount;
 } VSCDATA;

typedef VSCDATA *PVSCDATA;
```

Fields

cbSize (ULONG)
Data length.

Length of the control data in bytes.

usRowCount (USHORT)
Number of rows.

The number of rows in the value set control. The minimum number of rows is 1 and the maximum number of rows is 65,535.

usColumnCount (USHORT)
Number of columns.

The number of columns in the value set control. The minimum number of columns is 1 and the maximum number of columns is 65,535.

VSDRAGINFO

Structure that contains information about direct manipulation actions that occur over the value set control.

Syntax

```
typedef struct _VSDRAGINFO {
PDRAGINFO       pDragInfo;
USHORT          usRow;
USHORT          usColumn;
 } VSDRAGINFO;

typedef VSDRAGINFO *PVSDRAGINFO;
```

Fields

pDragInfo (PDRAGINFO)
Pointer to a DRAGINFO structure.

usRow (USHORT)
Row index.

The index of the row over which the direct manipulation action occurred.

usColumn (USHORT)
Column index.

The index of the column over which the direct manipulation action occurred.

VSDRAGINIT

Structure that contains information that is used to initialize a direct manipulation action over the value set control.

Syntax

```
typedef struct _VSDRAGINIT {
HWND         hwnd;
LONG         x;
LONG         y;
LONG         cx;
LONG         cy;
USHORT       usRow;
USHORT       usColumn;
 } VSDRAGINIT;

typedef VSDRAGINIT *PVSDRAGINIT;
```

Fields

hwnd (HWND)
Value set window handle.

Window handle of the value set control.

x (LONG)
X-coordinate.

X-coordinate of the pointing device pointer in desktop coordinates.

y (LONG)
Y-coordinate.

Y-coordinate of the pointing device pointer in desktop coordinates.

cx (LONG)
X-offset.

X-offset from the hot spot of the pointing device pointer, in pels, to the item origin. The item origin is the lower left corner of the item.

cy (LONG)
Y-offset.

Y-offset from the hot spot of the pointing device pointer, in pels, to the item origin. The item origin is the lower left corner of the item.

usRow (USHORT)
Row index.

The index of the row over which the direct manipulation action occurred.

usColumn (USHORT)
Column index.

The index of the column over which the direct manipulation action occurred.

VSTEXT

Value set text structure. This structure is used with the VM_QUERYITEM message only. See "VM_QUERYITEM" on page 7-12 for information about that message.

Syntax

```
typedef struct _VSTEXT {
PSZ          pszItemText;
ULONG        ulBufLen;
 } VSTEXT;

typedef VSTEXT *PVSTEXT;
```

Fields

pszItemText (PSZ)
: Pointer to a buffer to copy the string into.

ulBufLen (ULONG)
: Buffer size.

  Size of the buffer pointed to by the *pszItemText* field.

Summary

Following are tables that describe the OS/2 functions, window messages, notification messages, notification codes, and data structures used with value set controls:

Table 7-1. Value Set Control Functions

Function Name	Description
WinCreateWindow	Creates a new window.
WinSendMsg	Sends a message to a window.
WinShowWindow	Sets the visibility state of a window.

Table 7-2. Value Set Control Window Messages

Message Name	Description
VM_QUERYITEM	Queries the contents of the item indicated by the row and column values.
VM_QUERYITEMATTR	Queries the attributes of the item indicated by the row and column values.
VM_QUERYMETRICS	Queries the current size of each value set item or the spacing between items.
VM_QUERYSELECTEDITEM	Queries for the currently selected value set item indicated by the row and column values.
VM_SELECTITEM	Selects the value set item indicated by the row and column values.
VM_SETITEM	Specifies the type of information that will be contained by a value set item.
VM_SETITEMATTR	Sets the attributes of the item indicated by the row and column values.
VM_SETMETRICS	Sets the size of each item in the value set control, the spacing between items, or both.
WM_CHAR	Occurs when the user presses a key.
WM_PRESPARAMCHANGED	Sent when a presentation parameter is set or removed dynamically from a window instance.
WM_QUERYWINDOWPARAMS	Occurs when an application queries the window parameters.
WM_SETWINDOWPARAMS	Occurs when an application sets or changes the window parameters.
WM_SIZE	Occurs when a window changes its size.

Table 7-3. Value Set Control Notification Messages

Message Name	Description
WM_CONTROL	Occurs when the value set control has a significant event to notify to its owner.
WM_CONTROLPOINTER	Sent to the owner window of the value set control when the pointing device pointer moves over the value set control window, enabling the pointer to be set.
WM_DRAWITEM	Sent to the owner of the value set control each time an item is to be drawn.

Table 7-4. Value Set Control Notification Codes

Code Name	Description
VN_DRAGLEAVE	Sent when the value set receives a DM_DRAGLEAVE message.
VN_DRAGOVER	Sent when the value set receives a DM_DRAGOVER message.
VN_DROP	Sent when the value set receives a DM_DROPHELP message.
VN_DROPHELP	Sent when the value set receives a DM_DROPHELP message.
VN_ENTER	Sent when the user presses the Enter key while the value set window has the focus, or when the user double-clicks the select button while the pointer is over an item in the value set control.
VN_HELP	Sent when the value set receives a WM_HELP message.
VN_INITDRAG	Sent when the drag button is pressed and the pointer is moved while over the value set control.
VN_KILLFOCUS	Sent when the value set loses the focus.
VN_SELECT	Sent when an item in the value set is selected and given selected-state emphasis.
VN_SETFOCUS	Sent when the value set receives the focus.

Table 7-5. Value Set Control Data Structures

Data Structure Name	Description
VSCDATA	Contains information about the value set control.
VSDRAGINFO	Contains information about direct manipulation actions that occur over the value set control.
VSDRAGINIT	Contains information that is used to initialize a direct manipulation action over the value set control.
VSTEXT	Contains value set text. Used only with the VM_QUERYITEM message.

Chapter 8. Container Controls

A *container* control (WC_CONTAINER window class) is a visual component that holds objects. It provides a powerful and flexible component for easily developing products that conform to the Common User Access (CUA) user interface guidelines. This chapter describes the container control component and how to use it in PM applications.

About Container Controls

A container can display objects in various formats and views. Generally speaking, each view displays different information about each object. If a container's data is too large for the window's *work area*, scrolling mechanisms are enabled. The CUA direct manipulation protocol is fully supported, thereby enabling a user to visually drag an object in a container window and drop it on another object or container window. Containers are an integral component of the CUA user interface.

Container Control Functionality

The container control provides multiple views of a container's contents, such as Icon, Name, Text, Tree, and Details views. The container control lets you change container views quickly and easily, display each view with a different font, or vertically split the Details view into two parts so that a user can widen one part to see more information.

Graphical user interface (GUI) support is part of the container control. GUI support allows:

- Direct manipulation
- Multiple selection types: single, extended, and multiple selections.
- Multiple selection techniques: marquee, swipe, and first-letter selection.
- Multiple selection mechanisms: mouse button 1, mouse button 2, and keyboard augmentation.
- Multiple forms of emphasis: selected-state, unavailable-state, in-use, and target emphasis.
- Scrolling when a container's work area is not large enough for all the container items to be visible
- Dynamic scrolling to provide visible feedback to show the movement of the container items relative to the position of the scroll box.

The container control supports various data types, such as icons or bit maps for the Icon, Name, Tree, and Details views. In the Details view, this includes the ability to use icons or bit maps in column headings as well as in the columns themselves. The container control also supports text in the following situations:

- For container titles in all views
- Beneath icons or bit maps in the Icon view
- To the right of icons or bit maps in the Name and Tree views
- For any column or column heading in the Details view
- For container items in the Text view
- For container items in the Details view, text in date, time, and number format.

The container control provides a variety of options for enhancing the performance of the container:

- Direct editing of container control text
- Blank text fields in all views
- Ownerdraw, which enables an application, rather than the container control, to draw the container items
- Automatic reposition mode which is used in the Icon view. The container control provides an automatic reposition mode that repositions the items as a result of inserting, removing, sorting, filtering items, or changing window or font size.
- Arrange message mode that arranges overlapping icons or bit maps so they no longer overlap
- Data caching to efficiently remove items from and insert items into a container as they scroll in and out of view
- Methods for sharing records among multiple containers
- Memory usage optimization.

Container Items

Container items can be anything that your application or a user might store in a container. Examples are executable programs, word processing files, graphic images, and database records.

Container item data is stored in the RECORDCORE and MINIRECORDCORE data structures. Both the application and the container have access to the data stored in these records.

The application must allocate memory for each record by using the CM_ALLOCRECORD message.

The maximum number of records is limited only by the amount of memory in the user's computer. The container control does not limit the number of records that a container can have.

Container Views

When a user opens a container, the contents of that container are displayed in a window. A container window can present various views of its contents, and each view can provide different information about its container items. Table 8-1 on page 8-3 describes the views the container control provides.

Table 8-1. Container Control Views	
View Type	**Contents Displayed**
Icon view	Displays either icons or bit maps, with text beneath the icons or bit maps, to represent container items. These are called icon/text or bit-map/text pairs. Each icon/text or bit-map/text pair represents one container item. This is the default view.
Name view	Displays either icons or bit maps, with text to the right of the icons or bit maps, to represent container items. These are called icon/text or bit-map/text pairs. Each icon/text or bit-map/text pair represents one container item.
Text view	Displays a simple text list to represent container items.
Tree view	Displays a hierarchical view of the container items. Three types of Tree views are available: Tree text, Tree icon, and Tree name.
Details view	Displays detailed information about each container item. The same type of data is displayed for each container item, arranged in columns. The data in each column can consist of an icon or bit map, text, numbers, dates, or times.

If a text string is not specified for a view in a place where a text string could be used, a blank space is used as a placeholder. For example, if a text string is not placed beneath an icon in the Icon view, a blank space is inserted just as though the text string were there. If this blank space is not a read-only field, the user can put text into the space by editing it directly.

Icon View

The Icon view (CV_ICON attribute) displays icon/text pairs or bit-map/text pairs to represent container items; this is the default view. CV_ICON is an attribute of the CNRINFO data structure's *flWindowAttr* field.

In the Icon view, icon/text pairs and bit-map/text pairs are icons and bit maps, respectively, with one or more lines of text displayed below each icon or bit map. Each line can contain one or more text characters, which are centered below the icon or bit map. The container control does not limit the number of lines or the number of characters in each line.

Generally, the icon or bit map contains an image that depicts the type of container item that it represents. For example, an icon or bit map that represents a bar chart might contain an image of a bar chart.

Because the container control does not support both icons and bit maps in the same view, an application must specify which are used by setting either the CA_DRAWICON attribute or the CA_DRAWBITMAP attribute in the *flWindowAttr* field of the CNRINFO data structure. The default is the CA_DRAWICON attribute. The size of the icon or bit map can be specified in the *slBitmapOrIcon* field of the CNRINFO data structure.

In the Icon view, container items are positioned according to *x*- and *y*-coordinate positions. These are called *workspace coordinates*. You can supply these coordinates for each container item by using the *ptlIcon* field of the RECORDCORE data structure. Figure 8-1 provides an example of the Icon view with various *x*- and *y*- coordinates specified in the *ptlIcon* field.

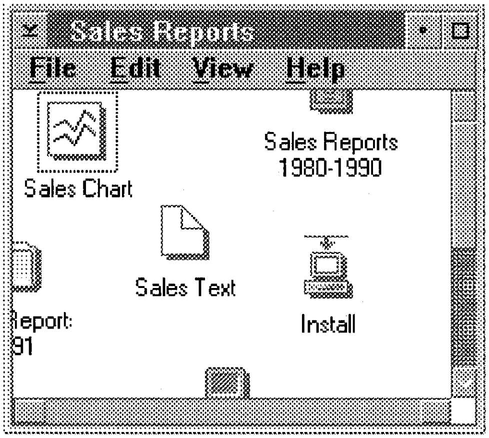

Figure 8-1. Icon View with Items Positioned at Workspace Coordinates

If you do not specify *x*- and *y*-coordinate positions, the container control places the icons or bit maps at (0,0). However, your application can arrange the icons or bit maps either by sending the CM_ARRANGE message or by setting the CCS_AUTOPOSITION style bit when creating a container. With both of these methods, the container items are arranged in rows, and any coordinates specified in the *ptlIcon* field are ignored. As they are arranged each *ptlIcon* is updated with its new location.

The container items fill the topmost row until the width of the work area is reached. The items then wrap to form another row immediately below the filled row. This process is repeated until all the container items are positioned in rows. Default spacing is implemented according to the guidelines for the CUA user interface.

If the CCS_AUTOPOSITION style bit is set and the container is displaying the Icon view, container items are arranged automatically, without the CM_ARRANGE message being sent, when:

- The window size changes
- Container items are inserted, removed, sorted, invalidated, or filtered
- The font or font size changes.

In all of these cases, container items are arranged the same as when the CM_ARRANGE message is sent. The CCS_AUTOPOSITION style bit is valid only when it is used with the Icon view.

If the CM_ARRANGE message is issued and the container control is not currently displaying the Icon view, the container items are still arranged logically. Nothing changes in the current view; the arrangement of the container items is not visible until the user switches to the Icon view. Figure 8-2 shows an example of the container after a CM_ARRANGE message was sent, or if the container was created with the CCS_AUTOPOSITION style bit set.

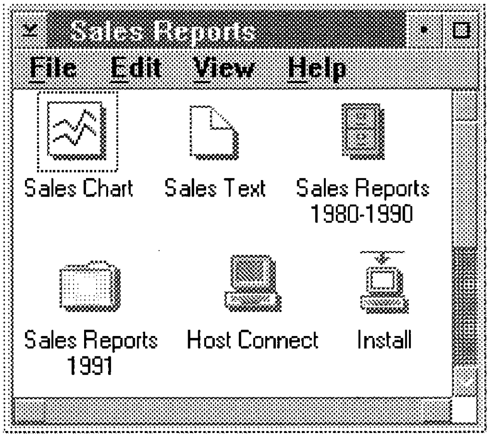

Figure 8-2. Icon View when Items Are Arranged or Automatically Positioned

Name View

The Name view (CV_NAME attribute) displays icon/text or bit-map/text pairs to represent container items. CV_NAME is an attribute of the CNRINFO data structure's *flWindowAttr* field.

In the Name view, icon/text pairs and bit-map/text pairs are icons and bit maps, respectively, with one or more lines of text displayed to the right of each icon or bit map. Each line can contain one or more text characters, which are left-justified. The container control does not limit the number of lines or the number of characters in each line.

The container control offers the option of flowing or not flowing the container items in the Name view. To *flow* container items means to dynamically arrange them in columns.

Non-Flowed Name View

If the container items are not flowed, the icon/text or bit-map/text pairs are placed in a single column in the leftmost portion of the work area, as shown in Figure 8-3 on page 8-6.

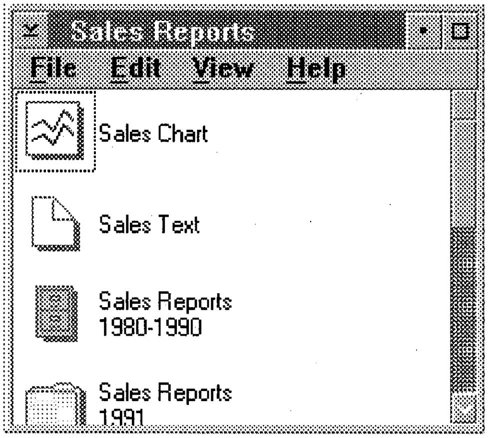

Figure 8-3. Non-Flowed Name View

Flowed Name View

If the container items are flowed (CV_NAME | CV_FLOW), the container appears as shown in Figure 8-4. In this case, the container items fill the leftmost column until the depth of the work area is reached. The items then wrap to form another column immediately to the right of the filled column. This process is repeated until all of the container items are positioned in columns.

The width of each column is determined by the widest text string within the column. The height of the work area is determined by the size of the window.

Figure 8-4. Flowed Name View

Text View

The Text view (CV_TEXT attribute) displays one or more lines of text to represent container items. CV_TEXT is an attribute of the CNRINFO data structure's *flWindowAttr* field.

Each line can contain one or more text characters, which are left-justified. The container control does not limit the number of lines or the number of characters in each line.

The container control offers the option of flowing or not flowing the container items in the Text view.

Non-Flowed Text View

If the text strings are not flowed, the text for each container item is placed in a single column in the leftmost portion of the work area, as shown in Figure 8-5.

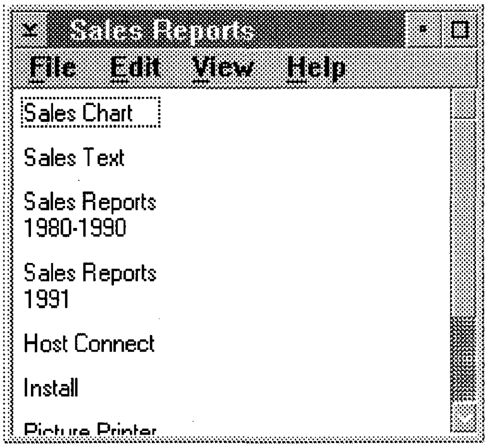

Figure 8-5. Non-Flowed Text View

Flowed Text View

If the text strings are flowed (CV_TEXT | CV_FLOW), the container appears as shown in Figure 8-6 on page 8-8. In this case, the text strings fill the leftmost column until the depth of the work area is reached. The text strings then wrap to form another column immediately to the right of the filled column. This process is repeated until all the text strings are positioned in columns.

The width of each column is determined by the widest text string within the column. The height of the work area is determined by the size of the window.

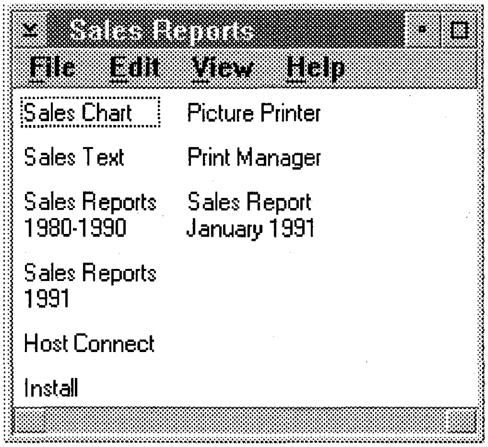

Figure 8-6. Flowed Text View

Tree View

The Tree view (CV_TREE attribute) displays container items arranged hierarchically. CV_TREE is an attribute of the CNRINFO data structure's *flWindowAttr* field.

The leftmost items displayed in the Tree view are at the *root level* and are the same items displayed in all the other container views. Items that contain other items are called *parent items*. The items that a parent item contains are called *child items* and can be displayed only in the Tree view. Child items that contain other items serve a dual role: they are the children of their parent item, but they are parent items as well, with children of their own. For example, a parent item might be a book that contains individual child items for its chapters or a folder that contains several reports. The chapters or reports, in turn, could be parent items that contain their own children, such as the major sections of a chapter or report.

If the child items of a parent item are not displayed, the parent item can be *Expanded* to display them as a new branch in the Tree view. Once a parent item has been expanded, it can be *Collapsed* to remove its child items from the display.

You can use the *cxTreeIndent* and *cxTreeLine* fields of the CNRINFO data structure to specify the number of pels that a new branch is to be indented horizontally, and the width of the lines that are used to connect branches of the tree. These lines are displayed only if the CA_TREELINE attribute is specified in the *flWindowAttr* field.

The Tree view has three different types: Tree icon view, Tree text view, and Tree name view. If CV_TREE is specified, the Tree icon view is the default view. If neither CV_ICON, CV_TEXT, or CV_NAME are specified, CV_ICON is assumed. Figure 8-7 on page 8-9 shows an example of the Tree icon view with root level, parent, and child items.

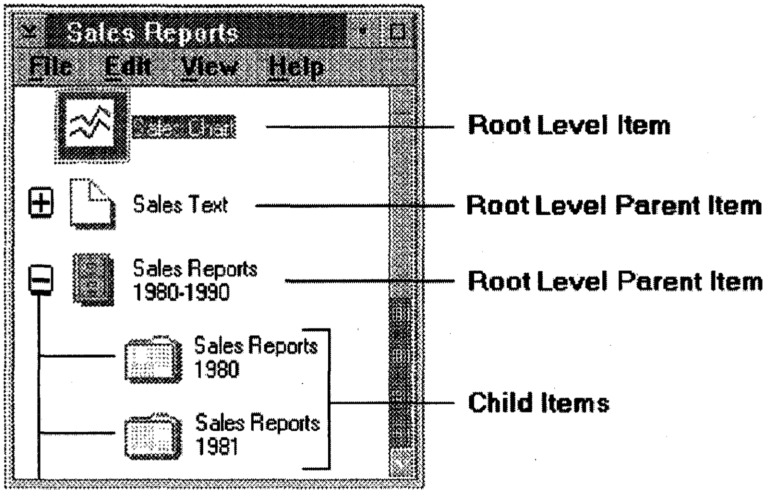

Figure 8-7. Tree View Showing Root Level, Parent, and Child Items

Tree Icon View and Tree Text View

The Tree icon and Tree text views are identical in every aspect except their appearance on the screen. Container items in the Tree icon view (CV_TREE | CV_ICON) are displayed as either icon/text pairs or bit-map/text pairs. The items are drawn as icons or bit maps with one or more lines of text displayed to the right of each icon or bit map. Figure 8-8 shows an example of the Tree icon view with the default Expanded and Collapsed bit maps.

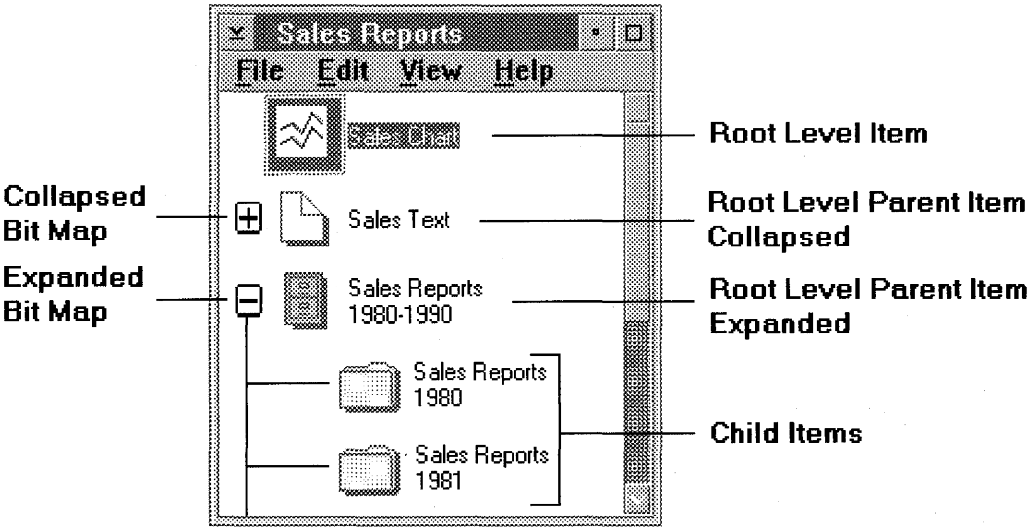

Figure 8-8. Tree Icon View

Container items in the Tree text view (CV_TREE | CV_TEXT) are displayed as text strings. In both views, the container control does not limit the number of lines of text or the number of characters in each line. Figure 8-9 provides an example of the Tree text view, again showing the default Expanded and Collapsed bit maps.

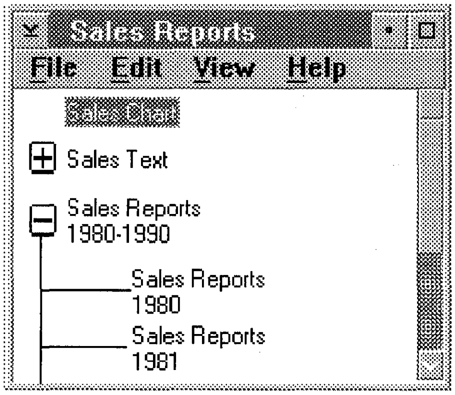

Figure 8-9. Tree Text View

In the Tree icon and Tree text views, a parent item is expanded by selecting the Collapsed icon/bit map, which is displayed to the left of the parent item.

The Collapsed icon/bit map should contain some visible indication that the item can be expanded. The default Collapsed bit map that is provided by the container control uses a plus sign (+) to indicate that more items, the children of this parent, can be added to the view.

When the child items of a parent item are displayed, the Collapsed icon/bit map to the left of that parent item changes to an Expanded icon/bit map. Just as the Collapsed icon/bit map provides a visible indication that an item can be expanded, so should the Expanded icon/bit map indicate that an item can be collapsed. The default Expanded bit map provided by the container control uses a minus sign (–) to indicate that the child items of this parent can be subtracted from the view. If any of the child items have children of their own, a Collapsed or Expanded icon/bit map is displayed to their immediate left as well.

To display your own Collapsed and Expanded icons or bit maps, specify their handles by using the *hptrCollapsed* and *hptrExpanded* fields of the CNRINFO data structure for icons, and the *hbmCollapsed* and *hbmExpanded* fields for bit maps. Also, you can use the *slTreeBitmapOrIcon* field to specify the size, in pels, of these Collapsed and Expanded icons and bit maps.

Tree Name View

Container items in the Tree name view (CV_TREE | CV_NAME) are displayed as either icon/text pairs or bit-map/text pairs. Similar to the Tree icon view, the items are drawn as icons or bit maps with one or more lines of text displayed to the right of each icon or bit map. The container control does not limit the number of lines or the number of characters in each line of text.

Unlike the Tree icon view, however, separate Collapsed and Expanded icons/bit maps are not used. Instead, if an item is a parent, the icon or bit map that represents that item contains the same type of visible indication that is placed in a separate icon/bit map in the Tree icon view to show that an item can be collapsed or expanded. In this way, the icon or bit map that represents the parent item can serve a dual purpose and thus preserve space on the screen, an important consideration if the text strings used to describe items become too long.

The container control does not provide default icons or bit maps for the Tree name view. To display your own Collapsed and Expanded icons or bit maps, specify their handles using the *hptrCollapsed* and *hptrExpanded* fields of the TREEITEMDESC data structure for icons, and the *hbmCollapsed* and *hbmExpanded* fields for bit maps. Also, you can use the *slBitmapOrIcon* field of the CNRINFO data structure to specify the size, in pels, of these Collapsed and Expanded icons and bit maps. Figure 8-10 shows an example of the Tree name view.

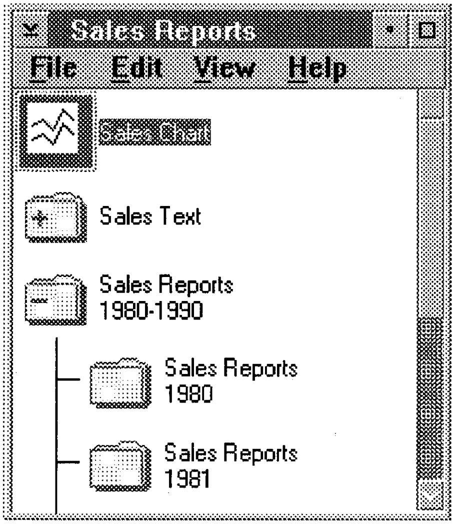

Figure 8-10. Tree Name View

Details View

The Details view (CV_DETAIL attribute) of the container control can display the following data types to represent container item: icons or bit maps, text, numbers, dates, and times. CV_DETAIL is an attribute of the CNRINFO data structure's *flWindowAttr* field.

The data is arranged in columns, which can have headings. Each column can contain data that belongs to only one of the valid data types. Column headings can contain text, icons, or bit maps.

The width of each column can be explicitly specified in the *cxWidth* field of the FIELDINFO data structure. If a column width is not specified, it is determined by the widest entry in the column.

Columns can be inserted or removed dynamically. All of the columns in a given row belong to a single container item; selecting the data portion of a row selects the entire row, not just the individual column.

Details view column headings and data can be top- or bottom-justified or vertically centered, as well as left- or right-justified or horizontally centered. In addition, horizontal separator lines can be specified between the column headings and the data; vertical separator lines can be placed between columns. Figure 8-11 shows an example of the Details view where Container Items, the icon, Description, and Item Size are the column headings.

Ownerdraw, where the application draws the container items, is supported for each column.

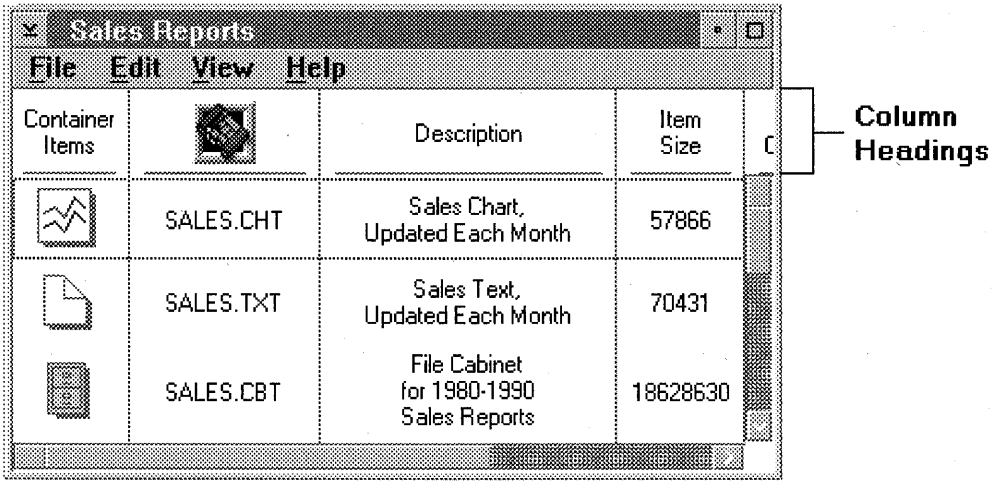

Figure 8-11. Details View

Determining the Width of a Column in a Details View

There are instances when you might want to determine the width of a column in the Details view. A function has been added to the container control to allow you to determine the width of the data in a column. You can then compute the width of the entire column by adding the width of the data to the left and right margins of the column. To determine the width of a column:

1. Define an attribute with a value of 0x0200 and give it a name such as CMA_DATAWIDTH.
2. Issue the CM_QUERYDETAILFIELDINFO message with the following values:
   a. Provide a pointer to the FIELDINFO data structure in *param1*.
   b. Specify your attribute (see step 1) in *param2*.

   c. Request a return value with a type of LONG, not PFIELDINFO, to retrieve the width of the column in the FIELDINFO data structure to which you are pointing. The value returned is the width of the data (text, icon, or bit map) in this column.

3. Use GpiQueryFontMetrics to query the average character width of the font used by the container. This value will be used to calculate the total column width.
4. Multiply 3 by the average character width and add this to the data width returned from step 2 on page 8-12 for all columns except the following:
   - The first and last columns in each split window. In these cases, multiply 2.5 by the average character width and add the column data width returned from step 2 on page 8-12.
   - The only other special case is where there is only 1 column in either the left or right split windows. In this case, you would multiply 2 by the average character width and add the column data width returned from step 2 on page 8-12.
5. The value returned is the total width of the column.

Split Bar Support for the Details View

A split bar enables the application to split the container window vertically between two column boundaries. This function is available only in the Details view.

The two portions of the work area on either side of the split bar appear side-by-side. They scroll in unison vertically, but they scroll independently horizontally.

The application is responsible for specifying the position of the split bar, which is defined with the *xVertSplitbar* field. Also, the rightmost column of the left split window is specified with the *pFieldInfoLast* field. *xVertSplitbar* and *pFieldInfoLast* are fields of the CNRINFO data structure.

The left split window cannot be empty if there is data in the right window. The right split window is not required to have data. However, because data cannot be scrolled from the right split window into the left split window, or from left to right, the split bar loses much of its usefulness if the right split window is empty.

The user can drag the vertical split bar within the limits of the window. As the user drags the split bar to the left, the right split window becomes wider; as the user drags the split bar to the right, the left split window becomes wider.

Each container control can have one vertical split bar. Horizontal split bars are not supported.

Figure 8-12 on page 8-14 shows an example of a split bar between the Description column and the Date Created column.

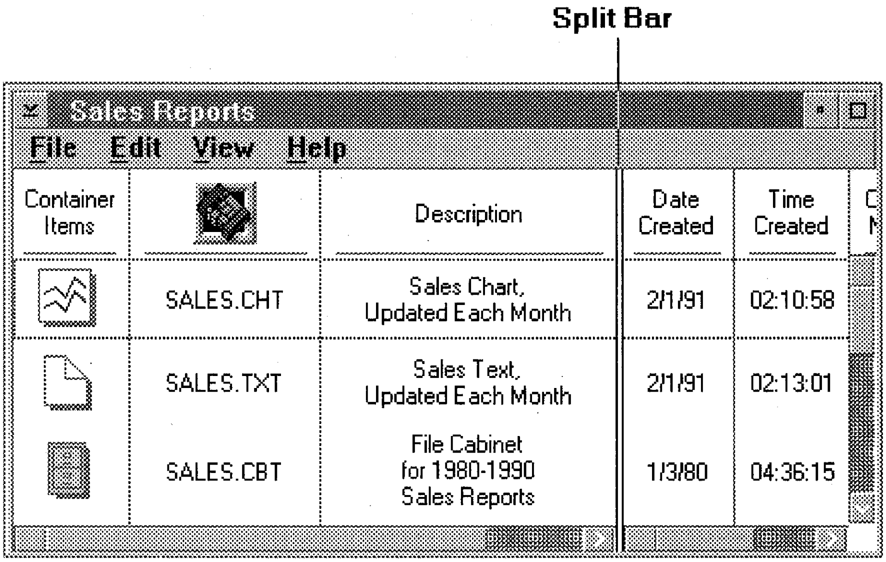

Figure 8-12. Details View with Split Bar

Using Container Controls

This section provides information about the following topics:

- Creating a container
- Allocating memory for container records
- Allocating memory for container columns
- Inserting container records
- Removing container records
- Setting the container control focus
- Using container views
- Changing a container view.

Note: Much of the sample code in this section is part of a complete program that creates a container for a small address book. The program is illustrated in "Sample Code for Container Controls" on page 8-38.

Creating a Container

You create a container by using the WC_CONTAINER window class name in the *ClassName* parameter of WinCreateWindow. Before you create the container, you can create a frame window as a parent. If you create the frame window, it sizes the container to fill its work area. The sample code illustrated in Figure 8-13 on page 8-15 shows the code to create both the frame and the container.

```
HAB      hab;
HWND     hPopupMenu;
HWND     hFrameWnd, hCnrWnd;   /* Frame and Container window handles */
PFNWP    SysWndProc;

INT main (VOID)
{
    HMQ        hmq;
    FRAMECDATA  fcd;
    QMSG       qmsg;

    if (!(hab = WinInitialize(0)))
      return FALSE;

    if (!(hmq = WinCreateMsgQueue(hab, 0)))
      return FALSE;

/**********************************************************************/
/*  Set up the frame control data for the frame window.               */
/**********************************************************************/
    fcd.cb = sizeof(FRAMECDATA);
    fcd.flCreateFlags = FCF_TITLEBAR      |
                        FCF_SYSMENU       |
                        FCF_SIZEBORDER    |
                        FCF_SHELLPOSITION |
                        FCF_MINMAX        |
                        FCF_TASKLIST;
    fcd.hmodResources = NULLHANDLE;
    fcd.idResources = 0;

/**********************************************************************/
/*  Create the frame to hold the container control.                   */
/**********************************************************************/
    hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                                WC_FRAME,
                                "Phone Book",
                                0, 0, 0, 0, 0,
                                NULLHANDLE,
                                HWND_TOP,
                                0,
                                &fcd,
                                NULL);
```
Figure 8-13 (Part 1 of 2). Sample Code for Creating a Container

```
/**********************************************************************/
/*  Verify that the frame was created; otherwise, stop.               */
/**********************************************************************/
    if (!hFrameWnd)
      return FALSE;

/**********************************************************************/
/*  Set an icon for the frame window.                                 */
/**********************************************************************/
    WinSendMsg(hFrameWnd,
               WM_SETICON,
               (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                          SPTR_FOLDER,
                                          FALSE),
               NULL);

/**********************************************************************/
/*  Create the container.                                             */
/**********************************************************************/
    hCnrWnd = WinCreateWindow(hFrameWnd,
                              WC_CONTAINER,
                              NULL,
                              CCS_AUTOPOSITION |
                              CCS_READONLY     |
                              CCS_SINGLESEL,
                              0, 0, 0, 0,
                              hFrameWnd,
                              HWND_BOTTOM,
                              FID_CLIENT,
                              NULL,
                              NULL);
```

Figure 8-13 (Part 2 of 2). Sample Code for Creating a Container

The container is created with a default set of control data, which can be changed using the CM_SETCNRINFO message.

Allocating Memory for Container Records

Your application must allocate memory for a container record by using the CM_ALLOCRECORD message, which also enables you to allocate memory for additional application data.

The maximum number of records is limited by the amount of memory in the user's computer. The container control does not limit the number of records that a container can have.

The sample code illustrated in Figure 8-14 on page 8-17 shows how to allocate memory for records that populate the container. A pointer to the record is returned.

```
    HWND             hIcon;
    PRECORDCORE      Address, FirstRec;
    RECORDINSERT     recsIn;
    ULONG            x;

/***********************************************************************/
/*  Allocate MAXFRIENDS records all at once -                          */
/*  CM_ALLOCRECORD returns them in a linked list.                      */
/***********************************************************************/
    Address = (PRECORDCORE)WinSendMsg(hWnd,
                                      CM_ALLOCRECORD,
                                      0,
                                      MPFROMLONG(MAXFRIENDS));
```

Figure 8-14. Sample Code for Allocating Memory for Container Records

Your application can use the CM_ALLOCRECORD message to allocate memory for one or more container records. The application can request *n* container records with an ***nRecords*** parameter. If ***n*** is one, a pointer to that record is returned. If ***n*** is greater than one, a pointer to the first record in a linked list of ***n*** records is returned.

Allocating Memory for Container Columns

In addition to allocating memory for records, an application also must allocate memory for columns of data if the details view is used. In the Details view, a container's data is displayed in columns, each of which is described in a FIELDINFO data structure.

Memory is allocated for FIELDINFO data structures using the CM_ALLOCDETAILFIELDINFO message. Unlike the CM_ALLOCRECORD message, the CM_ALLOCDETAILFIELDINFO message does not allow the application to allocate memory for additional application data. However, the *pUserData* field of the FIELDINFO data structure can be used to store a pointer to the application-allocated data.

Multiple FIELDINFO data structures can be allocated with the *nFieldInfo* parameter of the CM_ALLOCDETAILFIELDINFO message.

Inserting Container Records

After the memory is allocated, you can insert one or more container records by using the CM_INSERTRECORD message.

Figure 8-15 on page 8-18 provides a sample code that inserts records into a container for which memory was allocated in Figure 8-14.

```
/*********************************************************************/
/*  We will need the first record's address to                       */
/*  insert them into the container.                                  */
/*********************************************************************/
    FirstRec = Address;

/*********************************************************************/
/*  Loop through the address book, loading as we go.                 */
/*  Because the CM_ALLOCRECORD returns a linked                      */
/*  list, the address of the next record is retrieved                */
/*  from each record as we go (preccNextRecord).                     */
/*********************************************************************/
    for (x = 0; x < MAXFRIENDS; x++)
    {
       Address->cb = sizeof(RECORDCORE);    /* Standard records      */
       Address->hptrIcon = hIcon;           /* File icon             */
       Address->pszIcon = Friends[x].NickName;
       Address->pszName = Friends[x].FullName;
       Address->pszText = Friends[x].FullName;
       Address = Address->preccNextRecord;  /* Next record in list   */
    }

/*********************************************************************/
/*  Set up the insert record structure to place the                  */
/*  records in the container.                                        */
/*********************************************************************/
    recsIn.cb = sizeof(RECORDINSERT);

    /* Put the records in after any others */
    recsIn.pRecordOrder = (PRECORDCORE)CMA_END;

    /* All the records are top level (not children of other records) */
    recsIn.pRecordParent = NULL;

    /* The icons are top level */
    recsIn.zOrder = (USHORT)CMA_TOP;
```

Figure 8-15 (Part 1 of 2). Sample Code for Inserting Records into a Container

```
    /* Redraw the container */
    recsIn.fInvalidateRecord = TRUE;

    /* Set the number of records to insert */
    recsIn.cRecordsInsert = MAXFRIENDS;

/**********************************************************************/
/*  Insert the records into the container.                            */
/**********************************************************************/
    WinSendMsg(hWnd,
               CM_INSERTRECORD,
               (PRECORDCORE)FirstRec,
               &recsIn);
}
```

Figure 8-15 (Part 2 of 2). Sample Code for Inserting Records into a Container

The CM_INSERTRECORD message requires you to provide two pointers. The first pointer points to the record that is to be inserted, which is specified in the *FirstRec* parameter. When you are inserting multiple records, use this parameter to specify a pointer to a linked list of records.

The second pointer points to a RECORDINSERT data structure (*&recsIn*), which specifies information the container needs for inserting records.

One of the elements of information that this data structure contains is the order in which the records are to be inserted, which is specified in the *pRecordOrder* field. In this field you have two options. The first option is to specify a pointer to a container record. The records being inserted are placed immediately after that record. In this case, the *pRecordParent* field is ignored.

The second option is to specify whether the records being inserted are to be placed at the beginning or end of a list of records. This is done by specifying either the CMA_FIRST or CMA_END attributes. If you choose this option, the list of records used depends on the value of the *pRecordParent* field.

If CMA_FIRST or CMA_END is specified and the value of the *pRecordParent* field is NULL, the inserted records are placed at the beginning or end of the root-level records. However, if CMA_FIRST or CMA_END is specified and *pRecordParent* contains a pointer to a parent item record, the records are inserted at the beginning or end of the list of child item records that this parent record contains.

The RECORDINSERT data structure also lets you specify the z-order position of the records being inserted. The CMA_TOP and CMA_BOTTOM attributes of the *zOrder* field place the record at the top or bottom, relative to the other records in the z-order list. This field applies to the Icon view only.

To specify the number of records that are being inserted, use the *cRecordsInsert* field. The value of this field must be greater than 0.

The last field in the RECORDINSERT data structure is *flInvalidateRecord*, which enables you to control whether the records are displayed automatically when they are inserted. If you specify TRUE in this field, the display is updated automatically. However, if you specify FALSE, the application must send the CM_INVALIDATERECORD message after the records are inserted to update the display.

Where items are positioned in a container depends on the view the user has specified. If the Icon view is specified and the CCS_AUTOPOSITION style bit is not set, the *x*- and *y*-coordinates for each record, which are stored in the *ptlIcon* field of the RECORDCORE and MINIRECORDCORE data structures, determine its position. Records displayed in the Name view, Text view, Tree view, and Details view are positioned as previously described in this section.

Note: Records inserted into a list of child record items can be displayed in the Tree view only. These records are visible only if the parent record item to which these child record items belong is expanded.

Removing Container Records

The CM_REMOVERECORD message can be used to remove one or more container records from the container control. The sample code in Figure 8-16 removes all records from a container and frees the memory associated with those records. It is the application's responsibility to free all application-allocated memory that is associated with the removed container records. The container is invalidated and repainted.

```
USHORT cNumRecord;           /* Number of records to be removed     */
USHORT fRemoveRecord;        /* Container message attributes        */

/********************************************************************/
/*  Zero means remove all records.                                  */
/********************************************************************/
cNumRecord = 0;

/********************************************************************/
/*  Specify attributes to invalidate the container                  */
/*  and free the memory.                                            */
/********************************************************************/
fRemoveRecord =
  CMA_INVALIDATERECORD | CMA_FREE;
```

Figure 8-16 (Part 1 of 2). Sample Code for Removing Container Records

```
/*************************************************************************/
/*  Remove the records.                                                  */
/*************************************************************************/
WinSendMsg(hwndCnr,             /* Container window handle           */
  CM_REMOVERECORD,              /* Container message for removing    */
                                /* records                           */
  NULL,                         /* NULL PRECORDARRAY                 */
  MPFROM2SHORT(
    cNumRecord,                 /* Number of records                 */
    fRemoveRecord));            /* Memory invalidation flags         */
```

Figure 8-16 (Part 2 of 2). Sample Code for Removing Container Records

The application must set the pointers to each record to be removed in an array. If the *fRemoveRecord* parameter of this message includes the CMA_FREE attribute, the records are removed and the memory is freed. If this attribute is not set, the records are removed from the list of items in the container, and the application must use the CM_FREERECORD message to free the memory. The default is not to free the memory.

If the *fRemoveRecord* parameter includes the CMA_INVALIDATERECORD attribute, the container is invalidated after the records are removed. The default is not to invalidate the container. The CMA_INVALIDATERECORD attribute can be used with the CMA_FREE attribute, separated by a logical OR operator (|), to free the record's memory and invalidate the container.

Setting the Container Control Focus

The application must set the focus to the container control using WinSetFocus so that all mouse and keyboard activity goes to the container window. The sample code illustrated in Figure 8-17 shows how to use WinSetFocus.

```
WinSetFocus(HWND_DESKTOP,       /* Desktop window handle            */
            hListWnd)           /* Handle of window to receive focus */
```

Figure 8-17. Sample Code Showing How to Use WinSetFocus

Using Container Views

Container views are specified by using attributes on the *flWindowAttr* field of the CNRINFO data structure.

Because the container control does not support both icons and bit maps in the same view, an application must specify which are used by setting either the CA_DRAWICON attribute or the CA_DRAWBITMAP attribute in the *flWindowAttr* field of the CNRINFO data structure. The default is the CA_DRAWICON attribute. The size of the icon or bit map can be specified in the *slBitmapOrIcon* field of the CNRINFO data structure.

Changing a Container View

The sample code illustrated in Figure 8-18 shows how to use the CM_SETCNRINFO message to change from the current view of a container (Name, Details, or Text) to the Icon view.

```
CNRINFO cnrInfo;

/**********************************************************************/
/*  Set the attribute field to the Icon view.                         */
/**********************************************************************/
cnrInfo.flWindowAttr = CV_ICON;

/**********************************************************************/
/*  Change the view from the current view to the Icon view.           */
/**********************************************************************/
WinSendMsg(
  hwndCnr,                      /* Container window handle            */
  CM_SETCNRINFO,                /* Container message for setting      */
  MPFROMP(&cnrInfo),            /* Container control data             */
  MPFROMLONG(
    CMA_FLWINDOWATTR));         /* Message attribute that sets        */
                                /* container window attributes        */
```

Figure 8-18. Sample Code for Changing a Container View

Graphical User Interface Support for Container Controls

This section describes the container control support for graphical user interfaces (GUIs). Except where noted, this support conforms to CUA interface design guidelines. The GUI support provided by the container control consists of the following:

- Scrolling
- Dynamic scrolling
- Selecting container items
- Providing emphasis
- Using direct manipulation.

Scrolling

The container control automatically provides horizontal or vertical scroll bars, or both, whenever the container window's work area is not large enough to display all of the container items.

If all container items are visible in the work area, the scroll bars are either removed or disabled, depending on the view and how the items are positioned, as follows:

- If container items are displayed in the icon or tree view, and one or more items are not visible in the work area, a horizontal scroll bar, vertical scroll bar, or both, are provided, depending on the position of the items outside of the work area. If container items are positioned to the right or left of the work area, a horizontal scroll bar is provided; if container items are positioned below or above the work area, a vertical scroll bar is provided.

  Scroll bars are not provided if all the container items are visible in the work area. Scroll bars are removed from the container window if either of the following occurs:

  - Container items positioned outside the work area are moved into the work area.
  - The size of the container window is increased so that container items formerly not visible become visible.

- If container items are displayed in non-flowed text and non-flowed Name views, a vertical scroll bar is provided; this scroll bar is disabled if all the container items are visible in the work area. A horizontal scroll bar is used in these views only when the work area is too narrow to allow the widest container item to be seen in its entirety. If the user changes the window size to allow the entire widest container item to be seen, the horizontal scroll bar is removed.

- If container items are displayed in flowed text and flowed name views, a horizontal scroll bar is provided; this scroll bar is disabled if all the container items are visible in the work area. A vertical scroll bar is used in these views only when the work area is too short to allow the tallest container item to be seen in its entirety. If the user changes the window size to allow the entire tallest container item to be seen, the vertical scroll bar is removed.

- If container items are displayed in the Details view, both horizontal and vertical scroll bars are provided. These scroll bars are disabled if all the container items are visible in the work area.

  Note: A Details view that is split has two horizontal scroll bars, one for each portion of the split window.

Dynamic Scrolling

The container control supports *dynamic scrolling*, which enables the user to drag the scroll box in the scroll bar and get immediate visible feedback on where the scrolling stops when the scroll box is dropped. If the scrolling range is greater than 32K pels, dynamic scrolling is disabled.

Selecting Container Items

Except during direct manipulation and direct editing of text in a container, a user must select a container item before performing an action on it. The container control provides several selection types, along with selection techniques to implement those types. The container control also supports two selection mechanisms: pointing device, such as a mouse, and the keyboard.

Selection Types

The container control supports the following selection types:

- Single selection

  Single selection enables a user to select only one container item at a time. This is the default selection type for all views and is the only selection type supported for the Tree view.

- Extended selection

  Extended selection enables a user to select one or more container items, in any combination. The CUA-defined keyboard augmentation keys are implemented for extended selection. When used with a pointing device, these keys enable a user to select discontiguous sets of container items. Extended selection is valid for all views except the Tree view.

- Multiple selection

  Multiple selection enables a user to select none, some, or all of the container items. Multiple selection is valid for all views except the Tree view.

Only one of these selection types can be used for each container. The selection type for a container is defined when the container is created.

Selection Techniques

Depending on the type of view and the type of selection, a user can select container items using the following selection techniques:

- Marquee selection

  Marquee selection is supported only in the Icon view and is only valid with the extended and multiple selection types. This selection technique enables a user to begin selection from an anchor point that is established by moving the pointer to white space in the container and pressing, but not releasing, the select button on the pointing device. As the user presses the select button and drags the pointer, a tracking rectangle is drawn between the anchor point and the current pointer position. All items whose icons or bit maps are entirely within the tracking rectangle are dynamically selected.

- Swipe selection

  Swipe selection is valid only with the extended and multiple selection types. The container control implements two techniques for swipe selection: touch swipe and range swipe.

  - Touch swipe

    Touch-swipe selection is implemented in the Icon view. With this selection technique, the pointer must pass over some portion of a container item while the user is pressing the select button for that item to be selected.

  - Range swipe

    In views other than the Icon and Tree views, range-swipe selection is available. With this method, the user presses the select button while moving the pointer. However, the pointer does not have to pass directly over a container item for that item to be selected. Aside from pressing the select button and moving the pointer, the only other requirement for selection is that the container item must be within a range of items that is being selected. The range begins at the pointer's position when the user presses the select button; it ends at the pointer's position when the user releases the select button.

- First-letter selection

  For the Icon, Name, Text, and Tree views, first letter selection occurs when a character key is pressed, and the first container item whose text begins with that character is displayed with selected-state emphasis. The same is true for the Details view, except that all the columns for a record are searched for a matching character before the next record is searched. The effect of first letter selection on other selected container items depends on the chosen selection type (single, multiple, or extended).

Note: If more than one container window is open, selecting a container item in one window has no effect on the selections in any other window.

Selection Mechanisms

Mouse button 1 (the select button) is used for selecting container items, and mouse button 2 (the drag button) is used for dragging and dropping container items during direct manipulation. These definitions also apply to the same buttons on any other pointing device.

In addition, a user can press a keyboard key while pressing a mouse button; this is called *keyboard augmentation*. The only instance of keyboard augmentation defined specifically for the container control is pressing the Alt key with the select button, which starts direct editing of text in a container.

In addition, the container control supports two keyboard cursors that can be moved by using keyboard navigation keys:

- The *selection cursor*, a dotted black box drawn around a container item, which represents the current position for the purpose of keyboard navigation.
- The *text cursor*, a vertical line that shows the user where text can be inserted or deleted when container text is being edited directly.

Keyboard navigation consists of the use of the Up and Down Arrow, Left and Right Arrow, Home, End, PgUp, and PgDn keys. If container items are not visible within the work area, navigation with these keys causes the items to scroll into view if the user is not editing container text directly.

Providing Emphasis

The container control supports various types of emphasis. The following list describes forms of emphasis that have a distinct visible representation in the container control:

- Selected-state and unavailable-state emphasis

  When a container item is selected, the container item receives *selected-state emphasis*, which means that the emphasis is applied to icon/text or bit-map/text pairs in the Icon, Name, Tree icon, and Tree name views; text strings in the Text and Tree text views; and an entire row that represents a container item in the Details view. Figure 8-19 illustrates an example of selected-state and unavailable-state emphasis; the emphasis on the choice in the pull-down menu indicates that the choice is unavailable.

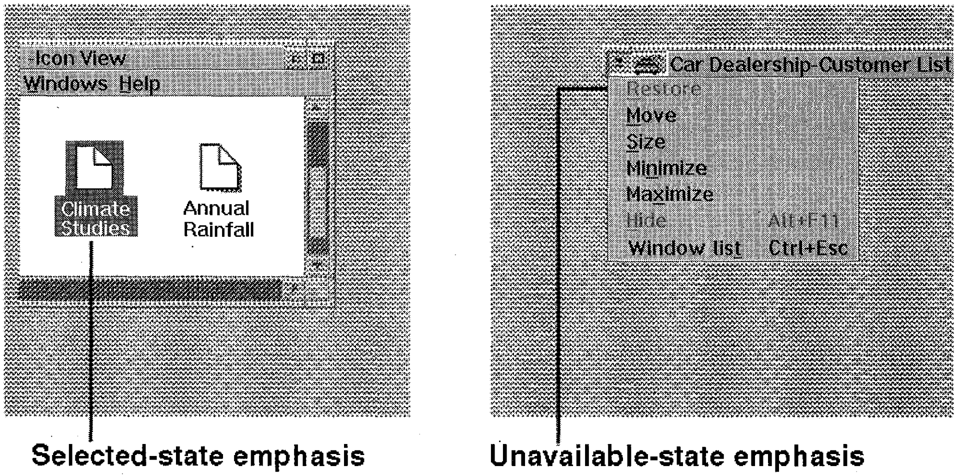

Figure 8-19. Selected-State and Unavailable-State Emphasis

  The color for selected-state emphasis can be changed by using the control panel or WinSetPresParam, which results in a WM_PRESPARAMCHANGED message being sent to the container.

- In-use emphasis

  Cross-hatching behind an icon or bit map indicates *in-use emphasis*. In-use emphasis is not applied to container items in the Text view, Tree text view, or Details view when it contains text only. However, the Details view often includes icons or bit maps in one column of each record, usually the leftmost column. In this situation, specify the column that contains the icons or bit maps so that in-use emphasis can be applied to them. This column can be set by using the *pFieldInfoObject* field of the CNRINFO data structure.

- Target emphasis

  Target emphasis is used during direct manipulation. When a user drags one container item over another, the item beneath the dragged item displays *target emphasis*. Two forms of target emphasis (visible feedback) are available: a black line and a black border. These forms of emphasis indicate the *target*, where the container item is dropped if the user releases the drag button.

Using Direct Manipulation

Direct manipulation is a protocol that enables the user to drag a container item within its current window or from one window to another. The user can drop the container item either on white space in a window or on another item.

Direct manipulation can be performed with all views of the container control. A function is provided so that the application is notified if an item is dropped on another item in the container and if an item is dragged from the container.

The user can drag any container item, whether or not it is selected. If the user presses the drag button when the pointer is over a selected container item, the application drags all selected items.

If the user presses the drag button when the pointer is over a container item that is not selected, the application drags only the item that the pointer is over.

The container control fully supports direct manipulation.

Enhancing Container Controls Performance and Effectiveness

The following topics offer information about fine-tuning a container to enhance its performance and effectiveness:

- Positioning container items
- Specifying space between container items
- Providing target emphasis
- Specifying deltas for large amounts of data
- Direct editing of text in a container
- Specifying container titles
- Specifying fonts and colors
- Drawing container items and painting backgrounds
- Filtering container items
- Optimizing container memory usage
- Sharing records among multiple containers.

Positioning Container Items

Container items are positioned in the Icon view according to workspace coordinates.

The *workspace* is a two-dimensional Cartesian-coordinate system. The user can see a portion of the workspace in the work area, which is the scrollable viewing area of the container that is defined by the size of the container window. The work area is logically scrollable within the workspace.

Figure 8-20 on page 8-28 shows the *x*- and *y*-axes of the workspace with a container window and its work area superimposed. (This figure is not drawn to scale.)

Scrollable Workspace Areas

The workspace is indicated by the solid black line that runs even with:

- The top and bottom edges of the topmost and bottommost container items
- The left and right edges of the leftmost and rightmost container items.

The workspace is defined by the minimum and maximum *x*- and *y*-coordinates of the items in the container. The work area of the container window can be scrolled only within the workspace and only as far as is necessary to see the topmost, bottommost, leftmost, and rightmost container items.

Figure 8-20 shows the scrollable work area of the workspace.

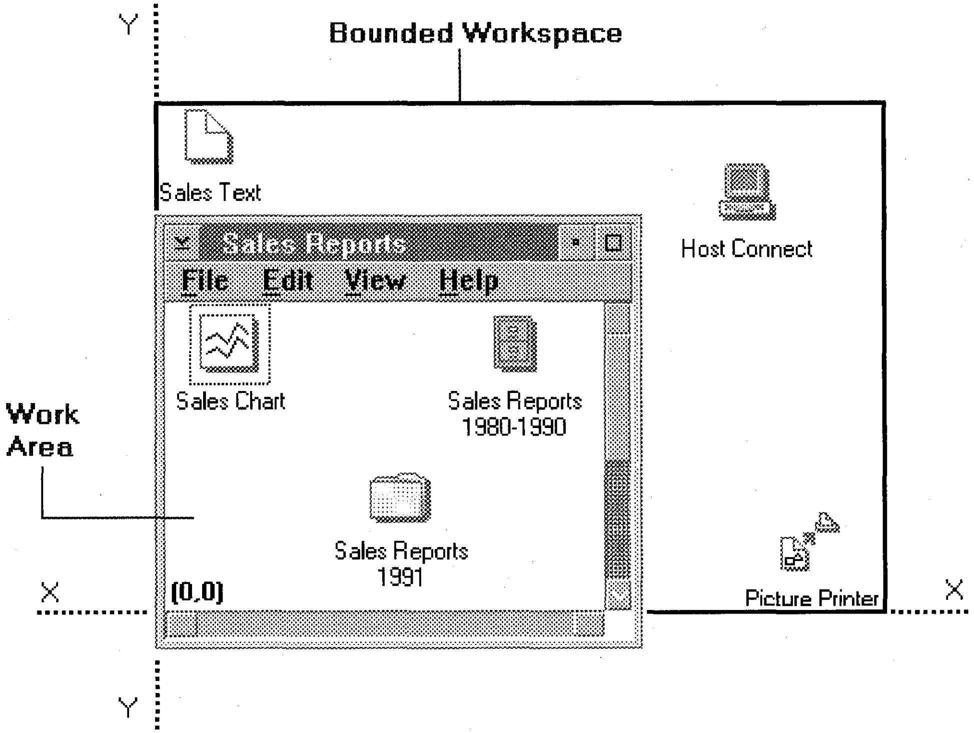

Figure 8-20. Workspace X- and Y-Axes

Figure 8-21 on page 8-29 further illustrates a bounded workspace. In this example, the topmost and bottommost container items limit the workspace. The work area has been scrolled so that the elements are not all within the work area. The work area could be scrolled to the left so that it would include the leftmost element, or scrolled down and to the right to include the rightmost element, but it could not be scrolled any farther in either direction.

Workspace and Work Area Origins

When the container is created, the work area and workspace share the same origin, (0,0), as represented in Figure 8-20 on page 8-28. If the application requires that the work area and the workspace have different origins, the application can use the *ptlOrigin* field of the CNRINFO data structure and the CM_SETCNRINFO message to set the origin of the work area. The application can use the CM_QUERYCNRINFO and CM_SETCNRINFO messages to obtain the origin when the user ends the application and to reset the origins when the user restarts the application.

Container items are located in reference to the workspace origin. There is a visual shift as the work area is scrolled; however, because the work area moves over a fixed workspace, the coordinates of the container items do not change.

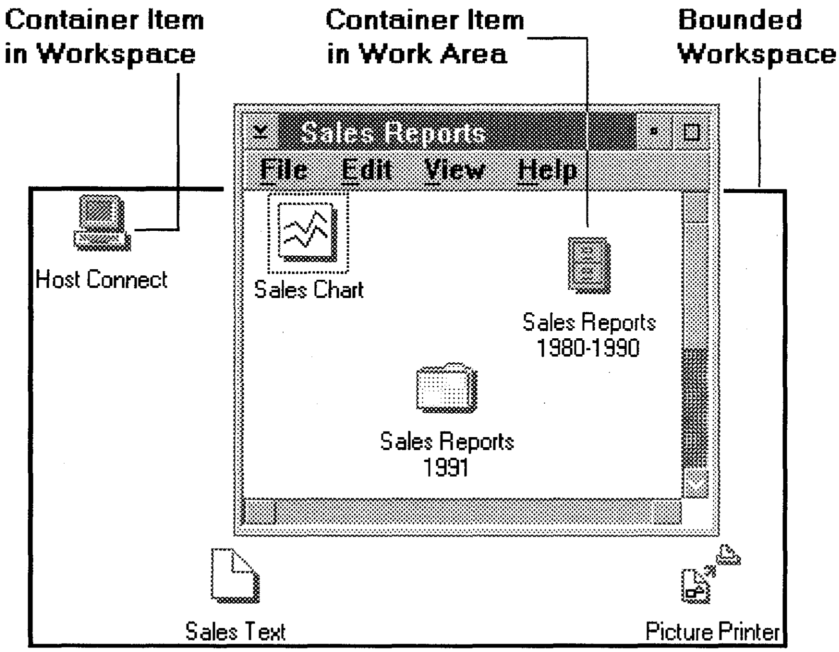

Figure 8-21. Workspace Bounds

Specifying Space between Container Items

You can specify the amount of vertical space, in pels, to allow between container items by using the *cyLineSpacing* field of the CNRINFO data structure. If you do not specify how much vertical space can be used, the container control sets the space between the items using a default value. For the Tree view, you can specify the horizontal distance between the levels by using the *cxTreeIndent* field of the CNRINFO data structure. If this value is less than 0, a default is used.

Providing Source Emphasis

Source emphasis is the type of emphasis provided when a context menu is displayed. It appears as a dotted box with rounded corners that surrounds the item for which the context menu is requested, or the item that is being dragged.

To provide source emphasis for container items issue the CM_SETRECORDEMPHASIS message with the following values:

1. Provide a pointer to the RECORDCORE or MINIRECORDCORE data structure in *param1*.

   You can provide source emphasis for the entire container by setting *param1* to NULL.

2. Set the *usChangeEmphasis* parameter to TRUE in *param2*.
3. Set the *fEmphasisAttribute* parameter in *param2* to CRA_SOURCE (0x00004000L).

To remove source emphasis follow the same procedure outlined above, but set the *usChangeEmphasis* parameter in *param2* to FALSE instead of TRUE.

Providing Target Emphasis

The CA_ORDEREDTARGETEMPH and CA_MIXEDTARGETEMPH attributes of the CNRINFO data structure's *flWindowAttr* field determine the form of emphasis applied for the Text, Name, and Details views, as follows:

- If the CA_ORDEREDTARGETEMPH attribute is set:
  - The CN_DRAGAFTER notification code is sent when a container item is being dragged.
  - A black line is drawn between container items to show the current target position.
- If the CA_MIXEDTARGETEMPH attribute is set:
  - The CN_DRAGAFTER and CN_DRAGOVER notification codes are sent when a container item is being dragged. The notification code sent depends on the position of the pointer relative to the item it is positioned over.
  - A black line is drawn if the pointer is positioned such that the item being dragged is inserted between two target items.
  - A black border is drawn around either the entire target item for the Text and Details views or the icon or bit map for the Name view if the pointer is positioned such that the item being dragged is dropped on the target item.
- If the CA_ORDEREDTARGETEMPH and CA_MIXEDTARGETEMPH attributes are not set:
  - The CN_DRAGOVER notification code is sent when a container item is being dragged.
  - A black border is drawn around the entire target item for the Text and Details views, and around the icon or bit map only for the Name view.

For the Icon and Tree view, the CA_ORDEREDTARGETEMPH and CA_MIXEDTARGETEMPH attributes are ignored, so target emphasis is applied as follows:

- The CN_DRAGOVER notification code is sent when a container item is dragged.
- A black border is drawn around the target, as follows:
  - For the Icon view, if the target is another container item, a black border is drawn around the icon or bit map that represents the container item, but not around the text string beneath it. If the target is white space, a black border is drawn around the outer edge of the entire work area.
  - For the Tree icon and Tree name views, a black border is drawn around the icon or bit map that represents the container item, but not around the text string to the right of it.
  - For the Tree text view, a black border is drawn around the entire target item.

Specifying Deltas for Large Amounts of Data

The container control can accommodate large amounts of data with an application-defined delta. The *delta* is an application-defined threshold, or number of container items, from either end of the list. The application is responsible for specifying the delta value in the CNRINFO data structure's *cDelta* field. It also is responsible for setting the delta value with the CMA_DELTA attribute of the CM_SETCNRINFO message's *ulCnrInfoFl* parameter.

The container control monitors its place in the list of container items when the user is scrolling through it. When the user scrolls to the delta from either end of the list, the container control sends a CN_QUERYDELTA notification code to the application as a request for more container items in the list.

The application is responsible for managing the records in the container. When the application receives the CN_QUERYDELTA notification code, the application is responsible for removing and inserting container records by using the CM_REMOVERECORD message and the CM_INSERTRECORD message, respectively.

Notes:

1. The delta concept is intended for applications with large amounts of data, or several thousand records. Applications with smaller amounts of data are not required to use the delta function. The default delta value is 0.
2. The delta function is not available in the Icon view because it is intended for data displayed in a linear format.

Direct Editing of Text in a Container

Direct editing of text is supported for any text field in a container, including the container title, column headings, and container items. If a text field, such as the text field beneath an icon in the Icon view, has no text and is not read-only, a user can place text in that field by editing the field directly. The font specified for the container by the application is used for the edited text.

Direct editing is supported only for text data. Therefore, if the data type in the Details view is other than CFA_STRING, a user cannot edit it. CFA_STRING is an attribute of the FIELDINFO data structure's *flData* field.

You can prevent a user from editing any of the text in a container window by setting the CCS_READONLY style bit when a container is created. If you do not set this style bit, the user can edit any of the text in a container window unless you set the following read-only attributes:

- CA_TITLEREADONLY of the CNRINFO data structure's *flWindowAttr* field
- CRA_RECORDREADONLY of the RECORDCORE data structure's *flRecordAttr* field
- CFA_FIREADONLY of the FIELDINFO data structure's *flData* field
- CFA_FITITLEREADONLY of the FIELDINFO data structure's *flTitle* field.

If one of these read-only attributes is set, a user's attempts to edit container text directly are ignored.

A user can edit container text directly by doing either of the following:

- Moving the pointer to an editable text field, holding down the Alt key, and clicking the select button.
- Sending a CM_OPENEDIT message to the container control. The application can assign a key or menu choice to this message so that the keyboard can be used to edit container text directly.

The container control responds by using the WM_CONTROL message to send the CN_BEGINEDIT notification code to the application. A window that contains a multiple-line entry (MLE) field opens to show that container text can be edited directly.

The editing actions supported by MLEs, such as **Cut**, **Copy**, and **Paste**, are also supported by the container control. These actions can be performed using system-defined *shortcut* keys. The actions and shortcut keys are defined by CUA interface design guidelines.

If the user enters a text string that is longer than the text field, the text string scrolls. If multiple lines of text are needed or wanted, a user can press the Enter key to insert a new line.

A user can end the direct editing of container text and save the changes by doing either of the following:

- Moving the pointer outside the MLE and pressing the select button.
- Sending a CM_CLOSEEDIT message to the container control. The application can assign a key or menu choice to this message so that the keyboard can be used to end the direct editing of container text.

The container responds by sending the WM_CONTROL message to the application again, but this time with the CN_REALLOCPSZ notification code. The application can allocate more memory on receipt of the CN_REALLOCPSZ notification code, if necessary. If the application returns TRUE, the container control copies the new text to the application's text string. If the application returns FALSE, the text change in the MLE is disregarded. The container then sends the WM_CONTROL message to the application again, this time with the CN_ENDEDIT notification code. The MLE field is removed from the screen, leaving only the text string.

A user can end the direct editing of container text without saving any changes to the text in numerous ways, including the following:

- Pressing the Esc key
- Dragging the container item that is being edited
- Pressing the Alt key and the select button before the direct editing of container text has ended
- Scrolling the container window.

The CN_ENDEDIT notification code is sent to the application in each of these cases.

Searching for Exact Text String Matches

There might be times when you need to search the container for a text string that is an exact match of your search string argument. To find an exact match:

- In the SEARCHSTRING data structure, specify values for the fields as you normally would, with the following exception:

  Along with an attribute for the type of view being displayed in the container, in the *usView* field specify the CV_EXACTLENGTH (0x10000000L) flag. For example:

  ```
  CV_EXACTLENGTH | CV_ICON
  ```

  Note: The *usView* field if used for specifying the exact match attribute, and the type of view. Despite the "us" prefix this field is a ULONG. The "us" prefix is used in the header files to maintain backward compatibility.

Specifying Container Titles

The container control can have a non-scrollable title that consists of one or more lines of text. The container control does not limit the number of lines or the number of characters in each line. If specified, this title is the first line or lines of the container control. The text of the title is determined by the application and can be used to identify the container or to contain status information. Figure 8-22 on page 8-34 shows an example of a container title.

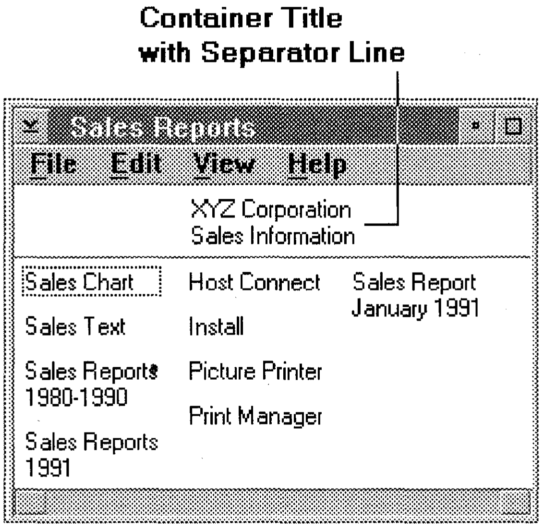

Figure 8-22. Non-Flowed Text View with Container Title

The CA_CONTAINERTITLE attribute must be set to include a title in a container window, as shown in Figure 8-23 on page 8-35. The default is no container title.

If you do not want the user to be able to edit the container title directly, you can set the CA_TITLEREADONLY attribute. The default is that the container title can be edited.

Below the title in Figure 8-22, a horizontal line separates the container title from the container items. The CA_TITLESEPARATOR attribute must be set in order to include a separator line in a container window. The default is no separator line, as shown in Figure 8-23 on page 8-35.

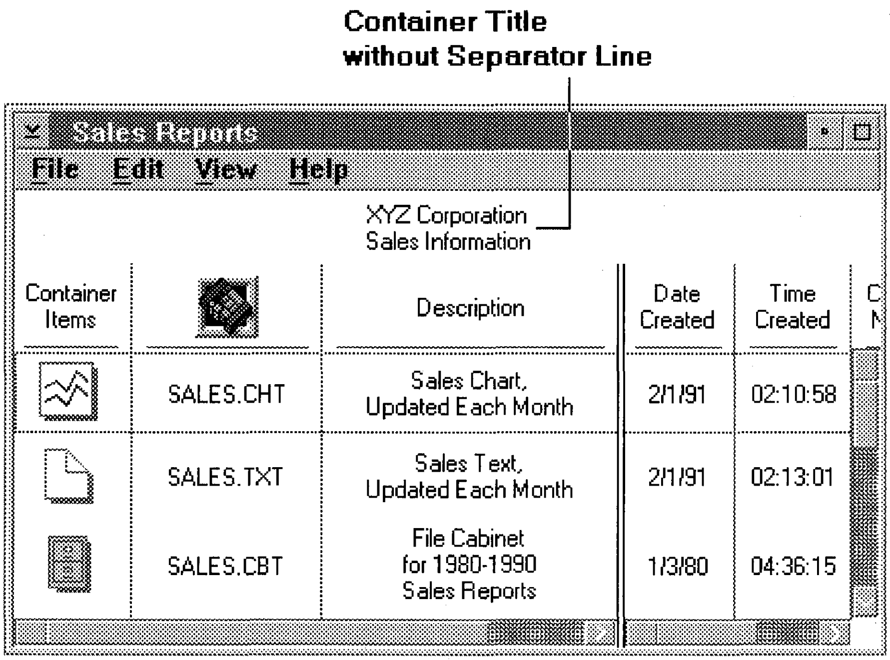

Figure 8-23. Split Details View with Container Title

The container titles in both figures are centered. This is the default. However, the CA_TITLECENTER, CA_TITLELEFT, or CA_TITLERIGHT attribute can be used to specify whether a container title is to be centered, left-justified, or right-justified.

All the container attributes described here are attributes of the CNRINFO data structure's *flWindowAttr* field.

Specifying Fonts and Colors

A different font can be specified for each view. The same font is used for the text within each view. Text color can be configured from the system control panel. The application can override the system-defined font and colors by using WinSetPresParam.

The font and color can be changed for the text in all views. However, font and color cannot be changed for text in individual columns in the Details view. Therefore, all text in the details view, including the container title, columns, and column headings, has the same font and color.

Drawing Container Items and Painting Backgrounds

The container control enables your application to paint the container's background, draw the container items, or both. If the CA_OWNERPAINTBACKGROUND attribute is set, the container control sends the CM_PAINTBACKGROUND message to itself. Your application can control background painting by subclassing the container control and intercepting the

CM_PAINTBACKGROUND message. CA_OWNERPAINTBACKGROUND is an attribute of the CNRINFO data structure's *flWindowAttr* field.

To support *ownerdraw*, the drawing of container items by the application, the container control provides the CA_OWNERDRAW attribute of the CNRINFO data structure's *flWindowAttr* field. If this attribute is set and the application processes the WM_DRAWITEM window message, the application is responsible for drawing each container item, including the types of emphasis.

In addition, the container control supports ownerdraw for each column in the Details view. This support is indicated by the CFA_OWNER attribute, which is specified in the FIELDINFO data structure's *flData* field.

If the CA_OWNERDRAW attribute or CFA_OWNER attribute is set, the container control sends the application a WM_DRAWITEM message with a pointer to an OWNERITEM data structure as the *owneritem* parameter.

Filtering Container Items

If the CRA_FILTERED attribute is set for a container item, that item is not displayed. Therefore, filtering can be used to hide container items. CRA_FILTERED is an attribute of the RECORDCORE data structure's *flRecordAttr* field.

Optimizing Container Memory Usage

The container control provides an option to enable you to develop applications that minimize the amount of memory used for each container record. This is done by specifying the CCS_MINIRECORDCORE style bit when the container is created, which causes a smaller version of the RECORDCORE data structure, MINIRECORDCORE, to be used. Table 8-2 shows the differences between these two data structures.

Table 8-2. Differences between RECORDCORE and MINIRECORDCORE

RECORDCORE	MINIRECORDCORE
Up to eight image handles can be specified for each record.	Only one image handle can be specified for each record. **Note:** This image must be an icon.
Up to four text strings can be specified for each record.	Only one text string can be specified for each record.

Allocating Memory for when Using MINIRECORDCORE

The sample code illustrated in Figure 8-24 shows how to allocate memory for one container record when the MINIRECORDCORE data structure is used. A pointer to the MINIRECORDCORE data structure is returned.

```
HWND             hwndCnr;         /* Container window handle           */
PMINIRECORDCORE pRecord;         /* Pointer to MINIRECORDCORE structure */
ULONG            nRecords = 1;    /* 1 record to be allocated           */

pRecord =
  (PMINIRECORDCORE)WinSendMsg(
    hwndCnr,                      /* Container window handle           */
    CM_ALLOCRECORD,               /* Message for allocating the record   */
    NULL,                         /* No additional memory              */
    (MPARAM)nRecords);            /* Number of records to be allocated   */
```

Figure 8-24. Sample Code for Allocating Memory for Smaller Container Records

Sharing Records among Multiple Containers

The container control enables the application to share records that are allocated among multiple containers in the same process. That is, records can be allocated once and then inserted into many containers in the same process. Only one copy of each record is in memory, but the container provides the flexibility for the records to appear as though they are independent of one another.

When a record is inserted into the container, the *flRecordAttr* and *ptlIcon* fields of the record structure are saved internally. The values in these fields cause the record attributes for all views and the icon position for the Icon view to be associated with the specific container into which the record is inserted. If the same record is inserted into multiple containers, the attributes and icon location of each record are maintained separately. The application uses the CM_QUERYRECORDINFO message to retrieve the current values of these two fields for a particular record in a specific container.

Invalidating Records Shared by Multiple Containers

When a record is invalidated by an application, the *flRecordAttr* and *ptlIcon* fields are saved internally, just as when a record is inserted. The CM_QUERYRECORDINFO message is used to acquire the current data for each record that is being invalidated. After querying the current data, the data can be changed before invalidating its record.

Freeing Records Shared by Multiple Containers

When an application attempts to free a record in an open container, the record is freed only if it is not being used in any other open container. The methods of freeing records in an open container are to use the CM_FREERECORD message, or use the CM_REMOVERECORD message and specify the CMA_FREE attribute.

Sample Code for Container Controls

This section illustrates a complete container control sample program. Several parts of this program are explained in "Using Container Controls" on page 8-14.

Container Application Sample Code

The container application includes the following files:

- Contain.C
- Contain.RC
- Contain.H
- Contain.LNK
- Phones.H

Figure 8-25 illustrates the container application sample code.

```
.
================
CONTAIN.C
================

#define  INCL_WIN
#define  INCL_GPI

#include <os2.h>
#include <stdio.h>
#include <string.h>
#include "contain.h"
#include "phones.h"
```

Figure 8-25 (Part 1 of 12). Sample Code for a Container Application

```
/**********************************************************************/
/*  Program Overview:                                                 */
/*                                                                    */
/*  This program creates a frame window as a parent, then creates     */
/*  a container window as a child.  The frame window sizes the        */
/*  container to fill its client area.                                */
/*                                                                    */
/*  After the windows are created successfully, the container         */
/*  window is populated.  First, the container is sent a message to   */
/*  allocate memory for each of the records which will be inserted.   */
/*  After the memory is allocated, we set the values for each record. */
/*  (This sample program reads data from a static array - you could   */
/*  also load values from a file.) Then, the container is sent a      */
/*  message to insert the records (which makes them visible).         */
/*                                                                    */
/*  This container is read-only, which means the end user cannot      */
/*  change the title text.  It supports single selection.            */
/*                                                                    */
/*  In the message loop, we must check for WM_CONTROL messages,       */
/*  which are generated from the container control.  This sample      */
/*  processes CN_ENTER messages, when an item in the container is     */
/*  selected (either with the mouse or the keyboard), and             */
/*  CN_CONTEXTMENU messages, when a context menu is requested.  The   */
/*  context menu allows the user to change the display mode of the    */
/*  container.  Our container supports Icon, Text, and Name views.    */
/*                                                                    */
/*  When a CN_ENTER message is received, we loop through the array    */
/*  of names until we find a match.  On a match, we pop up a message  */
/*  box which contains the nickname, name, and number of the person   */
/*  selected.                                                         */
/*                                                                    */
/**********************************************************************/

#pragma linkage (main,optlink)
INT main(VOID);
VOID LoadDatabase(HWND);
```

Figure 8-25 (Part 2 of 12). Sample Code for a Container Application

```
/**********************************************************************/
/*  Main() - program entry point.                                     */
/**********************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);

HAB     hab;
HWND    hPopupMenu;
HWND    hFrameWnd, hCnrWnd;
PFNWP   SysWndProc;

INT main (VOID)
{
   HMQ         hmq;
   FRAMECDATA  fcd;
   QMSG        qmsg;

   if (!(hab = WinInitialize(0)))
     return FALSE;

   if (!(hmq = WinCreateMsgQueue(hab, 0)))
     return FALSE;

/**********************************************************************/
/*  Set up the frame control data for the frame window.               */
/**********************************************************************/
   fcd.cb = sizeof(FRAMECDATA);
   fcd.flCreateFlags = FCF_TITLEBAR |
                       FCF_SYSMENU |
                       FCF_SIZEBORDER |
                       FCF_SHELLPOSITION |
                       FCF_MINMAX |
                       FCF_TASKLIST;
   fcd.hmodResources = NULLHANDLE;
   fcd.idResources = 0;
```

Figure 8-25 (Part 3 of 12). Sample Code for a Container Application

```
/**********************************************************************/
/*  Create the frame to hold the container control.                   */
/**********************************************************************/
   hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                               WC_FRAME,
                               "Phone Book",
                               0, 0, 0, 0, 0,
                               NULLHANDLE,
                               HWND_TOP,
                               0,
                               &fcd,
                               NULL);

/**********************************************************************/
/*  Verify that the frame was created; otherwise, stop.               */
/**********************************************************************/
   if (!hFrameWnd)
     return FALSE;

/**********************************************************************/
/*  Set an icon for the frame window.                                 */
/**********************************************************************/
   WinSendMsg(hFrameWnd,
              WM_SETICON,
              (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                         SPTR_FOLDER,
                                         FALSE),
               NULL);
```

Figure 8-25 (Part 4 of 12). Sample Code for a Container Application

```
/**********************************************************************/
/*  Create the container.                                             */
/**********************************************************************/
   hCnrWnd = WinCreateWindow(hFrameWnd,
                             WC_CONTAINER,
                             NULL,
                             CCS_AUTOPOSITION |
                             CCS_READONLY |
                             CCS_SINGLESEL,
                             0, 0, 0, 0,
                             hFrameWnd,
                             HWND_BOTTOM,
                             FID_CLIENT,
                             NULL,
                             NULL);

/**********************************************************************/
/*  If we got it, fill it up.                                         */
/**********************************************************************/
   if (hCnrWnd)
     LoadDatabase(hCnrWnd);

/**********************************************************************/
/*  We must intercept the frame window's messages                     */
/*  (to capture any input from the container control).                */
/*  We save the return value (the current WndProc),                   */
/*  so we can pass it all the other                                   */
/*  messages the frame gets.                                          */
/**********************************************************************/
   SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

/**********************************************************************/
/*  Load the popup menu from the resources                            */
/*  and show the frame window.                                        */
/**********************************************************************/
   hPopupMenu = WinLoadMenu(HWND_OBJECT, NULLHANDLE, IDM_DISPLAY);

   WinShowWindow(hFrameWnd, TRUE);
```

Figure 8-25 (Part 5 of 12). Sample Code for a Container Application

```
/**************************************************************************/
/*  Standard PM message loop - get it, dispatch it.                       */
/**************************************************************************/
   while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
   {
      WinDispatchMsg(hab, &qmsg);
   }

/**************************************************************************/
/*  Clean up on the way out.                                              */
/**************************************************************************/
   WinDestroyMsgQueue(hmq);
   WinTerminate(hab);

   return TRUE;
}

/**************************************************************************/
/*  LocalWndProc() - window procedure for the frame window.               */
/*  Called by PM whenever a message is sent to the frame.                 */
/**************************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
   char                 szBuffer[80];
   CNRINFO              cnrInfo;
   PNOTIFYRECORDENTER   Selected;
   POINTL               pt;
   int                  x;

   switch(msg)
   {
      case WM_CONTROL:
      switch (SHORT2FROMMP(mp1))
      {
```

Figure 8-25 (Part 6 of 12). Sample Code for a Container Application

```
/***********************************************************************/
/*  Context menu - usually right mouse button clicked                 */
/*  on window. Popup a menu to allow the user to                      */
/*  select a new view of the container.                               */
/***********************************************************************/
         case CN_CONTEXTMENU:
            WinQueryPointerPos(HWND_DESKTOP, &pt);
            WinPopupMenu(HWND_DESKTOP,
                         hwnd,
                         hPopupMenu,
                         (SHORT)pt.x,
                         (SHORT)pt.y,
                         IDM_ICON,
                         PU_NONE |
                         PU_MOUSEBUTTON1 |
                         PU_KEYBOARD |
                         PU_SELECTITEM);
            break;

         case CN_ENTER:

/***********************************************************************/
/*  User selected an item - we take the icon text                     */
/*  and spin through the array of Friends, looking for                */
/*  a match - on match, print out the phone number                    */
/***********************************************************************/
         Selected = (PNOTIFYRECORDENTER)mp2;
         for (x = 0; x < MAXFRIENDS; x++)
         {
            if (!strcmpi(Friends[x].NickName,
              Selected->pRecord->pszIcon))
            {
```

Figure 8-25 (Part 7 of 12). Sample Code for a Container Application

```
                sprintf(szBuffer,
                        "'%s' (%s) %s",
                        Friends[x].NickName,
                        Friends[x].FullName,
                        Friends[x].Phone);
                        WinMessageBox(HWND_DESKTOP,
                                      HWND_DESKTOP,
                                      szBuffer,
                                      "Phone",
                                      0,
                                      MB_OK);
             }
          }
          break;
       }
       break;

       case WM_COMMAND:
          switch (SHORT1FROMMP(mp1))
          {
             case IDM_ICON:
                cnrInfo.flWindowAttr = CV_ICON;
                break;
             case IDM_NAME:
                cnrInfo.flWindowAttr = CV_NAME;
                break;
             case IDM_TEXT:
                cnrInfo.flWindowAttr = CV_TEXT;
                break;
             default:
                return (*SysWndProc)(hwnd, msg, mp1, mp2);
                break;
          }

          WinSendMsg(hCnrWnd,
                     CM_SETCNRINFO,
                     &cnrInfo,
                     MPFROMLONG(CMA_FLWINDOWATTR));
          break;
```

Figure 8-25 (Part 8 of 12). Sample Code for a Container Application

```
/**********************************************************************/
/*  Send the message to the usual WC_FRAME WndProc.                   */
/**********************************************************************/
         default:
            return (*SysWndProc)(hwnd, msg, mp1, mp2);
            break;
   }
   return (*SysWndProc)(hwnd, msg, mp1, mp2);
}

/**********************************************************************/
/*  LoadDatabase() - utility function                                 */
/*  called after the WC_CONTAINER window is created successfully,     */
/*  allocates and populates container records, and then inserts       */
/*  the records into the container window.                            */
/**********************************************************************/
VOID LoadDatabase (HWND hWnd)
{
   HWND             hIcon;
   PRECORDCORE      Address, FirstRec;
   RECORDINSERT     recsIn;
   ULONG            x;

/**********************************************************************/
/*  The Icon view for each of the records in the                      */
/*  container will use the standard File icon,                        */
/*  so we grab the handle now for reference later.                    */
/**********************************************************************/
   hIcon = WinQuerySysPointer(HWND_DESKTOP, SPTR_FILE, FALSE);

/**********************************************************************/
/*  Allocate MAXFRIENDS records all at once -                         */
/*  CM_ALLOCRECORD returns them in a linked list.                     */
/**********************************************************************/
   Address = (PRECORDCORE)WinSendMsg(hWnd,
                                  CM_ALLOCRECORD,
                                  0,
                                  MPFROMLONG(MAXFRIENDS));
```

Figure 8-25 (Part 9 of 12). Sample Code for a Container Application

```
/*********************************************************************/
/*  We will need the first record's address to                       */
/*  insert them into the container.                                 */
/*********************************************************************/
   FirstRec = Address;

/*********************************************************************/
/*  Loop through the address book, loading as we go.                 */
/*  Because the CM_ALLOCRECORD returns a linked list,                */
/*  the address of the next record is retrieved                      */
/*  from each record as we go (preccNextRecord).                     */
/*********************************************************************/
   for (x = 0; x < MAXFRIENDS; x++)
   {
      Address->cb       = sizeof(RECORDCORE); /* Standard records     */
      Address->hptrIcon = hIcon;              /* File icon            */
      Address->pszIcon  = Friends[x].NickName;
      Address->pszName  = Friends[x].FullName;
      Address->pszText  = Friends[x].FullName;
      Address = Address->preccNextRecord;     /* Next record in list  */
   }

/*********************************************************************/
/*  Set up the insert record structure to place                      */
/*  the records in the container.                                    */
/*********************************************************************/
   recsIn.cb = sizeof(RECORDINSERT);

   /* Put the records in after any others */
   recsIn.pRecordOrder = (PRECORDCORE)CMA_END;

   /* All the records are top level (not children of other records) */
   recsIn.pRecordParent = NULL;

   /* The icons are top level */
   recsIn.zOrder = (USHORT)CMA_TOP;

   /* Redraw the container */
   recsIn.fInvalidateRecord = TRUE;

   /* Set the number of records to insert */
   recsIn.cRecordsInsert = MAXFRIENDS;
```

Figure 8-25 (Part 10 of 12). Sample Code for a Container Application

```
/**********************************************************************/
/*  Insert the records into the container.                            */
/**********************************************************************/
   WinSendMsg(hWnd,
              CM_INSERTRECORD,
              (PRECORDCORE)FirstRec,
              &recsIn);
}

================
CONTAIN.RC
================
#include <os2.h>
#include "contain.h"

MENU            IDM_DISPLAY
BEGIN
    MENUITEM    "Icon",     IDM_ICON
    MENUITEM    "Text",     IDM_TEXT
    MENUITEM    "Name",     IDM_NAME
END

================
CONTAIN.H
================
#define DLG_ADDRBOOK     100
#define CNR_ADDRBOOK     101
#define PB_ADD           102
#define PB_DIAL          103
#define PHONEBOOK        256
#define IDM_DISPLAY      400
#define IDM_ICON         401
#define IDM_NAME         402
#define IDM_TEXT         403

================
CONTAIN.LNK
================
contain.obj
contain.exe
contain.map
contain.def
```

Figure 8-25 (Part 11 of 12). Sample Code for a Container Application

```
================
PHONES.H
================
#define MAXFRIENDS  9

/*********************************************************************/
/*  This is a simple phone book database.                            */
/*********************************************************************/
typedef struct _Phones
{
   PSZ NickName;
   PSZ FullName;
   PSZ Phone;
}PhoneBook;

/*********************************************************************/
/*  Normal programs would read this data from a file.                */
/*********************************************************************/
PhoneBook Friends[MAXFRIENDS] =
{
   "Giles",      "Kevin Giles",        "214-555-1212",
   "Bubba",      "Hank Smith",         "713-555-1212",
   "Fred",       "Fred Bicycle",       "817-555-1212",
   "Jack",       "Jack Anjill",        "919-555-1212",
   "John",       "John Richards",      "214-555-1212",
   "Toni",       "Toni Henderson",     "919-555-1212",
   "Babe",       "George Herman Ruth", "212-555-1212",
   "Kevin",      "Kevin Kortrel",      "817-555-1212",
   "Honest Abe", "Abraham Lincoln",    "none"
};
```

Figure 8-25 (Part 12 of 12). Sample Code for a Container Application

Related Window Messages

This section covers the window messages that are related to container controls.

CM_ALLOCDETAILFIELDINFO

This message allocates memory for one or more FIELDINFO structures.

Parameters

param1

nFieldInfo (USHORT)
Number of FIELDINFO structures to be allocated.

The value of this parameter must be greater than 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

pFieldInfo (PFIELDINFO)
Pointer or error.

0 Reserved value, 0. The WinGetLastError function may return the following errors:

- PMERR_INSUFFICIENT_MEMORY
- PMERR_INVALID_PARAMETERS.

Other If the *nFieldInfo* parameter has a value of 1, a pointer to a FIELDINFO data structure is returned.

A pointer to the first FIELDINFO structure in a linked list of FIELDINFO structures is returned if the *nFieldInfo* parameter has a value greater than 1. The pointer to the next FIELDINFO structure is set in each *pNextFieldInfo* field of the FIELDINFO data structure. The last pointer is set to NULL.

CM_ALLOCRECORD

This message allocates memory for one or more RECORDCORE structures.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

Parameters

param1

cbRecordData (ULONG)
Bytes of additional memory.

The number of bytes of additional memory that you want to reserve for your application's private use. This parameter must have a value between 0 and 64,000. If the value is 0, no additional memory is allocated, but a RECORDCORE data structure is allocated.

param2

nRecords (USHORT)
Number of records.

The number of container records to be allocated. This parameter must have a value greater than 0.

Returns

pRecord (PRECORDCORE)
Returns a pointer or an error.

NULL Allocation failed. The WinGetLastError function may return the following errors:

- PMERR_INSUFFICIENT_MEMORY
- PMERR_INVALID_PARAMETERS.

Other If the *nRecords* parameter has a value of 1, a pointer to a RECORDCORE structure is returned.

If the *nRecords* parameter has a value greater than 1, a pointer to the first RECORDCORE structure in the linked list of records is returned. The pointer to the next container record is set in the *preccNextRecord* field in each RECORDCORE data structure. The last pointer is set to NULL.

CM_ARRANGE

This message arranges the container records in the icon view of the container control.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Icon/text or bit-map/text pairs were successfully arranged.
FALSE	An error occurred.

CM_CLOSEEDIT

This message closes the window that contains the multiple-line entry (MLE) field used to edit container text directly.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	The direct editing of container item text was successfully ended.
FALSE	The direct editing of container item text was not successfully ended. The WinGetLastError function may return the following error: PMERR_INSUFFICIENT_MEMORY.

CM_COLLAPSETREE

This message causes one parent item in the tree view to be collapsed.

Parameters

param1

pRecord (PRECORDCORE)
Pointer to the RECORDCORE structure that is to be collapsed.

If this is NULL, all expanded parent items are collapsed.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE The item was successfully collapsed.
FALSE An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_ERASERECORD

This message erases the source record from the current view when a move occurs as a result of direct manipulation.

Parameters

param1

pRecord (PRECORDCORE)
Pointer to the container record that is to be erased from the current view.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE The record was successfully erased.
FALSE The record was not erased. The WinGetLastError function may return the following errors:
- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY.

CM_EXPANDTREE

This message causes one parent item in the tree view to be expanded.

Parameters

param1

pRecord (PRECORDCORE)
Pointer to the RECORDCORE structure that is to be expanded.

If this is NULL, all collapsed parent items are expanded.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE The item was successfully expanded.

FALSE An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_FILTER

This message filters the contents of a container so that a subset of the container items is viewable.

Parameters

param1

pfnFilter (PFN)
Pointer to an application-supplied filter function.

param2

pStorage (PVOID)
Application use.

Available for application use.

Returns

rc (BOOL)
Success indicator.

TRUE — A subset was successfully created.

FALSE — An error occurred. The WinGetLastError function may return the following errors:

- PMERR_NO_FILTERED_ITEMS
- PMERR_INSUFFICIENT_MEMORY.

CM_FREEDETAILFIELDINFO

This message frees the memory associated with one or more FIELDINFO structures.

Parameters

param1

pFieldInfoArray (PVOID)
Pointer to an array of pointers to FIELDINFO structures that are to be freed.

param2

cNumFieldInfo (USHORT)
Number of structures.

Number of FIELDINFO structures to be freed.

Returns

rc (BOOL)
Success indicator.

TRUE Memory associated with a specified FIELDINFO structure or structures in the container was freed.

FALSE Associated memory was not freed. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_MEMORY_DEALLOCATION_ERR
- PMERR_FI_CURRENTLY_INSERTED.

CM_FREERECORD

This message frees the memory associated with one or more RECORDCORE structures.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

Parameters

param1

pRecordArray (PVOID)
Pointer to an array of pointers to RECORDCORE structures that are to be freed.

param2

cNumRecord (USHORT)
Number of records.

Number of container records to be freed.

Returns

rc (BOOL)
Success indicator.

TRUE Memory associated with a record or records in the container was freed.

FALSE Associated memory was not freed. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_MEMORY_DEALLOCATION_ERR
- PMERR_RECORD_CURRENTLY_INSERTED.

CM_HORZSCROLLSPLITWINDOW

This message scrolls a split window in the split details view.

Parameters

param1

usWindow (USHORT)
Window indicator.

CMA_LEFT	The left split window is scrolled.
CMA_RIGHT	The right split window is scrolled.

param2

lScrollInc (LONG)
Amount to scroll.

Amount (in pixels) by which to scroll the window.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_INSERTDETAILFIELDINFO

This message inserts one or more FIELDINFO structures into a container control.

Parameters

param1

pFieldInfo (PFIELDINFO)
Pointer to the FIELDINFO structure or structures to insert.

param2

pFieldInfoInsert (PFIELDINFOINSERT)
Pointer to the FIELDINFOINSERT data structure.

See "FIELDINFOINSERT" on page 8-103 for the descriptions of this structure's fields as they apply to the CM_INSERTDETAILFIELDINFO message.

Returns

cFields (USHORT)
Number of structures.

0 The FIELDINFO structure or structures were not inserted. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY
- PMERR_FI_CURRENTLY_INSERTED.

Other The number of FIELDINFO structures in the container.

CM_INSERTRECORD

This message inserts one or more RECORDCORE structures into a container control.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

Parameters

param1

pRecord (PRECORDCORE)
Pointer to the RECORDCORE structure or structures to insert.

param2

pRecordInsert (PRECORDINSERT)
Pointer to the RECORDINSERT data structure.

See "RECORDINSERT" on page 8-117 for definitions of this structure's fields as they apply to the CM_INSERTRECORD message.

Returns

cRecords (ULONG)
Number of structures.

0 The RECORDCORE structure was not inserted. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY
- PMERR_RECORD_CURRENTLY_INSERTED.

Other The number of RECORDCORE structures in the container.

CM_INVALIDATEDETAILFIELDINFO

This message notifies the container control that any or all FIELDINFO structures are not valid and that the view must be refreshed.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	FIELDINFO structures were successfully refreshed.
FALSE	FIELDINFO structures were not successfully refreshed.

CM_INVALIDATERECORD

This message notifies the container control that a RECORDCORE structure or structures are not valid and must be refreshed.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

Parameters

param1

pRecordArray (PVOID)
Pointer to an array of pointers to RECORDCORE structures that are to be refreshed.

param2

cNumRecord (USHORT)
Number of container records to be refreshed.

If the *cNumRecord* parameter has a value of 0, all of the records in the container are refreshed and the *pRecordArray* parameter is ignored.

fInvalidateRecord (USHORT)
Flags used to optimize container record invalidation.

The CMA_REPOSITION, CMA_NOREPOSITION, and CMA_TEXTCHANGED attributes are mutually exclusive. However, any of them can be combined with the CMA_ERASE attribute by using a logical OR operator (|).

CMA_ERASE	Flag used when the icon view is displayed to minimize painting of a container record's background when it has changed. If specified, the background is erased when the display is refreshed. The default is to not erase the background when the display is refreshed.
CMA_REPOSITION	Flag used to reposition all container records. This flag must be used if container records are inserted or removed, or if many changes have occurred. If a container record is inserted, the *pRecordArray* parameter points to the inserted record. If a container record is removed, the *pRecordArray* parameter points to the record that precedes the removed one. If several container records have changed, an array of container record pointers must be used. The container determines the first record to be invalidated. This is the default.
CMA_NOREPOSITION	Flag used to indicate that container records do not need to be repositioned. The container draws the record or records pointed to in the *pRecordArray* parameter. The container does not do any validation; therefore it is the application's responsibility to make sure repositioning is not needed or changing the longest text line is not necessary.
CMA_TEXTCHANGED	Flag used if text has changed and you do not know whether repositioning is needed. The container determines whether the longest line or the height of the record has changed. If so, the container repositions and redraws the necessary visible container records. It may be necessary to reposition the container records if the number of lines of text has changed. **Warning:** The application must send a CM_INVALIDATERECORD message if text changes. Otherwise, any further processing is unreliable.

Returns

rc (BOOL)
Success indicator.

TRUE Records were successfully refreshed.

FALSE An error occurred. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY.

CM_OPENEDIT

This message opens the window that contains the multiple-line entry (MLE) field used to edit container text directly.

Parameters

param1

pCnrEditData (PCNREDITDATA)
Pointer to the CNREDITDATA structure.

See "CNREDITDATA" on page 8-88 for definitions of this structure's fields as they apply to the CM_OPENEDIT message.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE Direct editing of container text was successfully started.

FALSE Direct editing of container text was not successfully started. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_PAINTBACKGROUND

This message informs an application whenever a container's background is painted if the CA_OWNERPAINTBACKGROUND attribute of the CNRINFO data structure is specified.

Parameters

param1

pOwnerBackground (POWNERBACKGROUND)
Pointer to the OWNERBACKGROUND structure.

See "OWNERBACKGROUND" on page 8-110 for definitions of this structure's fields as they apply to the CM_PAINTBACKGROUND message.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Process indicator.

TRUE	The application processed the CM_PAINTBACKGROUND message.
FALSE	The application did not process the CM_PAINTBACKGROUND message.

CM_QUERYCNRINFO

This message returns the container's CNRINFO structure.

Parameters

param1

pCnrInfo (PCNRINFO)
Pointer to a buffer into which the CNRINFO structure is copied.

param2

cbBuffer (USHORT)
Number of bytes.

Maximum number of bytes to copy.

Returns

cbBytes (USHORT)
Success indicator.

0 Container data was not successfully returned. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

Other Actual number of bytes copied.

CM_QUERYDETAILFIELDINFO

This message returns a pointer to the requested FIELDINFO structure.

Parameters

param1

pfldinfoBase (PFIELDINFO)
Pointer to the FIELDINFO structure used to search for the next or previous column.

If the CMA_FIRST or CMA_LAST attribute is specified, this is ignored.

param2

cmd (USHORT)
Command that indicates which FIELDINFO structure to retrieve.

CMA_FIRST	First column in the container.
CMA_LAST	Last column in the container.
CMA_NEXT	Next column in the container.
CMA_PREV	Previous column in the container.

Returns

pFieldInfo (PFIELDINFO)
Pointer to the FIELDINFO structure for which data was requested.

NULL No FIELDINFO structures to retrieve.

–1 The data from the FIELDINFO structure was not returned. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

Other Pointer to the FIELDINFO structure for which data was requested.

CM_QUERYDRAGIMAGE

This message returns a handle to the icon or bit map for the record in the current view.

Parameters

param1

pRecord (PRECORDCORE)
Pointer to the RECORDCORE structure that is to be queried for the image.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

hImage (LHANDLE)
Image handle.

NULLHANDLE — If no image is defined, NULLHANDLE is returned.

Other — Handle of an icon or bit map.

- If the CA_DRAWICON attribute and the CV_MINI style bit are specified, the RECORDCORE structure's *hptrMiniIcon* field is returned.
- If the CA_DRAWICON attribute is specified without the CV_MINI style bit, the RECORDCORE structure's *hptrIcon* field is returned.
- If the CA_DRAWBITMAP attribute and the CV_MINI style bit are specified, the RECORDCORE structure's *hbmMiniBitmap* field is returned.
- If the CA_DRAWBITMAP attribute is specified without the CV_MINI style bit, the RECORDCORE structure's *hbmBitmap* field is returned.

CM_QUERYRECORD

This message returns a pointer to the requested RECORDCORE structure.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

Parameters

param1

pRecord (PRECORDCORE)
Pointer to the RECORDCORE structure used to search for the next or previous container record.

If the CMA_FIRST or CMA_LAST attribute is specified, this is ignored.

param2

cmd (USHORT)
Command that indicates which container record to retrieve:

CMA_FIRST	First record in the container.
CMA_FIRSTCHILD	First child record of *pRecord* specified in *param1*.
CMA_LAST	Last record in the container.
CMA_LASTCHILD	Last child record of *pRecord* specified in *param1*.
CMA_NEXT	Next record of *pRecord* specified in *param1*.
CMA_PARENT	Parent of *pRecord* specified in *param1*.
CMA_PREV	Previous record of *pRecord* specified in *param1*.

fsSearch (USHORT)
Enumeration order.

Specifies the enumeration order. This value is one of the following:

CMA_ITEMORDER	Container records are enumerated in item order, first to last.
CMA_ZORDER	Container records are enumerated by z-order, from first record in the z-order to the last record. The last z-order record is the last record to be drawn. This flag is valid for the icon view only.

Returns

pRecord (PRECORDCORE)
Pointer to the RECORDCORE structure for which data was requested.

NULL No RECORDCORE structures to retrieve.

–1 The container record data was not returned. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

Other Pointer to the container record for which data was requested.

CM_QUERYRECORDEMPHASIS

This message queries for a container record with the specified emphasis attributes.

Parameters

param1

pSearchAfter (PRECORDCORE)
Pointer to the specified container record.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

The values of this parameter can be:

CM_FIRST	Start the search with the first record in the container.
Other	Start the search after the record specified by this pointer.

param2

fEmphasisMask (USHORT)
Emphasis attribute.

Specifies the emphasis attribute of the container record. The following states can be combined using a logical OR operator (|):

CRA_COLLAPSED	Specifies that a record is collapsed.
CRA_CURSORED	Specifies that a record will be drawn with a selection cursor.

CRA_DISABLED	Specifies that a record will be drawn with unavailable-state emphasis.
CRA_DROPONABLE	Specifies that a record can be a target for direct manipulation.
CRA_EXPANDED	Specifies that a record is expanded.
CRA_FILTERED	Specifies that a record is filtered and, therefore, hidden from view.
CRA_INUSE	Specifies that a record will be drawn with in-use emphasis.
CRA_PICKED	Specifies that the container record willl be picked up as part of the drag set.
CRA_SELECTED	Specifies that a record will be drawn with selected-state emphasis.
CRA_SOURCE	Specifies that a record will be drawn with source-menu emphasis.

Returns

pRecord (PRECORDCORE)

Pointer to the record with the specified emphasis.

NULL This implies that none of the records that follow the pointer specified in the *pSearchAfter* parameter meet those specifications.

–1 The container record data was not returned.

The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS (1208)

Other Pointer to a container record with the specified emphasis.

This is the first record that follows the record pointed to by the *pSearchAfter* parameter and satisfies the criteria specified in the *fEmphasisMask* parameter. To find the next record that satisfies this criteria, send this message again, but this time use the value returned in the *pRecord* parameter for the value of the *pSearchAfter* parameter.

CM_QUERYRECORDFROMRECT

This message queries for a container record that is bounded by the specified rectangle.

Parameters

param1

pSearchAfter (PRECORDCORE)
Pointer to the specified container record.

To get all the container records within the specified rectangle, this message is sent repeatedly, each time this parameter is set to the pointer that is returned by the previous usage of this message.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

The values of this parameter can be:

CMA_FIRST	Start the search with the first record in the container.
Other	Start the search after the record specified by this pointer.

param2

pQueryRecFromRect (PQUERYRECFROMRECT)
Pointer to the QUERYRECFROMRECT data structure.

See "QUERYRECFROMRECT" on page 8-112 for definitions of this structure's fields as they apply to the CM_QUERYRECORDFROMRECT message.

Returns

pRecord (PRECORDCORE)
Pointer to the container records within the bounding rectangle.

NULL No container records are within the bounding rectangle.

–1 The container record data was not returned. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

Other Pointer to the container record within the bounding rectangle.

CM_QUERYRECORDINFO

This message updates the specified records with the current information for the container.

Parameters

param1

pRecordArray (PVOID)
Pointer to an array of pointers to RECORDCORE structures to which the current information is to be copied.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE all applicable data structures and messages.

param2

cNumRecord (USHORT)
Number of records.

The number of container records to be updated. If the *cNumRecord* parameter has a value of 0, all of the records in the container are updated and the *pRecordArray* parameter is ignored.

Returns

rc (BOOL)
Success indicator.

TRUE Record information was successfully updated.
FALSE An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_QUERYRECORDRECT

This message returns the rectangle of the specified container record, relative to the container window origin.

Parameters

param1

prclItem (PRECTL)
Pointer to the RECTL structure, into which the rectangular coordinates are placed.

param2

pQueryRecordRect (PQUERYRECORDRECT)
Pointer to the QUERYRECORDRECT structure.

See "QUERYRECORDRECT" on page 8-113 for definitions of this structure's fields as they apply to the CM_QUERYRECORDRECT message.

Returns

rc (BOOL)
Success indicator.

TRUE A rectangle with valid coordinates is returned.

FALSE The rectangle is not successfully returned. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_QUERYVIEWPORTRECT

This message returns a rectangle that contains the coordinates of the container's client area. These are virtual coordinates that are relative to the origin of the coordinate space requested.

Parameters

param1

prclViewport (PRECTL)
Pointer to the RECTL structure.

Pointer to the RECTL structure that the virtual coordinates of the client area rectangle are to be written into.

param2

usIndicator (USHORT)
Coordinate space indicator.

One of the following must be used:

CMA_WINDOW Returns the client area rectangle in container window coordinates.

CMA_WORKSPACE Return the client area rectangle in coordinates relative to the origin of the container's workspace.

fRightSplitWindow (BOOL)
Flag.

Flag that specifies the right or left window in the split details view. This flag is ignored if the view is not the split details view.

TRUE Right split window is returned.
FALSE Left split window is returned.

Returns

rc (BOOL)
Success indicator.

TRUE The client area rectangle was returned successfully.

FALSE An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_REMOVEDETAILFIELDINFO

This message removes one, multiple, or all FIELDINFO structures from the container control.

Parameters

param1

pFieldInfoArray (PVOID)
Pointer to an array of pointers to FIELDINFO structures that are to be removed.

param2

cNumFieldInfo (USHORT)
Number of FIELDINFO structures to be removed.

If the *cNumFieldInfo* parameter has a value of 0, all of the FIELDINFO structures in the container are removed and the *pFieldInfoArray* parameter is ignored.

fRemoveFieldInfo (USHORT)
Flags.

Flags that show whether memory must be freed and FIELDINFO structures invalidated.

CMA_FREE If specified, FIELDINFO structures are removed and memory associated with the FIELDINFO structures is freed. If not specified, FIELDINFO structures are removed and no memory is freed; this is the default.

CMA_INVALIDATE If specified, after FIELDINFO structures are removed, the container is invalidated, and any necessary repositioning of the FIELDINFO structures is performed. If not specified, invalidation is not performed.

Returns

cFields (SHORT)
Number of structures.

–1 An error occurred. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_MEMORY_DEALLOCATION_ERR.

Other The number of FIELDINFO structures that remain in the container.

CM_REMOVERECORD

This message removes one, multiple, or all RECORDCORE structures from the container control.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

Parameters

param1

pRecordArray (PVOID)
Pointer to an array of pointers to RECORDCORE structures that are to be removed.

param2

cNumRecord (USHORT)
Number of records.

Number of container records to be removed. If the *cNumRecord* parameter has a value of 0, all of the records in the container are removed and the *pRecordArray* parameter is ignored.

fRemoveRecord (USHORT)
Flags.

Flags that show whether memory must be freed and container records invalidated.

CMA_FREE	If specified, RECORDCORE structures are removed and memory associated with the RECORDCORE structures is freed. If not specified, RECORDCORE structures are removed and no memory is freed; this is the default.
CMA_INVALIDATE	If specified, after RECORDCORE structures are removed the container is invalidated and any necessary repositioning of the container records is performed. If not specified, invalidation is not performed. This option is not valid in the icon view unless the CCS_AUTOPOSITION style bit is not set. In the icon view, the container record is refreshed if the CCS_AUTOPOSITION style bit is set. regardless of whether the CMA_INVALIDATE attribute is set.

Returns

cRecords (LONG)
Number of structures.

−1 An error occurred. The WinGetLastError function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_MEMORY_DEALLOCATION_ERR.

Other Number of root level RECORDCORE structures that remain in the container.

CM_SCROLLWINDOW

This message scrolls an entire container window.

Parameters

param1

fsScrollDirection (USHORT)
Scroll direction.

Direction in which to scroll the container window.

CMA_VERTICAL	Scroll vertically.
CMA_HORIZONTAL	Scroll horizontally.

param2

lScrollInc (LONG)
Scroll increment.

Amount (in pixels) by which to scroll the window.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS.

CM_SEARCHSTRING

This message returns the pointer to a container record whose text matches the string.

Parameters

param1

pSearchString (PSEARCHSTRING)
Pointer to the SEARCHSTRING structure.

See "SEARCHSTRING" on page 8-118 for definitions of this structure's fields as they apply to the CM_SEARCHSTRING message.

param2

pSearchAfter (PRECORDCORE)
Pointer to the starting container record.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CMA_FIRST	Start the search at the first container record.
Other	Start the search after the container record specified by this pointer. To get all of the records in the container whose text matches the string, this message is sent repeatedly. Each time this message is sent, the *pSearchAfter* parameter contains a pointer to the last record that was found.

Returns

pRecord (PRECORDCORE)
Pointer to the found container record.

NULL No container record's text matches the search string.

–1 An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

Other Pointer to the container record whose text matches the search string.

CM_SETCNRINFO

This message sets or changes the data for the container control.

Parameters

param1

pCnrInfo (PCNRINFO)
Pointer to the CNRINFO structure from which to set the data for the container.

param2

ulCnrInfoFl (ULONG)
Flags.

Flags that show which fields are to be set.

CMA_PSORTRECORD	Pointer to the comparison function for sorting container records. If NULL, which is the default condition, no sorting is performed. Sorting only occurs during record insertion and when changing the value of this field. The third parameter of the comparison function, *pStorage*, must be NULL. See CM_SORTRECORD for a further description of the comparison function.
CMA_PFIELDINFOLAST	Pointer to the last column in the left window of the split details view. The default is NULL, causing all columns to be positioned in the left window.
CMA_PFIELDINFOOBJECT	Pointer to a column that represents an object in the details view. This FIELDINFO structure must contain icons or bit maps. In-use emphasis is applied to this column of icons or bit maps only. The default is the leftmost column in the unsplit details view, or the leftmost column in the left window of the split details view.

CMA_CNRTITLE	Text for the container title. The default is NULL.
CMA_FLWINDOWATTR	Container window attributes.
CMA_PTLORIGIN	Lower-left origin of the container window in virtual workspace coordinates, used in the icon view. The default origin is **(0,0)**.
CMA_DELTA	An application-defined threshold, or number of records, from either end of the list of available records. Used when a container needs to handle large amounts of data. The default is 0. Refer to the description of the container control in the *OS/2 Programming Guide* for more information about specifying deltas.
CMA_SLBITMAPORICON	The size (in pels) of icons or bit maps. The default is the system size.
CMA_SLTREEBITMAPORICON	The size (in pels) of the expanded and collapsed icons or bit maps in the tree icon and tree text views.
CMA_TREEBITMAP	Expanded and collapsed bit maps in the tree icon and tree text views.
CMA_TREEICON	Expanded and collapsed icons in the tree icon and tree text views.
CMA_LINESPACING	The amount of vertical space (in pels) between the records. If this value is less than 0, a default value is used.
CMA_CXTREEINDENT	Horizontal distance (in pels) between levels in the tree view. If this value is less than 0, a default value is used.
CMA_CXTREELINE	Width of the lines (in pels) that show the relationship between items in the tree view. If this value is less than 0, a default value is used. Also, if the CA_TREELINE container attribute of the CNRINFO data structure's *flWindowAttr* field is not specified, these lines are not drawn.
CMA_XVERTSPLITBAR	The initial position of the split bar relative to the container, used in the details view. If this value is less than 0, the split bar is not used. The default value is negative one (-1).

rc (BOOL)
Success indicator.

TRUE — Container data was successfully set.
FALSE — Container data was not set. The WinGetLastError function may return the following errors:
- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY.

CM_SETRECORDEMPHASIS

This message sets the emphasis attributes of the specified container record.

Parameters

param1

pRecord (PRECORDCORE)
Pointer to the specified container record.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

param2

usChangeEmphasis (USHORT)
Change-emphasis-attribute flag.

TRUE — The container record's emphasis attribute is to be set ON if the change specified is not the same as the current state.

FALSE — The container record's emphasis attribute is to be set OFF if the change specified is not the same as the current state.

fEmphasisAttribute (USHORT)
Emphasis attribute of the container record.

The following states can be combined by using a logical OR operator (|):

CRA_CURSORED	Specifies that a record will be drawn with a selection cursor.
CRA_DISABLED	Specifies that a record will be drawn with unavailable-state emphasis.
CRA_INUSE	Specifies that a record will be drawn with in-use emphasis.
CRA_PICKED	Specifies that the container record willl be picked up as part of the drag set.
CRA_SELECTED	Specifies that a record will be drawn with selected-state emphasis.
CRA_SOURCE	Specifies that a record will be drawn with source-menu emphasis.

Returns

rc (BOOL)
Success indicator.

TRUE Successful completion

FALSE An error occurred.

The WinGetLastError function may return the following errors:

PMERR_INVALID_PARAMETERS (1208)
PMERR_INSUFFICIENT_MEMORY (203E)

CM_SORTRECORD

This message sorts the container records in the container control.

Parameters

param1

pfnCompare (PFN)
Pointer to a comparison function.

param2

pStorage (PVOID)
Application use.

Available for application use.

Returns

rc (BOOL)
Success indicator.

TRUE The records in the container were sorted.

FALSE The records in the container were not sorted. The WinGetLastError function may return the following errors:

- PMERR_COMPARISON_FAILED
- PMERR_INSUFFICIENT_MEMORY.

WM_PICKUP

This message adds objects to the drag set during a lazy drag operation.

Parameters

param1

ptlPointerPos (POINTL)
Pointer position in window coordinates relative to the bottom-left corner of the window.

param2

Reserved (ULONG)
Reserved value, must be 0.

Returns

returns

rc (BOOL)
Success indicator.

Possible values are described in the following list:

TRUE Message was processed.
FALSE Message was ignored.

WM_PRESPARAMCHANGED (in Container Controls)

For the cause of this message, see WM_PRESPARAMCHANGED.

Parameters

param1

attrtype (ULONG)
Presentation parameter attribute identity.

PP_BACKGROUNDCOLOR or PP_BACKGROUNDCOLORINDEX
Sets the background color of the container window. This color is initially set to SYSCLR_WINDOW.

PP_BORDERCOLOR or PP_BORDERCOLORINDEX
Sets the color of the title separators, column separators, and split bar. This color is initially set to SYSCLR_WINDOWFRAME.

PP_FONTNAMESIZE
Sets the font and font size of the text in the container. This font and font size defaults to the system font and font size.

PP_FOREGROUNDCOLOR or PP_FOREGROUNDCOLORINDEX
Sets the color of unselected text. This color is initially set to SYSCLR_WINDOWTEXT.

PP_HILITEBACKGROUNDCOLOR or PP_HILITEBACKGROUNDCOLORINDEX
Sets the color of selection emphasis, the color of the cursor of an unselected item in the details view, and the color of the cursor in all other views. This color is initially set to SYSCLR_HILITEBACKGROUND.

PP_HILITEFOREGROUNDCOLOR or PP_HILITEFOREGROUNDCOLORINDEX
Sets the color of the text of a selected item in all views and the color of the cursor of a selected item in the details view. This color is initially set to SYSCLR_HILITEFOREGROUND.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

Related Notification Messages

This section covers the notification messages that are related to container controls.

WM_CONTROL (in Container Controls)

For the cause of this message, see WM_CONTROL.

Parameters

param1

id (USHORT)
Container control ID.

notifycode (USHORT)
Notify code.

The container control uses the following notification codes. For the complete description of the specified *notifycode*, see Table 8-5 on page 8-123.

CN_BEGINEDIT	Container text is about to be edited.
CN_COLLAPSETREE	A parent item was collapsed in the tree view.
CN_CONTEXTMENU	The container received a WM_CONTEXTMENU message.
CN_DRAGAFTER	The container received a DM_DRAGOVER message. The CN_DRAGAFTER notification code is sent only if either the CA_ORDEREDTARGETEMPH or CA_MIXEDTARGETEMPH attribute of the CNRINFO data structure is set and the current view is the name, text, or details view.
CN_DRAGLEAVE	The container received a DM_DRAGLEAVE message.
CN_DRAGOVER	The container received a DM_DRAGOVER message. The CN_DRAGOVER notification code is sent only if the CA_ORDEREDTARGETEMPH attribute of the CNRINFO data structure is not set or the current view is the icon view or tree view.
CN_DROP	The container received a DM_DROP message.
CN_DROPNOTIFY	The container received a DM_DROPNOTIFY message.
CN_DROPHELP	The container received a DM_DROPHELP message.
CN_EMPHASIS	A container record's attributes changed.
CN_ENDEDIT	Direct editing of container text has ended.
CN_ENTER	The Enter key is pressed while the container window has the focus, or the select button is double-clicked while the pointer is over the container window.

CN_EXPANDTREE	A parent item is expanded in the tree view.
CN_HELP	The container received a WM_HELP message.
CN_INITDRAG	The drag button was pressed and the pointer was moved while the pointer was over the container control.
CN_KILLFOCUS	The container is losing the focus.
CN_PICKUP	The container received a WM_PICKUP message.
CN_QUERYDELTA	Queries for more data when a user scrolls to a preset delta value.
CN_REALLOCPSZ	Container text is edited. This message is sent before the CN_ENDEDIT notification code is sent.
CN_SCROLL	The container window scrolled.
CN_SETFOCUS	The container is receiving the focus.

param2

notifyinfo (ULONG)
Notify code information.

For the definition of this parameter, see the description of the specified *notifycode*Table 8-5 on page 8-123 .

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DRAWITEM (in Container Controls)

For the cause of this message, see WM_DRAWITEM.

Parameters

param1

id (USHORT)
Container control ID.

param2

pOwnerItem (POWNERITEM)
Pointer to an OWNERITEM data structure.

The following list defines the OWNERITEM data structure fields as they apply to the container control. See OWNERITEM for the default field values.

hwnd (HWND)
Handle of the window in which ownerdraw will occur. The following is a list of the window handles that can be specified for ownerdraw:

- The container window handle of the icon, name, text, and tree views
- The container title window handle
- The left or right window handles of the details view
- The left or right column heading windows of the details view.

hps (HPS)
Handle of the presentation space of the container window. For the details view that uses a split bar, the presentation space handle is either for the left or right window, depending upon the position of the column. If the details view does not have a split bar, the presentation space handle is for the left window.

fsState (ULONG)
Specifies emphasis flags. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

fsAttribute (ULONG)
Attributes of the record as given in the *flRecordAttr* field in the RECORDCORE data structure.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages. See "RECORDCORE" on page 8-114 and "MINIRECORDCORE" on page 8-104 for descriptions of these data structures.

fsStateOld (ULONG)
Previous emphasis. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

fsAttributeOld (ULONG)
Previous attribute. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

rclItem (RECTL)
This is the bounding rectangle into which the container item is drawn.

If the container item is an icon/text or bit-map/text pair, two WM_DRAWITEM messages are sent to the application. The first WM_DRAWITEM message contains the rectangle bounding the icon or bit map and the second contains the rectangle bounding the text.

If the container item contains only text, or only an icon or bit map, only one WM_DRAWITEM message is sent. However, if the current view is the tree icon or tree text view and if the item is a parent item, the application will receive an additional WM_DRAWITEM (in Container Controls) message. The additional message is for the icon or bit map that indicates whether the parent item is expanded or collapsed.

If the current view is the details view and the CFA_OWNER attribute is set, the rectangle's size is equal to the width of the column and the height of the tallest field in the container item. CFA_OWNER is an attribute of the FIELDINFO data structure's *flData* field.

idItem (ULONG)
Identifies the item being drawn. It can be one of the following:

- CMA_CNRTITLE
- CMA_ICON
- CMA_TEXT
- CMA_TREEICON.

This field is not used for the details view and is set to 0.

hItem (CNRDRAWITEMINFO)
Pointer to a CNRDRAWITEMINFO structure. This field is set to NULL if *idItem* is CMA_CNRTITLE.

See "CNRDRAWITEMINFO" on page 8-88 for descriptions of this structure's fields.

Returns

rc (BOOL)
Item-drawn indicator.

TRUE	The owner draws the item, and so the container control does not draw it.
FALSE	If the owner does not draw the item, the owner returns this value and the container control draws the item.

Related Data Structures

This section covers the data structures that are related to container controls.

CDATE

Structure that contains date information for a data element in the details view of a container control.

Syntax

```
typedef struct _CDATE {
UCHAR         day;
UCHAR         month;
USHORT        year;
  } CDATE;

typedef CDATE *PCDATE;
```

Fields

day (UCHAR)
Current day.

month (UCHAR)
Current month.

year (USHORT)
Current year.

CNRDRAGINFO

Structure that contains information about a direct manipulation event that is occurring over the container. The information specified for this structure depends on the container notification code with which it is used. The differences are specified in the following field descriptions. The applicable notification codes are:

- CN_DRAGAFTER
- CN_DRAGLEAVE
- CN_DRAGOVER
- CN_DROP
- CN_DROPHELP

Syntax

```
typedef struct _CNRDRAGINFO {
PDRAGINFO        pDragInfo;
PRECORDCORE      pRecord;
 } CNRDRAGINFO;

typedef CNRDRAGINFO *PCNRDRAGINFO;
```

Fields

pDragInfo (PDRAGINFO)
Pointer to a DRAGINFO structure.

pRecord (PRECORDCORE)
Pointer to a RECORDCORE structure.

The structure that is pointed to depends on the notification code being used.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages. For the CN_DRAGAFTER notification code, this field contains a pointer to the RECORDCORE structure after which ordered target emphasis is drawn. If ordered target emphasis is applied above the first record in item order, the CM_FIRST attribute is returned.

For the CN_DRAGLEAVE notification code, this field is NULL.

For the CN_DRAGOVER, CN_DROP, and CN_DROPHELP notification codes, this field contains a pointer to a container record over which direct manipulation occurred. This field has a value of NULL if the direct manipulation event occurs over white space.

CNRDRAGINIT

Structure that contains information about a direct manipulation event that is initiated in a container. This structure is used with the CN_INITDRAG notification code only. See CN_INITDRAG for information about that notification code.

Syntax

```
typedef struct _CNRDRAGINIT {
HWND            hwndCnr;
PRECORDCORE     pRecord;
LONG            x;
LONG            y;
LONG            cx;
LONG            cy;
 } CNRDRAGINIT;

typedef CNRDRAGINIT *PCNRDRAGINIT;
```

Fields

hwndCnr (HWND)
Container control handle.

pRecord (PRECORDCORE)
Pointer to the RECORDCORE where direct manipulation started.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

The *pRecord* field can have one of the following values:

NULL Direct manipulation started over white space.
Other Container record over which direct manipulation started.

x (LONG)
X-coordinate of the pointer of the pointing device in desktop coordinates.

y (LONG)
Y-coordinate of the pointer of the pointing device in desktop coordinates.

cx (LONG)
X-offset from the hot spot of the pointer of the pointing device (in pels) to the record origin.

cy (LONG)
Y-offset from the hot spot of the pointer of the pointing device (in pels) to the record origin.

CNRDRAWITEMINFO

Structure that contains information about the container item being drawn. This structure is used with the WM_DRAWITEM (in Container Controls) message only. See "WM_DRAWITEM (in Container Controls)" on page 8-82 for information about that message.

Syntax

```
typedef struct _CNRDRAWITEMINFO {
PRECORDCORE       pRecord;
PFIELDINFO        pFieldInfo;
 } CNRDRAWITEMINFO;

typedef CNRDRAWITEMINFO *PCNRDRAWITEMINFO;
```

Fields

pRecord (PRECORDCORE)

Pointer to the RECORDCORE structure for the record being drawn.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

pFieldInfo (PFIELDINFO)

Pointer to the FIELDINFO structure for the container column being drawn in the details view.

For all other views, this field is NULL.

CNREDITDATA

Structure that contains information about the direct editing of container text. The information specified for this structure depends on the container notification code or message with which it is used. The differences are specified in the following field descriptions. The applicable notification codes and message are:

- CN_BEGINEDIT
- CN_ENDEDIT
- CN_REALLOCPSZ
- "CM_OPENEDIT" on page 8-61

Syntax

```
typedef struct _CNREDITDATA {
ULONG               cb;
HWND                hwndCnr;
PRECORDCORE         pRecord;
PFIELDINFO          pFieldInfo;
PSZ                *ppszText;
ULONG               cbText;
ULONG               id;
 } CNREDITDATA;

typedef CNREDITDATA *PCNREDITDATA;
```

Fields

cb (ULONG)
Structure size.

The size (in bytes) of the CNREDITDATA data structure.

hwndCnr (HWND)
Container window handle.

pRecord (PRECORDCORE)
Pointer to a RECORDCORE data structure, or NULL.

This field is NULL if container titles are to be edited.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

For the CN_BEGINEDIT, CN_ENDEDIT, and CN_REALLOCPSZ notification codes, this field is a pointer to the edited RECORDCORE data structure.

For the CM_OPENEDIT message, this field is a pointer to the RECORDCORE data structure to be edited.

pFieldInfo (PFIELDINFO)
Pointer to a FIELDINFO data structure, or NULL.

Pointer to a FIELDINFO data structure if the current view is the details view and the user is not editing the container title. Otherwise, this field is NULL.

If the current view is the details view:

- For the CN_BEGINEDIT, CN_ENDEDIT, and CN_REALLOCPSZ notification codes, this field contains a pointer to the FIELDINFO structure being edited.
- For the CM_OPENEDIT message, this field is a pointer to the FIELDINFO data structure to be edited.

ppszText (PSZ *)
Pointer to a PSZ text string.

For the CN_BEGINEDIT and CN_REALLOCPSZ notification codes, this field is a pointer to the current PSZ text string.

For the CN_ENDEDIT notification code, this field is a pointer to the new PSZ text string.

For the CM_OPENEDIT message, this field is NULL.

cbText (ULONG)
Number of bytes in the text string.

For the CN_BEGINEDIT notification code, this field is 0.

For the CN_ENDEDIT and CN_REALLOCPSZ notification codes, this field is the number of bytes in the new text string.

For the CM_OPENEDIT message, this field is 0.

id (ULONG)
ID of the window to be edited.

The ID can be one of the following:

CID_CNRTITLEWND
Title window.
CID_LEFTDVWND
Left details view window; default if unsplit window.
CID_RIGHTDVWND
Right details view window.
CID_LEFTCOLTITLEWND
Left details view column headings window; default if unsplit window.
CID_RIGHTCOLTITLEWND
Right details view column headings window.
An application-defined container-ID
Container window.

CNRINFO

Structure that contains information about the container.

Syntax

```
typedef struct _CNRINFO {
ULONG          cb;
PVOID          pSortRecord;
PFIELDINFO     pFieldInfoLast;
PFIELDINFO     pFieldInfoObject;
PSZ            pszCnrTitle;
ULONG          flWindowAttr;
POINTL         ptlOrigin;
ULONG          cDelta;
ULONG          cRecords;
SIZEL          slBitmapOrIcon;
SIZEL          slTreeBitmapOrIcon;
HBITMAP        hbmExpanded;
HBITMAP        hbmCollapsed;
HPOINTER       hptrExpanded;
HPOINTER       hptrCollapsed;
LONG           cyLineSpacing;
LONG           cxTreeIndent;
LONG           cxTreeLine;
ULONG          cFields;
LONG           xVertSplitbar;
 } CNRINFO;

typedef CNRINFO *PCNRINFO;
```

Fields

cb (ULONG)
Structure size.

The size (in bytes) of the CNRINFO data structure.

pSortRecord (PVOID)
Pointer to the comparison function for sorting container records, or NULL.

If NULL, which is the default condition, no sorting is performed. Sorting only occurs during record insertion and when changing the value of this field. The third parameter of the comparison function, *pStorage*, must be NULL. See "CM_SORTRECORD" in the *Control Program Programming Reference* for a further description of the comparison function.

pFieldInfoLast (PFIELDINFO)
Pointer to last column in the left window of the split details view, or NULL.

The default is NULL, causing all columns to be positioned in the left window.

pFieldInfoObject (PFIELDINFO)
Pointer to a column that represents an object in the details view.

The data for this FIELDINFO structure must contain icons or bit maps. In-use emphasis is applied to this column of icons or bit maps only. The default is the leftmost column in the unsplit details view, or the leftmost column in the left window of the split details view.

pszCnrTitle (PSZ)
Title text, or NULL.

Text for the container title. The default is NULL.

flWindowAttr (ULONG)
Window attributes.

Consists of the following container window attributes:

- Specify one of the following container views, which determine the presentation format of items in a container:

  CV_ICON
  In the icon view, the container items are represented as icon/text or bit-map/text pairs, with text beneath the icons or bit maps. This is the default view. This view can be combined with the CV_MINI style bit by using an OR operator (|). See CV_MINI on page 8-94 for more information.

  CV_NAME
  In the name view, the container items are represented as icon/text or bit-map/text pairs, with text to the right of the icons or bit maps. This view can be combined with the CV_MINI and CV_FLOW style bits by using OR operators (|). See CV_MINI on page 8-94 and CV_FLOW on page 8-94 for more information.

  CV_TEXT
  In the text view, the container items are displayed as a list of text strings. This view can be combined with the CV_FLOW style bit by using an OR operator (|). See CV_FLOW on page 8-94 for more information.

CV_TREE

In the tree view, the container items are represented in a hierarchical manner. The tree view has three forms, which are defined in the following list. If you specify CV_TREE by itself, the tree icon view is used.

- Tree icon view

  The tree icon view is specified by using a logical OR operator to combine the tree view with the icon view (CV_TREE | CV_ICON). Container items in this view are represented as icon/text pairs or bit-map/text pairs, with text to the right of the icons or bit maps. Also, a collapsed or expanded icon or bit map is displayed to the left of parent items. If this icon or bit map is a *collapsed* icon or bit map, selecting it will cause the parent item to be expanded so that its child items are displayed below it. If this icon or bit map is an *expanded* icon or bit map, selecting it will cause the parent's child items to be removed from the display. The default collapsed and expanded bit maps provided by the container use a plus sign (+) and a minus sign (-), respectively, to indicate that items can be added to or subtracted from the display.

- Tree name view

  The tree name view is specified by using a logical OR operator to combine the tree view with the name view (CV_TREE | CV_NAME). Container items in this view are displayed as either icon/text pairs or bit-map/text pairs, with text to the right of the icons or bit maps. However, the indicator that represents whether an item can be collapsed or expanded, such as a plus or minus sign, is included in the icon or bit map that represents that item, not in a separate icon or bit map as in the tree icon and tree text views. The container control does not provide default collapsed and expanded bit maps for the tree name view.

- Tree text view

  The tree text view is specified by using a logical OR operator to combine the tree view with the text view (CV_TREE | CV_TEXT). Container items in this view are displayed as a list of text strings. As in the tree icon view, a collapsed or expanded icon or bit map is displayed to the left of parent items.

CV_DETAIL
In the details view, the container items are presented in columns. Each column can contain icons or bit maps, text, numbers, dates, or times.

- Specify one or both of the following view styles by using an OR operator (|) to combine them with the specified view. These view styles are optional.

CV_MINI
Produces a mini-icon whose size is based on the Presentation Manager (PM) SV_CYMENU system value to produce a device-dependent mini-icon.

The CV_MINI view style bit is ignored when:

- The text view (CV_TEXT), tree view (CV_TREE), or details view (CV_DETAIL) are displayed
- The CCS_MINIRECORDCORE style bit is specified.

If this style bit is not specified and the icon view (CV_ICON) or name view (CV_NAME) is used, the default, regular-sized icon is used. The size of regular-sized icons is based on the value in the *slBitmapOrIcon* field of the CNRINFO data structure. If this field is equal to 0, the PM SV_CXICON and SV_CYICON system values for width and height, respectively, are used. Icon sizes are consistent with PM-defined icon sizes for all devices.

CV_FLOW
Dynamically arranges container items in columns in the name and text views. These are called flowed name and flowed text views. If this style bit is set for the name view (CV_NAME) or text view (CV_TEXT), the container items are placed in a single column until the bottom of the client area is reached. The next container item is placed in the adjacent column to the right of the filled column. This process is repeated until all of the container items are positioned in the container. The width of each column is determined by the longest text string in that column. The size of the window determines the depth of the client area.

If this style bit is not specified, the default condition for the name and text views is to vertically fill the container in a single column without flowing the container items. If this style bit is set for the icon view (CV_ICON) or details view (CV_DETAIL), it is ignored.

- Specify either of the following to indicate whether the container will display icons or bit maps:

  CA_DRAWICON
  Icons are used for the icon, name, tree, or details views. This is the default. This container attribute should be used with the *hptrIcon* and *hptrMiniIcon* fields of the RECORDCORE data structure.

  CA_DRAWBITMAP
  Bit maps are used for the icon, name, tree, or details views. This container attribute can be used with the *hbmBitmap* and *hbmMiniBitmap* fields of the RECORDCORE data structure.

  Notes:

  1. If both the CA_DRAWICON and CA_DRAWBITMAP attributes are specified, the CA_DRAWICON attribute is used.
  2. If the CCS_MINIRECORDCORE style bit is specified when a container is created, the *hptrIcon* field of the MINIRECORDCORE data structure is used.

- Specify one of the following attributes to provide target emphasis for the name, text, and details views. If neither ordered nor mixed target emphasis is specified, the emphasis is drawn around the record.

  CA_ORDEREDTARGETEMPH
  Shows where a container record can be dropped during direct manipulation by drawing a line beneath the record. Ordered target emphasis does not apply to the icon and tree views.

  CA_MIXEDTARGETEMPH
  Shows where a container record can be dropped during direct manipulation either by drawing a line between two items or by drawing lines around the container record. Mixed target emphasis does not apply to the icon and tree views.

- Specify the following attribute to draw lines that show the relationship between items in the tree view.

  CA_TREELINE
  Shows the relationship between all items in the tree view.

- Specify the following to draw container records, paint the background of the container, or both:

  CA_OWNERDRAW
  Ownerdraw for the container, which allows the application to draw container records.

  CA_OWNERPAINTBACKGROUND
  Allows the application to subclass the container and paint the background. If specified, and the container is subclassed, the application receives the CM_PAINTBACKGROUND message in the subclass procedure. Otherwise, the container paints the background using the color specified by SYSCLR_WINDOW, which can be changed by using the PP_BACKGROUNDCOLOR or PP_BACKGROUNDCOLORINDEX presentation parameter in the WM_PRESPARAMCHANGED (in Container Controls)

- Specify the following if the container is to have a title:

  CA_CONTAINERTITLE
  Allows you to include a container title. The default is no container title.

- Specify one or both of the following container title attributes. These are valid only if the CA_CONTAINERTITLE attribute is specified.

  CA_TITLEREADONLY
  Prevents the container title from being edited directly. The default is to allow the container title to be edited.

  CA_TITLESEPARATOR
  Puts a separator line between the container title and the records beneath it. The default is no separator line.

- Specify one of the following to position the container title. These are valid only if the CA_CONTAINERTITLE attribute is specified.

  CA_TITLECENTER
  Centers the container title. This is the default.

  CA_TITLELEFT
  Left-justifies the container title.

  CA_TITLERIGHT
  Right-justifies the container title.

- Specify the following to display column headings in the details view:

  CA_DETAILSVIEWTITLES
  Allows you to include column headings in the details view. The default is no column headings.

ptlOrigin (POINTL)
Workspace origin.

Lower-left origin of the workspace in virtual coordinates, used in the icon view. The default origin is **(0,0)**.

cDelta (ULONG)
Threshold.

An application-defined threshold, or number of records, from either end of the list of available records. Used when a container needs to handle large amounts of data. The default is 0. Refer to the *OS/2 Programming Guide* for more information about specifying deltas.

cRecords (ULONG)
Number of records.

The number of records in the container. Initially this field is 0.

slBitmapOrIcon (SIZEL)
Icon/bit-map size.

The size (in pels) of icons or bit maps. The default is the system size.

slTreeBitmapOrIcon (SIZEL)
Icon/bit-map size.

The size (in pels) of the expanded and collapsed icons or bit maps used in the tree icon and tree text views.

hbmExpanded (HBITMAP)
Bit-map handle.

The handle of the bit map to be used to represent an expanded parent item in the tree icon and tree text views. If neither an icon handle (see *hptrExpanded*) nor a bit-map handle is specified, a default bit map with a minus sign (–) is provided.

hbmCollapsed (HBITMAP)
Bit-map handle.

The handle of the bit map to be used to represent a collapsed parent item in the tree icon and tree text views. If neither an icon handle (see *hptrCollapsed*) nor a bit-map handle is specified, a default bit map with a plus sign (+) is provided.

hptrExpanded (HPOINTER)
Icon handle.

The handle of the icon to be used to represent an expanded parent item in the tree icon and tree text views. If neither an icon handle nor a bit-map handle (see *hbmExpanded*) is specified, a default bit map with a minus sign (-) is provided.

hptrCollapsed (HPOINTER)
Icon handle.

The handle of the icon to be used to represent a collapsed parent item in the tree icon and tree text views. If neither an icon handle nor a bit-map handle (see *hbmCollapsed*) is specified, a default bit map with a plus sign (+) is provided.

cyLineSpacing (LONG)
Vertical space.

The amount of vertical space (in pels) between the records. If you specify a value that is less than 0, a default value is used.

cxTreeIndent (LONG)
Horizontal space.

The amount of horizontal space (in pels) between levels in the tree view. If you specify a value that is less than 0, a default value is used.

cxTreeLine (LONG)
Line width.

The width of the lines (in pels) that show the relationship between tree items. If you specify a value that is less than 0, a default value is used. Also, if the CA_TREELINE container attribute of the *flWindowAttr* field is not specified, these lines are not drawn.

cFields (ULONG)
Number of columns.

The number of FIELDINFO structures in the container. Initially this field is 0.

xVertSplitbar (LONG)
Split bar position.

The initial position of the split bar relative to the container, used in the details view. If this value is less than 0, the split bar is not used. The default value is negative one (–1).

CNRLAZYDRAGINFO

Container lazy drag information.

Syntax

```
typedef struct _CNRLAZYDRAGINFO {
PDRAGINFO         pDragInfo;
PRECORDCORE       pRecord;
HWND              hwndTarget;
 } CNRLAZYDRAGINFO;

typedef CNRLAZYDRAGINFO *PCNRLAZYDRAGINFO;
```

Fields

pDragInfo (PDRAGINFO)
Pointer to the DRAGINFO structure.

pRecord (PRECORDCORE)
Pointer to a container RECORDCORE structure.

A value of NULL indicates that the lazy drag set was dropped over whitespace in the container. Any other value indicates that the lazy drag set was dropped on the record specified by this field.

hwndTarget (HWND)
Handle of the target winddow that the lazy drag set was dropped on.

CTIME

Structure that contains time information for a data element in the details view of a container control.

Syntax

```
typedef struct _CTIME {
UCHAR      hours;
UCHAR      minutes;
UCHAR      seconds;
UCHAR      ucReserved;
 } CTIME;

typedef CTIME *PCTIME;
```

Fields

hours (UCHAR)
Current hour.

minutes (UCHAR)
Current minute.

seconds (UCHAR)
Current second.

ucReserved (UCHAR)
Reserved.

FIELDINFO

Structure that contains information about column data in the details view of the container control. The details view displays each FIELDINFO structure as a column of data that contains specific information about each container record. For example, one FIELDINFO structure, or column, might contain icons or bit maps that represent each container record. Another FIELDINFO structure might contain the date or time that each container record was created.

Syntax

```
typedef struct _FIELDINFO {
ULONG                   cb;
ULONG                   flData;
ULONG                   flTitle;
PVOID                   pTitleData;
ULONG                   offStruct;
PVOID                   pUserData;
struct _FIELDINFO      *pNextFieldInfo;
ULONG                   cxWidth;
 } FIELDINFO;

typedef FIELDINFO *PFIELDINFO;
```

Fields

cb (ULONG)
Structure size.

The size (in bytes) of the FIELDINFO structure.

flData (ULONG)
Data attributes.

Attributes of the data in a field.

- Specify one of the following for each column to choose the type of data that is displayed in each column:

  CFA_BITMAPORICON
  The column contains bit-map or icon data.

  CFA_DATE
  The data in the column is displayed in date format. National Language Support (NLS) is enabled for date format. Use the data structure described in CDATE

  CFA_STRING
  Character or text data is displayed in this column.

  CFA_TIME
  The data in the column is displayed in time format. National Language Support (NLS) is enabled for time format. Use the data structure described in CTIME.

CFA_ULONG
Unsigned number data is displayed in this column. National Language Support (NLS) is enabled for number format.

- Specify any or all of the following column attributes:

CFA_FIREADONLY
Prevents text in a FIELDINFO data structure (text in a column) from being edited directly. This attribute applies only to columns for which the CFA_STRING attribute has been specified.

CFA_HORZSEPARATOR
A horizontal separator is provided beneath column headings.

CFA_INVISIBLE
Invisible container column. The default is visible.

CFA_OWNER
Ownerdraw is enabled for this container column.

CFA_SEPARATOR
A vertical separator is drawn after this column.

- Specify one of the following for each column to vertically position data in that column:

CFA_BOTTOM
Bottom-justifies field data.

CFA_TOP
Top-justifies field data.

CFA_VCENTER
Vertically centers field data. This is the default.

- Specify one of the following for each column to horizontally position data in that column. These attributes can be combined with the attributes used for vertical positioning of column data by using an OR operator (|).

CFA_CENTER
Horizontally centers field data.

CFA_LEFT
Left-justifies field data. This is the default.

CFA_RIGHT
Right-justifies field data.

flTitle (ULONG)
Column heading attributes.

- Specify the following if icon or bit-map data is to be displayed in the column heading:

  CFA_BITMAPORICON
  The column heading contains icon or bit-map data. If CFA_BITMAPORICON is not specified, any data that is assigned to a column heading is assumed to be character or text data.

- Specify the following to prevent direct editing of a column heading:

  CFA_FITITLEREADONLY
  Prevents a column heading from being edited directly.

- Specify one of the following for each column heading to vertically position data in that column heading:

  CFA_TOP
  Top-justifies column headings.

  CFA_BOTTOM
  Bottom-justifies column headings.

  CFA_VCENTER
  Vertically centers column headings. This is the default.

- Specify one of the following for each column heading to horizontally position data in that column heading. These attributes can be combined with the attributes used for vertical positioning of column heading data by using an OR operator (|).

  CFA_CENTER
  Horizontally centers column headings.

  CFA_LEFT
  Left-justifies column headings. This is the default.

  CFA_RIGHT
  Right-justifies column headings.

pTitleData (PVOID)
Column heading data.

Column heading data, which can be a text string, or an icon or bit map. The default is a text string. If the *flTitle* field is set to the CFA_BITMAPORICON attribute, this must be an icon or bit map.

offStruct (ULONG)
Structure offset.

Offset from the beginning of a RECORDCORE structure to the data that is displayed in this column.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

pUserData (PVOID)
Pointer to user data.

pNextFieldInfo (struct _FIELDINFO *)
Pointer to the next linked FIELDINFO data structure.

cxWidth (ULONG)
Column width.

Used to specify the width of a column. The default is an automatically sized column that is always the width of its widest element. If this field is set and the data is too wide, the data is truncated.

FIELDINFOINSERT

Structure that contains information about the FIELDINFO structure or structures that are being inserted into a container. This structure is used in the CM_INSERTDETAILFIELDINFO container message only. See "CM_INSERTDETAILFIELDINFO" on page 8-57 for information about that message.

Syntax

```
typedef struct _FIELDINFOINSERT {
ULONG           cb;
PFIELDINFO      pFieldInfoOrder;
ULONG           fInvalidateFieldInfo;
ULONG           cFieldInfoInsert;
 } FIELDINFOINSERT;

typedef FIELDINFOINSERT *PFIELDINFOINSERT;
```

Fields

cb (ULONG)
Structure size.

The size (in bytes) of the FIELDINFOINSERT structure.

pFieldInfoOrder (PFIELDINFO)
Column order.

Orders the FIELDINFO structure or structures relative to other FIELDINFO structures in the container. The values can be:

CMA_FIRST	Places a FIELDINFO structure, or list of FIELDINFO structures, at the front of the list of columns.
CMA_END	Places a FIELDINFO structure, or list of FIELDINFO structures, at the end of the list of columns.
Other	Pointer to a FIELDINFO structure that this structure, or list of structures, is to be inserted after.

fInvalidateFieldInfo (ULONG)
Update flag.

Flag that indicates an automatic display update after the FIELDINFO structures are inserted.

TRUE	The display is automatically updated after FIELDINFO structures are inserted.
FALSE	The application must send the CM_INVALIDATEDETAILFIELDINFO message after the FIELDINFO structures are inserted.

cFieldInfoInsert (ULONG)
Number of columns.

The number of FIELDINFO structures to be inserted. The *cFieldInfoInsert* field value must be greater than 0.

MINIRECORDCORE

Structure that contains information for smaller records than those defined by the RECORDCORE data structure. This data structure is used if the CCS_MINIRECORDCORE style bit is specified when a container is created.

Syntax

```
typedef struct _MINIRECORDCORE {
ULONG                        cb;
ULONG                        flRecordAttr;
POINTL                       ptlIcon;
struct _MINIRECORDCORE      *preccNextRecord;
PSZ                          pszIcon;
HPOINTER                     hptrIcon;
 } MINIRECORDCORE;

typedef MINIRECORDCORE *PMINIRECORDCORE;
```

Fields

cb (ULONG)
Structure size.

The size (in bytes) of the MINIRECORDCORE structure.

flRecordAttr (ULONG)
Attributes of container records.

Contains any or all of the following:

CRA_COLLAPSED	Specifies that a record is collapsed.
CRA_CURSORED	Specifies that a record will be drawn with a selection cursor.
CRA_DROPONABLE	Specifies that a record can be a target for direct manipulation.
CRA_EXPANDED	Specifies that a record is expanded.
CRA_FILTERED	Specifies that a record is filtered, and therefore hidden from view.
CRA_INUSE	Specifies that a record will be drawn with in-use emphasis.
CRA_RECORDREADONLY	Prevents a record from being edited directly.
CRA_SELECTED	Specifies that a record will be drawn with selected-state emphasis.
CRA_TARGET	Specifies that a record will be drawn with target emphasis.

ptlIcon (POINTL)
Record position.

Position of a container record in the icon view.

preccNextRecord (struct _MINIRECORDCORE *)
Pointer to the next linked record.

pszIcon (PSZ)
Record text.

Text for the container record.

hptrIcon (HPOINTER)
Record icon.

Icon that is displayed for the container record.

NOTIFYDELTA

Structure that contains information about the placement of delta information for a container. This structure is used in the CN_QUERYDELTA container notification code only. See CN_QUERYDELTA for information about that notification code.

Syntax

```
typedef struct _NOTIFYDELTA {
HWND        hwndCnr;
ULONG       fDelta;
 } NOTIFYDELTA;

typedef NOTIFYDELTA *PNOTIFYDELTA;
```

Fields

hwndCnr (HWND)
Container control handle.

fDelta (ULONG)
Placement of delta information. The values can be:

CMA_DELTATOP	The record that represents the delta value scrolls into view at the top of the client area.
CMA_DELTABOT	The record that represents the delta value scrolls into view at the bottom of the client area.
CMA_DELTAHOME	The container scrolls to the beginning of the list of all container records that are available to be inserted into the container, such as the first record in a database.
CMA_DELTAEND	The container scrolls to the end of the list of all container records that are available to be inserted into the container, such as the last record in a database.

NOTIFYRECORDEMPHASIS

Structure that contains information about emphasis that is being applied to a container record. This structure is used in the CN_EMPHASIS container notification code only. See CN_EMPHASIS for information about that notification code.

Syntax

```
typedef struct _NOTIFYRECORDEMPHASIS {
HWND               hwndCnr;
PRECORDCORE        pRecord;
ULONG              fEmphasisMask;
  } NOTIFYRECORDEMPHASIS;

typedef NOTIFYRECORDEMPHASIS *PNOTIFYRECORDEMPHASIS;
```

Fields

hwndCnr (HWND)
Container control handle.

pRecord (PRECORDCORE)
Pointer to a RECORDCORE data structure whose emphasis attribute has been changed.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

fEmphasisMask (ULONG)
Changed emphasis attributes.

Specifies the emphasis attribute or attributes that changed in the container record. The following states can be combined with a logical OR operator (|):

- CRA_CURSORED
- CRA_INUSE
- CRA_SELECTED.

NOTIFYRECORDENTER

Structure that contains information about the input device that is being used with the container control. This structure is used in the CN_ENTER container notification code only. See CN_ENTER for information about that notification code.

Syntax

```
typedef struct _NOTIFYRECORDENTER {
HWND            hwndCnr;
ULONG           fKey;
PRECORDCORE     pRecord;
 } NOTIFYRECORDENTER;

typedef NOTIFYRECORDENTER *PNOTIFYRECORDENTER;
```

Fields

hwndCnr (HWND)
Container control handle.

fKey (ULONG)
Flag.

Flag that determines whether the Enter key was pressed or the select button was double-clicked.

TRUE	The Enter key was pressed.
FALSE	The select button was double-clicked.

pRecord (PRECORDCORE)
Pointer to the RECORDCORE data structure over which an action occurred.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

- If a user presses the Enter key, a pointer to the record with the selection cursor is returned.
- If a user double-clicks the select button when the pointer of the pointing device is over a record, a pointer to the record is returned.
- If a user double-clicks the select button when the pointer of the pointing device is over white space, NULL is returned.

NOTIFYSCROLL

Structure that contains information about scrolling a container control window. This structure is used in the CN_SCROLL container notification code only. See CN_SCROLL for information about that notification code.

Syntax

```
typedef struct _NOTIFYSCROLL {
HWND        hwndCnr;
LONG        lScrollInc;
ULONG       fScroll;
 } NOTIFYSCROLL;

typedef NOTIFYSCROLL *PNOTIFYSCROLL;
```

Fields

hwndCnr (HWND)
Container control handle.

lScrollInc (LONG)
Scroll amount.

Amount (in pixels) by which the window scrolled.

fScroll (ULONG)
Scroll flags.

Flags that show the direction in which the window scrolled and the window that was scrolled.

CMA_HORIZONTAL	A window was scrolled horizontally. If the split details view window is scrolled, a logical OR operator (|) is used to combine the CMA_HORIZONTAL attribute with either the CMA_LEFT attribute or the CMA_RIGHT attribute to indicate which window was scrolled. If the unsplit details view window is scrolled, the CMA_HORIZONTAL attribute is combined with the CMA_LEFT attribute.
CMA_VERTICAL	The container window scrolled vertically. If the split details view window is scrolled, a logical OR operator (|) is used to combine the CMA_VERTICAL attribute with the CMA_LEFT attribute and the CMA_RIGHT attribute. If the unsplit details view window is scrolled, the CMA_VERTICAL attribute is combined with the CMA_LEFT attribute.

OWNERBACKGROUND

Structure that contains information about painting the container window's background by the container owner. This structure is used in the CM_PAINTBACKGROUND container message only. See "CM_PAINTBACKGROUND" on page 8-62 for information about that message.

Syntax

```
typedef struct _OWNERBACKGROUND {
HWND        hwnd;
HPS         hps;
RECTL       rclBackground;
LONG        idWindow;
 } OWNERBACKGROUND;

typedef OWNERBACKGROUND *POWNERBACKGROUND;
```

Fields

hwnd (HWND)
Window handle.

Handle of the window to be painted.

hps (HPS)
Presentation-space handle.

rclBackground (RECTL)
Background rectangle.

Background rectangle in window coordinates.

idWindow (LONG)
Window ID.

Identity of the window to be painted.

OWNERITEM

Owner item.

Syntax

```
typedef struct _OWNERITEM {
HWND         hwnd;
HPS          hps;
ULONG        fsState;
ULONG        fsAttribute;
ULONG        fsStateOld;
ULONG        fsAttributeOld;
RECTL        rclItem;
LONG         idItem;
ULONG        hItem;
 } OWNERITEM;

typedef OWNERITEM *POWNERITEM;
```

Fields

hwnd (HWND)
Window handle.

hps (HPS)
Presentation-space handle.

fsState (ULONG)
State.

fsAttribute (ULONG)
Attribute.

fsStateOld (ULONG)
Old state.

fsAttributeOld (ULONG)
Old attribute.

rclItem (RECTL)
Item rectangle.

idItem (LONG)
Item identity.

hItem (ULONG)
Item.

QUERYRECFROMRECT

Structure that contains information about a container record that is bounded by a specified rectangle. This structure is used in the CM_QUERYRECORDFROMRECT container message only. See "CM_QUERYRECORDFROMRECT" on page 8-68 for information about that message.

Syntax

```
typedef struct _QUERYRECFROMRECT {
ULONG       cb;
RECTL       rect;
ULONG       fsSearch;
 } QUERYRECFROMRECT;

typedef QUERYRECFROMRECT *PQUERYRECFROMRECT;
```

Fields

cb (ULONG)
Structure size.

The size (in bytes) of the QUERYRECFROMRECT data structure.

rect (RECTL)
Rectangle.

The rectangle to query, in virtual coordinates relative to the container window origin. If the details view (CV_DETAIL) is displayed, the x-coordinates of the rectangle are ignored.

fsSearch (ULONG)
Search control flags.

One flag from each of the following groups can be specified:

- Search sensitivity:

  CMA_COMPLETE
  Returns the container records that are completely within the bounding rectangle.

  CMA_PARTIAL
  Returns the container records that are completely or partially within the bounding rectangle.

- Enumeration order:

  CMA_ITEMORDER
  Container records are enumerated in item order, lowest to highest.

  CMA_ZORDER
  Container records are enumerated by z-order, from top to bottom. This flag is valid for the icon view only.

QUERYRECORDRECT

Structure that contains information about the rectangle of the specified container record, relative to the container window origin. This structure is used in the CM_QUERYRECORDRECT container message only. See "CM_QUERYRECORDRECT" on page 8-69 for information about that message.

Syntax

```
typedef struct _QUERYRECORDRECT {
ULONG             cb;
PRECORDCORE       pRecord;
ULONG             fRightSplitWindow;
ULONG             fsExtent;
 } QUERYRECORDRECT;

typedef QUERYRECORDRECT *PQUERYRECORDRECT;
```

Fields

cb (ULONG)
Structure size.

The size (in bytes) of the QUERYRECORDRECT structure.

pRecord (PRECORDCORE)
Pointer.

Pointer to the specified RECORDCORE data structure.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

fRightSplitWindow (ULONG)
Window flag.

Flag that specifies the right or left window in the split details view.

This flag is ignored if the view is not the split details view.

TRUE	Right split window is returned.
FALSE	Left split window is returned.

fsExtent (ULONG)
Rectangle flags.

Flags that specify the extent of the desired rectangle.

These flags can be combined by using a logical OR operator (|) to return the rectangle that bounds the icon, the expanded and collapsed icon or bit map, and the text.

CMA_ICON	Returns the icon rectangle.
CMA_TEXT	Returns the text rectangle.
CMA_TREEICON	Returns the rectangle of the expanded and collapsed icons or bit maps. This flag is valid for the tree icon and tree text views only.

RECORDCORE

Structure that contains information for records in a container control. This data structure is used if the CCS_MINIRECORDCORE style bit is not specified when a container is created.

Syntax

```
typedef struct _RECORDCORE {
ULONG                    cb;
ULONG                    flRecordAttr;
POINTL                   ptlIcon;
struct _RECORDCORE      *preccNextRecord;
PSZ                      pszIcon;
HPOINTER                 hptrIcon;
HPOINTER                 hptrMiniIcon;
HBITMAP                  hbmBitmap;
HBITMAP                  hbmMiniBitmap;
PTREEITEMDESC            pTreeItemDesc;
PSZ                      pszText;
PSZ                      pszName;
PSZ                      pszTree;
 } RECORDCORE;

typedef RECORDCORE *PRECORDCORE;
```

Fields

cb (ULONG)
The size, in bytes, of the RECORDCORE structure.

flRecordAttr (ULONG)
Container record attributes.

This parameter can contain any or all of the following:

CRA_COLLAPSED	Specifies that a record is collapsed.
CRA_CURSORED	Specifies that a record will be drawn with a selection cursor.
CRA_DISABLED	Specifies that a record will be drawn with unavailable-state emphasis.
CRA_DROPONABLE	Specifies that a record can be a target for direct manipulation.
CRA_EXPANDED	Specifies that a record is expanded.
CRA_FILTERED	Specifies that a record is filtered and, therefore, hidden from view.
CRA_INUSE	Specifies that a record will be drawn with in-use emphasis.
CRA_PICKED	Specifies that the container record willl be picked up as part of the drag set.
CRA_SELECTED	Specifies that a record will be drawn with selected-state emphasis.
CRA_SOURCE	Specifies that a record will be drawn with source-menu emphasis.

ptlIcon (POINTL)
Position of a container record in the icon view.

preccNextRecord (struct _RECORDCORE *)
Pointer to the next linked record.

pszIcon (PSZ)
Text for the icon view (CV_ICON).

hptrIcon (HPOINTER)
Icon that is displayed when the CV_MINI style bit is not specified.

This field is used when the CA_DRAWICON container attribute of the CNRINFO data structure is set.

hptrMiniIcon (HPOINTER)
Icon that is displayed when the CV_MINI style bit is specified.

This field is used when the CA_DRAWICON container attribute of the CNRINFO data structure is set.

hbmBitmap (HBITMAP)
Bit map displayed when the CV_MINI style bit is not specified.

This field is used when the CA_DRAWBITMAP container attribute of the CNRINFO data structure is set.

hbmMiniBitmap (HBITMAP)
Bit map displayed when the CV_MINI style bit is specified.

This field is used when the CA_DRAWBITMAP container attribute of the CNRINFO data structure is set.

pTreeItemDesc (PTREEITEMDESC)
Pointer to a TREEITEMDESC structure.

The TREEITEMDESC structure contains the icons and bit maps used to represent the state of an expanded or collapsed parent item in the tree name view.

pszText (PSZ)
Text for the text view (CV_TEXT).

pszName (PSZ)
Text for the name view (CV_NAME).

pszTree (PSZ)
Text for the tree view (CV_TREE).

RECORDINSERT

Structure that contains information about the RECORDCORE structure or structures that are being inserted into a container. The RECORDINSERT structure is used in the CM_INSERTRECORD container message only. See "CM_INSERTRECORD" on page 8-58 for information about that message.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

Syntax

```
typedef struct _RECORDINSERT {
ULONG            cb;
PRECORDCORE      pRecordOrder;
PRECORDCORE      pRecordParent;
ULONG            fInvalidateRecord;
ULONG            zOrder;
ULONG            cRecordsInsert;
  } RECORDINSERT;

typedef RECORDINSERT *PRECORDINSERT;
```

Fields

cb (ULONG)
Structure size.

The size (in bytes) of the RECORDINSERT structure.

pRecordOrder (PRECORDCORE)
Record order.

Orders the RECORDCORE structure or structures relative to other RECORDCORE structures in the container. The values can be:

CMA_FIRST	Places a RECORDCORE structure, or list of RECORDCORE structures, at the beginning of the list of structures.
CMA_END	Places a RECORDCORE structure, or list of RECORDCORE structures, at the end of the list of structures.
Other	Pointer to a RECORDCORE structure that this structure, or list of structures, is to be inserted after.

pRecordParent (PRECORDCORE)
Pointer to record parent.

Pointer to a RECORDCORE structure that is the parent of the record or records to be inserted. This field is used only with the CMA_FIRST or CMA_END attributes of the *pRecordOrder* field.

fInvalidateRecord (ULONG)
Update flag.

Flag that indicates an automatic display update after RECORDCORE structures are inserted.

TRUE	The display is automatically updated after a RECORDCORE structure is inserted.
FALSE	The application must send the CM_INVALIDATERECORD message after a RECORDCORE structure is inserted.

zOrder (ULONG)
Record z-order.

Positions the RECORDCORE structure in z-order, relative to other records in the container. The values can be:

CMA_TOP	Places a RECORDCORE structure at the top of the z-order. This is the default value.
CMA_BOTTOM	Places a RECORDCORE structure at the bottom of the z-order.

cRecordsInsert (ULONG)
Number of root level structures.

The number of root level RECORDCORE structures to be inserted. The *cRecordsInsert* field value must be greater than 0.

SEARCHSTRING

Structure that contains information about the container text string that is the object of the search. This structure is used in the CM_SEARCHSTRING container message only. See "CM_SEARCHSTRING" on page 8-74 for information about that message.

Syntax

```
typedef struct _SEARCHSTRING {
ULONG       cb;
PSZ         pszSearch;
ULONG       fsPrefix;
ULONG       fsCaseSensitive;
ULONG       usView;
 } SEARCHSTRING;

typedef SEARCHSTRING *PSEARCHSTRING;
```

Fields

cb (ULONG)
Structure size.

The size (in bytes) of the SEARCHSTRING structure.

pszSearch (PSZ)
Pointer to the search string.

fsPrefix (ULONG)
Search flag.

Search flag that defines the criteria by which the string specified by the *pszSearch* field is to be compared with the text of the container records to determine the pointer to the first matching record.

TRUE	Matching occurs if the leading characters of the container record are the characters specified by the *pszSearch* field.
FALSE	Matching occurs if the container record contains a substring of the characters specified by the *pszSearch* field.

fsCaseSensitive (ULONG)
Case sensitivity flag.

Determines case sensitivity of the search.

TRUE	The search is case sensitive.
FALSE	The search is not case sensitive.

usView (ULONG)
View to search.

Search one of the container views for the string. Valid values are:

- CV_ICON
- CV_NAME
- CV_TEXT
- CV_TREE
- CV_DETAIL.

TREEITEMDESC

Structure that contains icons and bit maps used to represent the state of an expanded or collapsed parent item in the tree name view of a container control.

Syntax

```
typedef struct _TREEITEMDESC {
HBITMAP        hbmExpanded;
HBITMAP        hbmCollapsed;
HPOINTER       hptrExpanded;
HPOINTER       hptrCollapsed;
 } TREEITEMDESC;

typedef TREEITEMDESC *PTREEITEMDESC;
```

Fields

hbmExpanded (HBITMAP)
Expanded bit-map handle.

The handle of the bit map to be used to represent an expanded parent item in the tree name view.

hbmCollapsed (HBITMAP)
Collapsed bit-map handle.

The handle of the bit map to be used to represent a collapsed parent item in the tree name view.

hptrExpanded (HPOINTER)
Expanded icon handle.

The handle of the icon to be used to represent an expanded parent item in the tree name view.

hptrCollapsed (HPOINTER)
Collapsed icon handle.

The handle of the icon to be used to represent a collapsed parent item in the tree name view.

Summary

Following are tables that describe the OS/2 window messages, notification messages, notification codes, and data structures used with container controls:

Table 8-3 (Page 1 of 2). Container Control Window Messages

Message Name	Description
CM_ALLOCDETAILFIELDINFO	Allocates memory for one or more FIELDINFO data structures.
CM_ALLOCRECORD	Allocates memory for one or more RECORDCORE data structures.
CM_ARRANGE	Arranges the container records in the icon view.
CM_CLOSEEDIT	Closes the window containing the multiple-line entry (MLE) field used to edit container text directly.
CM_COLLAPSETREE	Causes one parent item in the tree view to be collapsed.
CM_ERASERECORD	Erases the source record from the current view when a move occurs as a result of direct manipulation.
CM_EXPANDTREE	Causes one parent item in the tree view to be expanded.
CM_FILTER	Filters the contents of a container so that a subset of the container items can be viewed.
CM_FREEDETAILFIELDINFO	Frees the memory associated with one or more FIELDINFO data structures.
CM_FREERECORD	Frees the memory associated with one or more RECORDCORE data structures.
CM_HORZSCROLLSPLITWINDOW	Scrolls a split window in the split details view.
CM_INSERTDETAILFIELDINFO	Inserts one or more FIELDINFO data structures into a container control.
CM_INSERTRECORD	Inserts one or more RECORDCORE data structures into a container control.
CM_INVALIDATEDETAILFIELDINFO	Notifies the container control that any or all FIELDINFO data structures are not valid and that the view must be refreshed.
CM_INVALIDATERECORD	Notifies the container control that any or all RECORDCORE data structures are not valid and must be refreshed.
CM_OPENEDIT	Opens the window that contains the multiple line entry (MLE) field used to edit container text directly.
CM_PAINTBACKGROUND	Informs an application when a container's background is painted if the CA_OWNERPAINTBACKGROUND attribute of the CNRINFO data structure is specified.
CM_QUERYCNRINFO	Returns the container's CNRINFO data structure.
CM_QUERYDETAILFIELDINFO	Returns a pointer to the requested FIELDINFO data structure.

Table 8-3 (Page 2 of 2). Container Control Window Messages

Message Name	Description
CM_QUERYDRAGIMAGE	Returns a handle to the icon or bit map for the record in the current view.
CM_QUERYRECORD	Returns a pointer to the requested RECORDCORE data structure.
CM_QUERYRECORDEMPHASIS	Queries for a container record with the specified emphasis attributes.
CM_QUERYRECORDFROMRECT	Queries for a container record that is bounded by the specified rectangle.
CM_QUERYRECORDINFO	Updates the specified records with the current information for the container.
CM_QUERYRECORDRECT	Returns the rectangle of the specified container record, relative to the container window origin.
CM_QUERYVIEWPORTRECT	Returns a rectangle that contains the coordinates of the container's work area.
CM_REMOVEDETAILFIELDINFO	Removes one, multiple, or all FIELDINFO data structures from the container control.
CM_REMOVERECORD	Removes one, multiple, or all RECORDCORE data structures from the container control.
CM_SCROLLWINDOW	Scrolls an entire container window.
CM_SEARCHSTRING	Returns the pointer to a container record whose text matches the string.
CM_SETCNRINFO	Sets or changes the data for the container control.
CM_SETRECORDEMPHASIS	Sets the emphasis attributes of the specified container record.
CM_SORTRECORD	Sorts the container records in the container control.
WM_PICKUP	Adds objects to the pickup set during a Pickup and Drop operation.
WM_PRESPARAMCHANGED	Sent when a presentation parameter is set or removed dynamically from a window instance.

Table 8-4. Container Control Notification Messages

Message Name	Description
WM_CONTROL	Occurs when the container control has a significant event to notify to its owner.
WM_CONTROLPOINTER	Sent to the container control's owner window when the pointing device pointer moves over the container window, thereby allowing the owner to set the pointing device pointer.
WM_DRAWITEM	Sent to the owner of the container control each time an item is to be drawn.

Table 8-5. Container Control Notification Codes	
Code Name	**Description**
CN_BEGINEDIT	Sent when container text is about to be edited.
CN_COLLAPSETREE	Sent when a parent item is collapsed in the tree view.
CN_CONTEXTMENU	Sent when the container receives a WM_CONTEXTMENU message.
CN_DRAGAFTER	Sent when the container receives a DM_DRAGOVER message.
CN_DRAGLEAVE	Sent when the container receives a DM_DRAGLEAVE message.
CN_DRAGOVER	Sent when the container receives a DM_DRAGOVER message.
CN_DROP	Sent when the container receives a DM_DROP message.
CN_DROPHELP	Sent when the container receives a DM_DROPHELP message.
CN_DROPNOTIFY	Sent when a LazyDrag drag set is dropped over the container.
CN_EMPHASIS	Sent when the attributes of a container record change.
CN_ENDEDIT	Sent when direct editing of the container text ends.
CN_ENTER	Sent either when the Enter key is pressed while the container window has the focus, or when the select button is double-clicked while the pointer is over the container window.
CN_EXPANDTREE	Sent when the container expands a parent item in the tree view.
CN_HELP	Sent when the container receives a WM_HELP message.
CN_INITDRAG	Sent when the drag button is pressed and the pointer is moved while over the container control.
CN_KILLFOCUS	Sent when the container is losing the focus.
CN_PICKUP	Sent when the container receives a WM_PICKUP message. Determines if mouse position is over target object, white space, or desktop.
CN_QUERYDELTA	Sent to query for more data when the user scrolls to a preset delta value.
CN_REALLOCPSZ	Sent when container text is edited (before CN_ENDEDIT is sent).
CN_SCROLL	Sent when the container window scrolls.
CN_SETFOCUS	Sent when the container receives the focus.

Table 8-6 (Page 1 of 2). Container Control Data Structures	
Data Structure Name	**Description**
CDATE	Contains date information for a data element in the details view of the container.
CNRDRAGINFO	Contains information about a direct manipulation event occurring over the container.
CNRDRAGINIT	Contains information about a direct manipulation event that was initiated in a container.
CNRDRAWITEMINFO	Contains information about the item being drawn in the container.
CNREDITDATA	Contains information about the direct editing of container text.
CNRINFO	Contains information about the container.
CNRLAZYDRAGINFO	Contains information about the DRAGINFO, RECORDCORE that is dropped on and the window handle of the target window.
CTIME	Contains time information for a data element in the details view of the container.
FIELDINFO	Contains information about column data in the details view of the container.
FIELDINFOINSERT	Contains information about the FIELDINFO data structures that are being inserted into the container.
MINIRECORDCORE	Contains information for container records that are smaller than those defined by the RECORDCORE data structure.
NOTIFYDELTA	Contains information about the placement of delta information for the container.
NOTIFYRECORDEMPHASIS	Contains information about the emphasis applied to a container record.
NOTIFYRECORDENTER	Contains information about the input device being used with the container.
NOTIFYSCROLL	Contains information about scrolling the container window.
OWNERBACKGROUND	Contains information about painting the container window's background.
OWNERITEM	Contains owner item.
QUERYRECFROMRECT	Contains information about a container record that is bounded by a specified rectangle.
QUERYRECORDRECT	Contains information about the rectangle of the specified container record, relative to the container window origin.
RECORDCORE	Contains information for container records.
RECORDINSERT	Contains information about the RECORDCORE data structures that are being inserted into the container.
SEARCHSTRING	Contains information about the container text string that is the object of the search.

Table 8-6 (Page 2 of 2). Container Control Data Structures

Data Structure Name	Description
TREEITEMDESC	Contains icons and bit maps used to represent the state of an expanded or collapsed parent item in the tree name view.

Chapter 9. Notebook Controls

A notebook control (WC_NOTEBOOK window class) is a visual component that organizes information on individual *pages* so that a user can find and display that information quickly and easily. This chapter explains how to use notebook controls in PM applications.

About Notebook Controls

This notebook control component simulates a real notebook but improves on it by overcoming a notebook's natural limitations. A user can select and display pages by using a pointing device or the keyboard. Figure 9-1 shows an example of a notebook control.

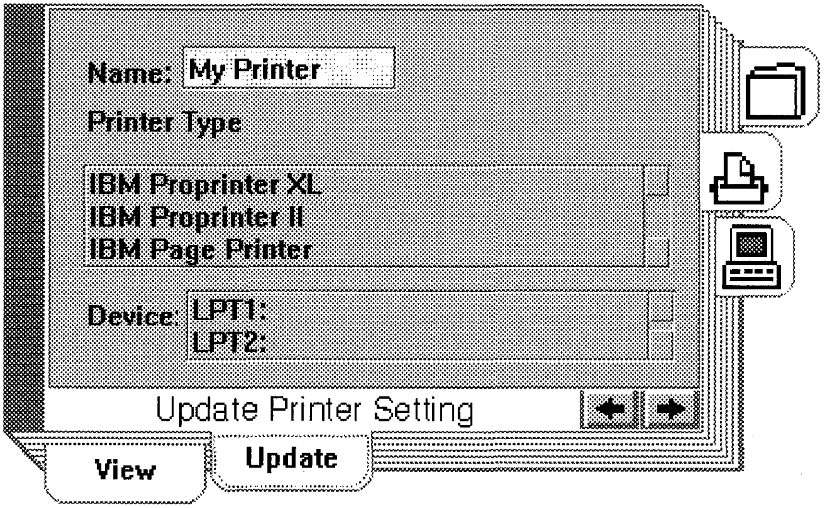

Figure 9-1. Notebook Example

The notebook can be customized to meet varying application requirements, while providing a user interface component that can be used easily to develop products that conform to the Common User Access (CUA) user interface guidelines. The application can specify different colors, sizes, and orientations for its notebooks, but the underlying function of the control remains the same. For a complete description of CUA notebooks, refer to the *SAA CUA Guide to User Interface Design* and the *SAA CUA Advanced Interface Design Reference*.

Notebook Styles

This section describes the following notebook style components:

- Page buttons
- Status line
- Binding
- Intersection of back pages
- Major and minor tabs

- Tab shapes.

Figure 9-2 shows how a notebook control looks when it is created. The figure assumes that pages have been inserted into the notebook with major and minor tab attributes.

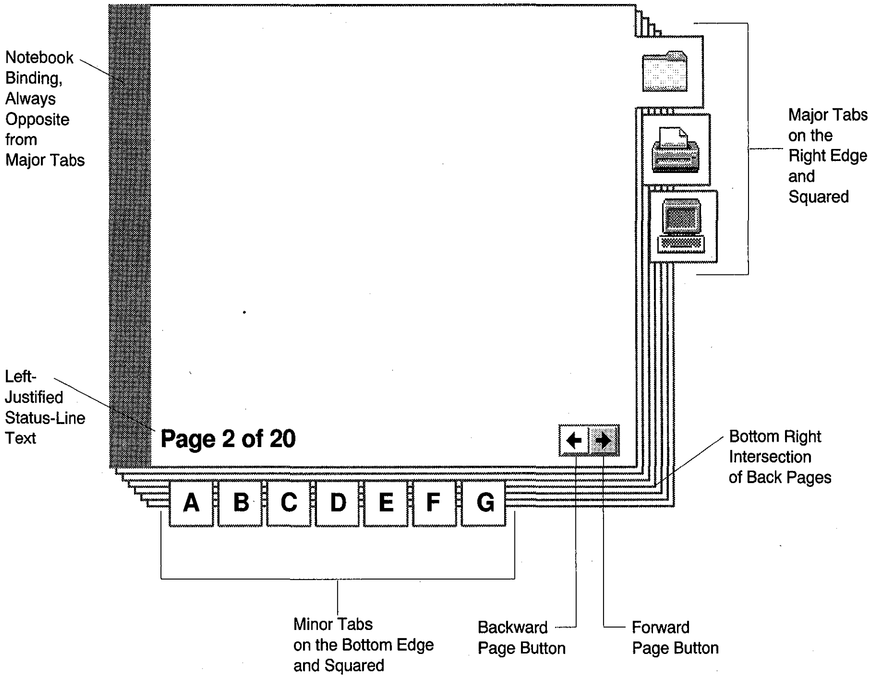

Figure 9-2. Notebook Style and Placement of Major and Minor Tabs

Page Buttons

In the bottom-right corner of the notebook in Figure 9-2 are the *page buttons*. These buttons let you bring one page of the notebook into view at a time. They are a standard component that is automatically provided with every notebook. However, the application can change the default width and height of the page buttons by using the BKM_SETDIMENSIONS message. The page buttons always are located in the corner where the recessed edges of the notebook intersect.

Selecting the *forward page button* (the arrow pointing to the right) causes the next page to be displayed and selecting the *backward page button* (the arrow pointing to the left) causes the previous page to be displayed. In Figure 9-2, the page buttons are displayed with available-state emphasis because pages have been inserted into the notebook. Prior to inserting pages in the notebook, the page buttons are displayed with unavailable-state emphasis; therefore, selecting either page button would not bring a page into view.

Status Line

To the left of the page buttons in the default notebook style setting is the *status line*, which enables the application to provide information to the user about the page currently displayed. The notebook does not supply any default text for the status line. The application is responsible for associating a text string with the status line of each page on which a text string is to be displayed.

The status text is drawn left-justified by default, but it can be drawn centered or right-justified. The same status text justification applies to all pages in the notebook. This setting is specified by the BKS_STATUSTEXTLEFT style bit. The location of the back pages intersection and the major tabs has no effect on the specification of the status line position. This style bit can be set for the entire notebook.

Binding

The notebook control resembles a real notebook in its general appearance. For example, as Figure 9-2 on page 9-2 shows, the notebook has a *binding* that, along with recessed pages on the right and bottom edges, gives the notebook a three-dimensional appearance. The default binding is solid and is placed on the left side. This binding is used if the BKS_SOLIDBIND style bit is specified or if no style bit is specified.

Two styles are provided for the notebook binding: solid and spiral. The notebook is displayed with a solid binding by default, but the application can specify BKS_SPIRALBIND to display a spiral binding.

The placement of the binding depends entirely on the placement of the back pages and major tabs, respectively. The binding always is located on the opposite side of the notebook from the major tabs.

Intersection of Back Pages

The recessed edges that intersect near the page buttons are called the *back pages*. The default notebook's back pages intersect in the bottom-right corner, which means the recessed pages are on the bottom and right edges. This setting is specified by the BKS_BACKPAGESBR style bit. The back pages are important because their intersection determines where the major tabs can be placed, which in turn determines the placement of the binding and the minor tabs.

Major Tabs

Major and *minor* tabs are used to organize related pages into sections. Minor tabs define subsections within major tab sections. The content of each section has a common theme, which is represented to the user by a tabbed divider that is similar to a tabbed page in a notebook.

The BKS_MAJORTABRIGHT style bit specifies that major tabs, if used, are to be placed on the right side of the notebook. This is the default major tab placement when the back pages intersect at the bottom-right corner of the notebook. The binding is located on the left, because it is always located on the opposite side of the notebook from the major tabs.

The placement of the major tabs is limited to one of the two edges on which there are recessed pages. For example, if the application specifies the back pages intersection at the bottom-right corner (BKS_BACKPAGESBR, the default), the major tabs can be placed on either the bottom edge (BKS_MAJORTABBOTTOM) or the right edge (BKS_MAJORTABRIGHT) of the notebook. In this situation, if the application specifies that major tabs are to be placed on the left or top edges of the notebook, the notebook control places them on the right edge anyway—the default placement for back pages intersecting at the bottom-right corner.

When major tabs are defined at the creation of the notebook they are not displayed on screen. Major tab attributes only show at the time a page is inserted into the notebook. This is done by specifying the BKA_MAJOR attribute in the BKM_INSERTPAGE message.

Minor Tabs

Minor tabs are specified using the BKA_MINOR attribute. Minor tabs always are placed perpendicular to the major tabs, based on the intersection of the back pages and the major tab placement. Only one major or minor tab attribute can be specified for each notebook page. Minor tabs are displayed only if the associated major tab page is selected or if the notebook has no major tab pages.

The placement of the minor tabs depends entirely on the placement of the back pages and major tabs, respectively. The minor tabs always are located on the recessed page side that has no major tabs.

Table 9-1 describes the available notebook control styles.

Table 9-1. Notebook Control Styles

Back Pages	**Major Tabs**	**Minor Tabs**	**Binding**
Bottom-right (default)	Bottom	Right	Top
Bottom-right (default)	Right (default)	Bottom	Left
Bottom-left	Bottom (default)	Left	Top
Bottom-left	Left	Bottom	Right
Top-right	Top (default)	Right	Bottom
Top-right	Right	Top	Left
Top-left	Top	Left	Bottom
Top-left	Left (default)	Top	Right

Tab Shapes and Contents

The default shape of the tabs used on notebook divider pages is square. This setting is specified by the BKS_SQUARETABS style bit. The shape of the tabs can be square, rounded, or polygonal. The tab text can be drawn left-justified, right-justified, or centered. Once set, these styles apply to the major and minor tabs for all pages in the notebook. The location of the back pages intersection and the major tabs has no effect on the specification of the tab-shape position. As with the page buttons, the application can change the default width and height of the major and minor tabs by using the BKM_SETDIMENSIONS message.

A notebook tab can contain either text or a bit map. Text is associated with a tab page by using the BKM_SETTABTEXT message. Notebook tab text is centered by default or by specifying the BKS_TABTEXTCENTER style when creating the notebook window. A bit map is placed on a tab by using the BKM_SETTABBITMAP message. A bit map cannot be positioned on a tab because the bit map stretches to fill the rectangular area of the tab; therefore, no style bit is used.

Summary of Notebook Styles

The notebook control provides style bits so that your application can specify or change the default style settings. One style bit from each of the following groups can be specified. If you specify more than one style bit, you must use an OR operator (|) to combine them.

- Type of binding

BKS_SOLIDBIND	Solid (default)
BKS_SPIRALBIND	Spiral

- Intersection of back pages

BKS_BACKPAGESBR	Bottom-right corner (default)
BKS_BACKPAGESBL	Bottom-left corner
BKS_BACKPAGESTR	Top-right corner
BKS_BACKPAGESTL	Top-left corner

- Location of major tabs

BKS_MAJORTABRIGHT	Right edge (default)
BKS_MAJORTABLEFT	Left edge
BKS_MAJORTABTOP	Top edge
BKS_MAJORTABBOTTOM	Bottom edge

- Shape of tabs

BKS_SQUARETABS	Square (default)
BKS_ROUNDEDTABS	Rounded
BKS_POLYGONTABS	Polygonal

- Alignment of text associated with tabs

BKS_TABTEXTCENTER	Centered (default)
BKS_TABTEXTLEFT	Left-justified
BKS_TABTEXTRIGHT	Right-justified

- Alignment of status-line text

BKS_STATUSTEXTLEFT	Left-justified (default)
BKS_STATUSTEXTRIGHT	Right-justified
BKS_STATUSTEXTCENTER	Centered

Using Notebook Controls

The following sections describe how to create pages, insert pages into, create and associate windows for, and delete pages from a notebook.

Notebook Creation

You create a notebook by using the WC_NOTEBOOK window class name in the *ClassName* parameter of WinCreateWindow. The sample code in Figure 9-3 shows the creation of the notebook. The style set in the *ulNotebookStyles* variable (the BKS_* values) specifies that the notebook is to be created with a solid binding and the back pages intersecting at the bottom-right corner, major tabs placed on the right edge, shape tab square, tab text centered, and status-line text left-justified These are the default settings and are given here only to show how notebook styles are set.

```
HWND  hwndNotebook;            /* Notebook window handle         */
ULONG ulNotebookStyles;        /* Notebook window styles         */
HMODULE hmod;                  /* Notebook DLL module handle     */

/*****************************************************************/
/* Set the BKS_style flags to customize the notebook.            */
/*****************************************************************/
ulNotebookStyles =
  BKS_SOLIDBIND     |          /* Use solid binding              */
  BKS_BACKPAGESBR   |          /* Set back pages to intersect at the */
                               /* bottom-right corner            */
  BKS_MAJORTABRIGHT |          /* Position major tabs on right side  */
  BKS_SQUARETABS    |          /* Make tabs square               */
  BKS_TABTEXTCENTER |          /* Center tab text                */
  BKS_STATUSTEXTLEFT;          /* Left-justified status-line text    */

/*****************************************************************/
/* Create the notebook control window.                           */
/*****************************************************************/
hwndNotebook =
  WinCreateWindow(
    hwndParent,                /* Parent window handle           */
    WC_NOTEBOOK,               /* Notebook window class          */
    NULL,                      /* No window text                 */
    ulNotebookStyles,          /* Notebook window styles         */
```

Figure 9-3 (Part 1 of 2). Sample Code for Creating a Notebook

```
    x, y, cx, xy                 /* Origin and size                    */
    hwndOwner,                   /* Owner window handle                */
    HWND_TOP,                    /* Sibling window handle              */
    ID_BOOK,                     /* Notebook window ID                 */
    NULL,                        /* No control data                    */
    NULL;                        /* No presentation parameters         */

/*********************************************************************/
/* Make the notebook control visible.                                */
/*********************************************************************/
WinShowWindow(
  hwndNotebook,                  /* Notebook window handle             */
  TRUE);                         /* Make the window visible            */
```

Figure 9-3 (Part 2 of 2). Sample Code for Creating a Notebook

Figure 9-4 shows how the default notebook control looks when it is created.

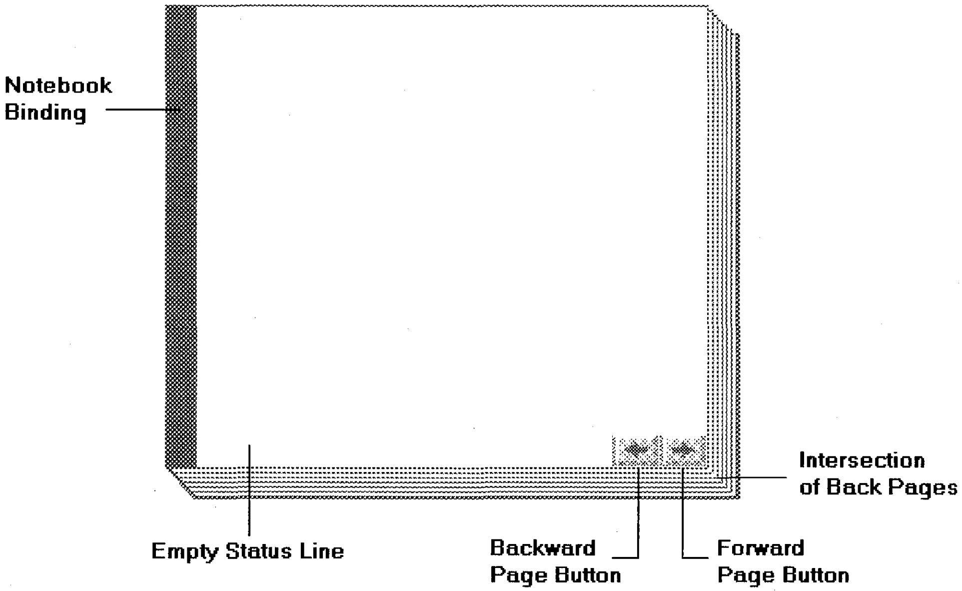

Figure 9-4. Default Notebook Style

Changing Notebook Styles

Figure 9-5 on page 9-8 shows some sample code fragments for setting the notebook style to spiral binding, back pages intersecting at the bottom-left corner, major tabs placed on the bottom edge, tab shape rounded, tab text left-justified, and status-line text centered.

```
/*********************************************************************/
/* Query for the existing notebook window style settings.           */
/*********************************************************************/
ulNotebookStyles =
  WinQueryWindowULong(hwndNotebook,    /* Notebook window handle */
                      QWL_STYLE);      /* Set notebook style     */

/*********************************************************************/
/* Reset notebook window style flags, leaving window flags unchanged. */
/*********************************************************************/
ulNotebookStyles &= 0xFFFF0000;

/*********************************************************************/
/* Setup the new notebook window style flags.                       */
/*********************************************************************/
ulNotebookStyles |=
  BKS_SPIRALBIND       |        /* Use spiral binding                */
  BKS_BACKPAGESBL      |        /* Set back pages to intersect at the */
                                /* bottom-left corner                */
  BKS_MAJORTABBOTTOM   |        /* Position major tabs on bottom edge */
  BKS_ROUNDEDTABS      |        /* Make tabs rounded                 */
  BKS_TABTEXTLEFT      |        /* Left-justified tab text           */
  BKS_STATUSTEXTCENTER;         /* Center status-line text           */

/*********************************************************************/
/* Set the new notebook style.                                       */
/*********************************************************************/
WinSetWindowULong(
  hwndNotebook,                 /* Notebook window handle            */
  QWL_STYLE,                    /* Window style                      */
  ulNotebookStyles);            /* Set notebook style                */

/*********************************************************************/
/* Invalidate to force a repaint.                                    */
/*********************************************************************/
WinInvalidateRectl(
  hwndNotebook,                 /* Notebook window handle            */
  NULL,                         /* Invalidate entire window,         */
  TRUE);                        /* including children                */
```

Figure 9-5. Sample Code for Changing the Notebook Style

Figure 9-6 on page 9-9 shows how the notebook appears when these style bits are set. This figure assumes that pages have been inserted into the notebook with major and minor tab attributes.

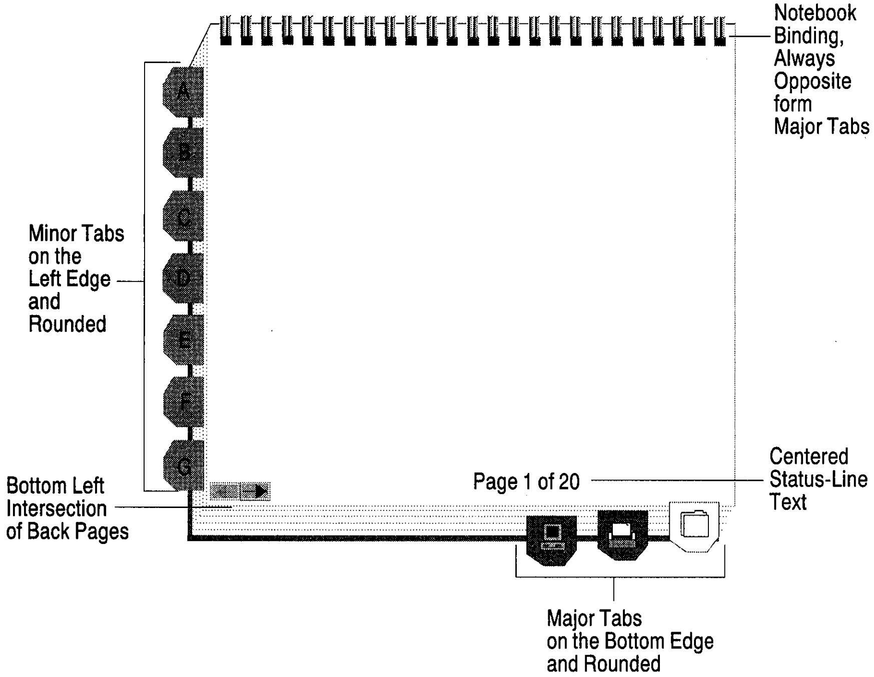

Figure 9-6. Notebook with Style Settings Changed

Inserting Notebook Pages

After a notebook is created, pages can be inserted into the notebook by using the BKM_INSERTPAGE message. BKM_INSERTPAGE provides several attributes that can affect the inserted pages. When inserting pages into either a new notebook or an existing one, consider carefully how the user expects those pages to be organized.

Major and Minor Tabs

The two attributes that have the most impact on how notebook pages are organized are BKA_MAJOR and BKA_MINOR, which specify major and minor tabs, respectively. Major tab pages define the beginning of major sections in the notebook, while minor tab pages define the beginning of subsections within a major section. Major sections should begin with a page that has a BKA_MAJOR attribute. Within major sections, information can be organized into minor sections, each of which should begin with a page that has a BKA_MINOR attribute.

For an existing notebook, the underlying hierarchy, if one exists, must be observed when inserting new pages, to provide efficient organization and navigation of the information in the notebook. For example, if the notebook has minor sections but no major sections, you could confuse the user if you inserted a page with a major tab attribute between related minor sections or at the end of the notebook.

If you insert pages without specifying tab attributes, those pages become part of the section in which they are inserted. For example, if page 7 of your notebook has a minor tab and you insert a new page 8 without specifying a tab attribute, page 8 becomes part of the section that begins with the minor tab on page 7.

Because tab pages are not mandatory, the application can create a notebook that contains no major or minor tab pages. That style would be similar to that of a composition notebook.

Another group of attributes that can affect the organization of pages being inserted into a notebook consists of BKA_FIRST, BKA_LAST, BKA_NEXT, and BKA_PREV. These attributes cause pages to be inserted at the end, at the beginning, after a specified page, and before a specified page of a notebook, respectively.

Status Line

Each page has an optional status line that can be used to display information for the user. To include this status line, the application must specify the BKA_STATUSTEXTON attribute when inserting the page. If the application inserts the page without specifying this attribute, the status line is not available for that page.

To display text on the status line of the specified page, the application must use the BKM_SETSTATUSLINETEXT message to associate a text string with the page. A separate message must be sent for each page that is to display status-line text. If the application does not send a BKM_SETSTATUSLINETEXT message for a page, no text is displayed in the status line of that page. The application can send this message to the notebook at any time to change the status-line text. The status line can be cleared by setting the text to NULL.

The sample code in Figure 9-7 on page 9-11 shows how to insert a page into a notebook, where the inserted page has a major tab attribute, the status line is available, and the page is inserted after the last page in the notebook. This sample code also shows how to associate a text string with the status line of the inserted page.

```
HWND  hwndNotebook;             /* Notebook window handle          */
ULONG ulPageId;                 /* Page identifier                 */

/**********************************************************************/
/* Insert a new page into a notebook.                                 */
/**********************************************************************/
ulPageId = (ULONG) WinSendMsg(
  hwndNotebook,                 /* Notebook window handle          */
  BKM_INSERTPAGE,               /* Message for inserting a page    */
  (MPARAM)NULL,                 /* NULL for page ID                */

  MPFROM2SHORT(
    BKA_MAJOR |                 /* Insert page with a major tab    */
                                /* attribute                       */
    BKA_STATUSTEXTON),          /* Make status-line text visible   */
    BKA_LAST));                 /* Insert this page at end of notebook */

/**********************************************************************/
/* Set the status-line text.                                          */
/**********************************************************************/
WinSendMsg(
  hwndNotebook,                 /* Notebook window handle          */
  BKM_SETSTATUSLINETEXT,        /* Message for setting status-line */
                                /* text                            */
  (MPARAM)ulPageId,             /* ID of page to receive status-line */
                                /* text                            */
  MPFROMP("Page 1 of 2"));      /* Text string to put on status line */
```

Figure 9-7. Sample Code for Inserting a Notebook Page

Setting and Querying Page Information

The information for a page in the notebook can be set and queried with BKM_SETPAGEINFO and BKM_QUERYPAGEINFO respectively. By using these messages, all the information associated with a page can be accessed at once. In addition, BKM_SETPAGEINFO can be used to delay the loading of a page until it is turned to, by setting the *bLoadDlg* field to FALSE. By doing this for all pages in a notebook, the notebook is created much more quickly.

Associating Application Page Windows with Notebook Pages

After a page is inserted into a notebook, you must facilitate the display of information for this page when it is brought to the top of the book. The notebook provides a top page area in which the application can display windows or dialogs for the topmost page. For each inserted page, the application must associate the handle of a window or dialog that is to be invalidated when the page is brought to the top of the book. The application can associate the same handle with different pages, if desired.

The application must send a BKM_SETPAGEWINDOWHWND message to the notebook in order to associate the application page window or dialog handle with the notebook page being inserted. Once done, the notebook invalidates this window or dialog whenever the notebook page is brought to the top of the book. If no application page window handle is specified for an inserted page, no invalidation can be done by the notebook for that page. However, the application receives a BKN_PAGESELECTED notification code when a new page is brought to the top of the notebook, at which time the application can invalidate the page.

The notebook also sends a BKN_PAGESELECTEDPENDING notification code to the application before the new page is selected. The application can use this message to prevent the page from being turned to. This is useful when the application wants to validate a page's contents.

The following sections describe how to associate either a window handle or a dialog handle with an inserted page.

Associating a Window with a Notebook Page

A calendar example is used to show how a page can be implemented as a window. Figure 9-8 shows a calendar that is divided into four years (major tabs). Within each year are months (minor tabs) grouped into quarters. The top page has a window associated with it. The window paint processing displays the days for the currently selected month and year.

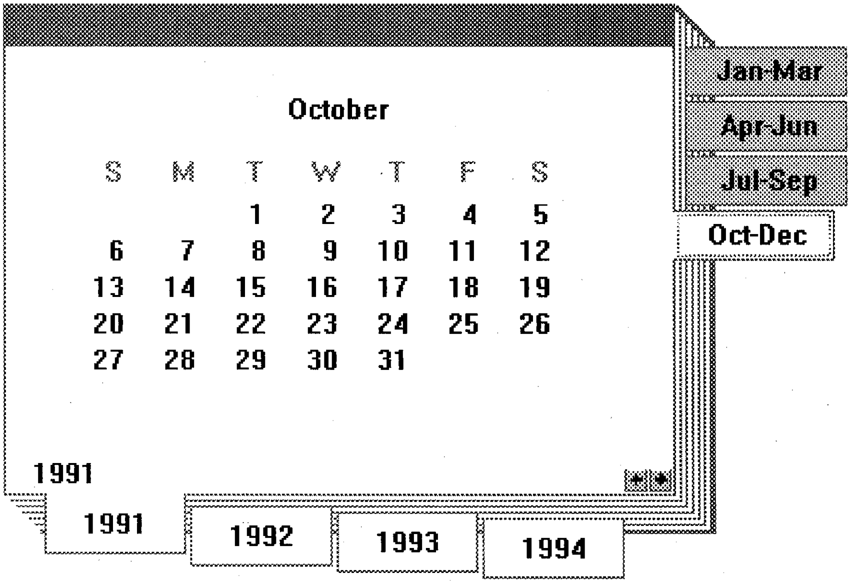

Figure 9-8. Calendar Inserted into an Application Page Window

The sample code in Figure 9-9 on page 9-13 shows how the window procedure for the calendar, in Figure 9-8, is registered with the application. Also, it shows how the window is created and associated with the notebook page. The example ends by showing the window procedure for the associated window.

```
/**********************************************************************/
/* Registration of window procedure for calendar.                     */
/**********************************************************************/
WinRegisterClass(hab,                /* Register a page window class  */
                 "Calendar Page",    /* Class name                    */
                 PageWndProc,        /* Window procedure              */
                 CS_SIZEREDRAW,      /* Class style                   */
                 0);                 /* No extra bytes reserved       */

/**********************************************************************/
/* Create the window.                                                 */
/**********************************************************************/
hwndPage = WinCreateWindow(hwndNotebook,       /* Parent              */
                           "Calendar Page",    /* Class               */
                           NULL,               /* Title text          */
                           0L,                 /* Style               */
                           0, 0, 0, 0,         /* Origin and size     */
                           hwndNotebook,       /* Owner               */
                           HWND_TOP,           /* Z-order             */
                           ID_WIN_CALENDAR_PAGE,   /* ID              */
                           NULL,               /* Control data        */
                           NULL);              /* Presparams          */

/**********************************************************************/
/* Associate window with the inserted notebook page.                  */
/**********************************************************************/
WinSendMsg(hwndBook,
           BKM_SETPAGEWINDOWHWND,
           MPFROMLONG(ulPageId),
           MPFROMHWND(hwndPage));
```

Figure 9-9 (Part 1 of 2). Sample Code for Associating a Window with a Notebook Page

```
/*******************************************************************/
/* Window procedure.                                               */
/*******************************************************************/
MRESULT EXPENTRY PageWndProc(HWND hwnd,USHORT msg,MPARAM mp1,MPARAM mp2)
{
HPS hps;

switch (msg)
{

  /*****************************************************************/
  /* WM_CREATE is sent when the window is created.                 */
  /*****************************************************************/
  case  WM_CREATE:

    /***************************************************************/
    /* Place window initialization code here.                      */
    /***************************************************************/
    break;

  case  WM_PAINT:
    /***************************************************************/
    /* Draw the calendar for the current selected year and month.  */
    /***************************************************************/
    hps = WinBeginPaint(hwnd, NULL, NULL);
    drawMonthCalendar(hps, windowSize, currDate.year, currDate.month);
    WinEndPaint(hps);
    break;

  default:
    return (WinDefWindowProc(hwnd, msg, mp1, mp2));
    break;
}

return (FALSE);
}
```

Figure 9-9 (Part 2 of 2). Sample Code for Associating a Window with a Notebook Page

Associating a Dialog with a Notebook Page

To illustrate the notebook implemented as a dialog, a Properties Notebook is used. In Figure 9-10 on page 9-15 the various objects whose properties can be changed or updated are displayed as major tabs. Included are sections that represent a folder, printer, and display (major tabs). The printer object is currently selected. Within the printer object, the user can choose to "View" or "Update" (minor tabs) the printer settings. The topmost page is a printer dialog from which the user can update the printer name, type, and device information.

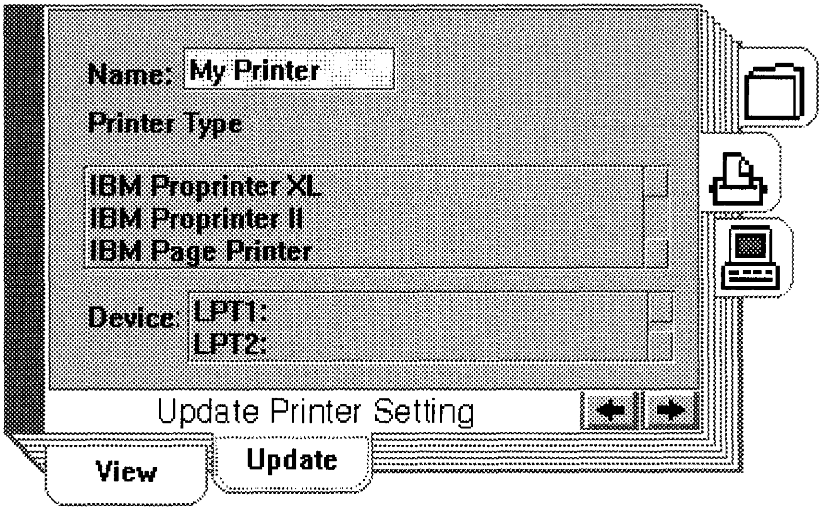

Figure 9-10. Dialog Used as an Application Page Window

The sample code in Figure 9-11 shows how the printer dialog is created and associated with a notebook page. The example ends by showing the dialog procedure for the associated dialog.

```
SEL sel = NULL;
PDLGTEMPLATE pDlgt;

/*********************************************************************/
/* Create a dialog.                                                  */
/*********************************************************************/
DosGetResource(NULL,RT_DIALOG,ID_DLG_PRINTDRV,&sel);
pDlgt = MAKEP(sel,0);
hwndPage = WinCreateDlg(HWND_DESKTOP,     /* Parent window handle     */
                        hwndBook,         /* Owner window handle      */
                        fnwpPrint,        /* Dialog procedure         */
                                          /* address                  */
                        pDlgt,            /* Dialog data structure    */
                                          /* address                  */
                        NULL);            /* Application data         */
```

Figure 9-11 (Part 1 of 2). Sample Code for Associating a Dialog with a Notebook Page

```
/*********************************************************************/
/* Associate dialog with the inserted notebook page.                 */
/*********************************************************************/
WinSendMsg(hwndBook,
           BKM_SETPAGEWINDOWHWND,
           MPFROMLONG(ulPageId),
           MPFROMHWND(hwndPage));

/*********************************************************************/
/* Dialog procedure.                                                 */
/*********************************************************************/
MRESULT EXPENTRY fnwpPrint(HWND hwndDlg,USHORT msg,MPARAM mp1,MPARAM mp2)
{
  switch (msg)
  {
    case WM_INITDLG:

      /***************************************************************/
      /* Place dialog initialization code here.                      */
      /***************************************************************/
      break;

    case WM_COMMAND:
      return ((MRESULT) FALSE);
      break;

    default:
      return WinDefDlgProc (hwndDlg,msg,mp1,mp2);
  }
  return WinDefDlgProc (hwndDlg,msg,mp1,mp2);
}
```

Figure 9-11 (Part 2 of 2). Sample Code for Associating a Dialog with a Notebook Page

Deleting Notebook Pages

The BKM_DELETEPAGE message is used to delete one or more pages from the notebook. The application can delete one page (BKA_SINGLE attribute), all pages within a major or minor tab section (BKA_TAB attribute), or all of the pages in the notebook (BKA_ALL attribute). The default, if no attributes are specified, is to delete no pages. The sample code in Figure 9-12 on page 9-17 shows how the BKM_QUERYPAGEID message is used to get the ID of the top page and how the BKM_DELETEPAGE message is then used to delete that page.

```
/**********************************************************************/
/* Set the range of pages to be deleted.                              */
/**********************************************************************/

/* Set attribute to delete a single page. */
usDeleteFlag = BKA_SINGLE

/**********************************************************************/
/* Get the ID of the notebook's top page.                             */
/**********************************************************************/
ulPageId = (ULONG) WinSendMsg(
  hwndNotebook,                /* Notebook window handle             */
  BKM_QUERYPAGEID,             /* Message to query a page ID         */
  NULL,                        /* NULL for page ID                   */
  (MPARAM)BKA_TOP);            /* Get ID of top page                 */

/**********************************************************************/
/* Delete the notebook's top page.                                    */
/**********************************************************************/
WinSendMsg(
  hwndNotebook,                /* Notebook window handle             */
  BKM_DELETEPAGE,              /* Message to delete the page         */
  MPFROMLONG(ulPageId),        /* ID of page to be deleted           */
  (MPARAM)usDeleteFlag);       /* Range of pages to be deleted       */
```

Figure 9-12. Sample Code for Deleting a Notebook Page

Notebook Colors

The application can change the color of any part of the notebook. The colors of some parts can be changed by specifying presentation parameter attributes in WinSetPresParam. Other colors can be changed by specifying notebook attributes in the BKM_SETNOTEBOOKCOLORS message. The following sections define which parts of the notebook can have their colors changed by each of these two methods.

Changing Colors Using WinSetPresParam

WinSetPresParam is used to change the color of the notebook outline, window background, selection cursor, and status-line text. The following list shows the mapping between the various notebook parts and their associated presentation parameter attributes.

Notebook outline
PP_BORDERCOLOR or PP_BORDERCOLORINDEX. This color is set initially to SYSCLR_WINDOWFRAME.

Notebook window background
PP_BACKGROUNDCOLOR or PP_BACKGROUNDCOLORINDEX. This color is set initially to SYSCLR_FIELDBACKGROUND.

Selection cursor
PP_HILITEBACKGROUNDCOLOR or PP_HILITEBACKGROUNDCOLORINDEX. This color is set initially to SYSCLR_HILITEBACKGROUND.

Status-line text
PP_FOREGROUNDCOLOR or PP_FOREGROUNDCOLORINDEX. This color is initially set to SYSCLR_WINDOWTEXT.

If a presentation parameter attribute is set, all parts of the notebook that are mapped to this color are changed. The sample code in Figure 9-13 shows how to change the color of the notebook outline.

```
/* Set number of bytes to be passed in usColorIdx */
/* for color index table value                    */
usColorLen = 4;
/* Set color index table value to be assigned */
ulColorIdx = 3;

/**********************************************************************/
/* Set the notebook outline color.                                    */
/**********************************************************************/
WinSetPresParam(
  hwndNotebook,               /* Notebook window handle            */
  PP_BORDERCOLOR,             /* Border color attribute            */
  usColorLen,                 /* Number of bytes in color index    */
                              /* table value                       */
  &ulColorIdx);               /* Color index table value           */
```

Figure 9-13. Sample Code for Changing the Color of the Notebook Outline

Changing Colors Using BKM_SETNOTEBOOKCOLORS

The BKM_SETNOTEBOOKCOLORS message is used to change the color of the major tab background and text, the minor tab background and text, and the notebook page background. The following list shows the mapping between the various notebook parts and their associated notebook attributes.

Major tab background
BKA_BACKGROUNDMAJORCOLOR or BKA_BACKGROUNDMAJORCOLORINDEX. This color is set initially to SYSCLR_PAGEBACKGROUND. The currently selected major tab has the same background color as the notebook page background.

Major tab text
BKA_FOREGROUNDMAJORCOLOR or BKA_FOREGROUNDMAJORCOLORINDEX. This color is set initially to SYSCLR_WINDOWTEXT.

Minor tab background
BKA_BACKGROUNDMINORCOLOR or BKA_BACKGROUNDMINORCOLORINDEX. This color is set initially to SYSCLR_PAGEBACKGROUND. The currently selected minor tab has the same background color as the notebook page background.

Minor tab text
BKA_FOREGROUNDMINORCOLOR or BKA_FOREGROUNDMINORCOLORINDEX. This color is set initially to SYSCLR_WINDOWTEXT.

Notebook page background
BKA_BACKGROUNDPAGECOLOR or BKA_BACKGROUNDPAGECOLORINDEX. This color is set initially to SYSCLR_PAGEBACKGROUND.

If a notebook attribute is set, all parts of the notebook that are mapped to this color are changed. The sample code in Figure 9-14 shows how to change the color of the major tab background.

```
/* Color index value     */
ulColorIdx    = SYSCLR_WINDOW;
/* Major tab background  */
ulColorRegion = BKA_BACKGROUNDMAJORCOLORINDEX;

WinSendMsg(hwndBook,
           BKM_SETNOTEBOOKCOLORS,
           MPFROMLONG(ulColorIdx),
           MPFROMLONG(ulColorRegion));
```

Figure 9-14. Sample Code for Changing the Color of the Major Tab Background

Graphical User Interface Support for Notebook Controls

The following section describes the support for graphical user interfaces (GUIs) provided by the notebook control. Except where noted, this support conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

The GUI support provided by the notebook control consists of the notebook navigation techniques.

Notebook Navigation Techniques

The notebook control supports the use of a pointing device and the keyboard for displaying notebook pages and tabs and for moving the selection cursor from the notebook tabs to the application window and the other way around.

Note: If more than one notebook window is open, displaying a page or tab in one notebook window has no effect on the pages or tabs displayed in any other notebook window.

Pointing Device Support

A user can use a pointing device to display notebook pages or tabs by selecting the notebook components described in the following list. The CUA guidelines define mouse button 1 (the select button) to be used for selecting these components. This definition also applies to the same button on any other pointing device a user might have.

- Selecting tabs using a pointing device

  A tab can be selected to bring a page that has a major or minor tab attribute to the top of the notebook. The *selection cursor*, a dotted outline, is drawn inside the tab's border to indicate the selected tab. In addition, the selected tab is given the same background color as the notebook page area. The color of the other tabs is specified in the BKM_SETNOTEBOOKCOLORS message. This helps the user distinguish the selected tab from the other tabs if different colors are used.

  Because all tabs are mutually exclusive, only one of them can be selected at a time. Therefore, the only type of selection supported by the notebook control is *single selection*. This selection type conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

  If the user moves the pointing device to a place in the notebook page window that can accept a cursor, such as an entry field, check box, or radio button, and presses the select button, the selection cursor is removed from the tab it is on and is displayed in the notebook page window. The selection cursor never can be displayed both on a tab and in the notebook page window at the same time.

- Selecting page buttons using a pointing device

  A forward or backward *page button* can be selected to display the next or previous page, respectively, one at a time. The arrow pointing to the right is the forward page button, and the arrow pointing to the left is the backward page button. When the selection of a page button brings a page that has a major or minor tab to the top of the notebook, the selection cursor is drawn inside that tab's border.

- Selecting tab scroll buttons using a pointing device

  A user can decrease the size of a notebook window so that some of the available notebook tabs cannot be displayed. When this happens, the notebook control automatically draws *tab scroll buttons* at the corners of the notebook side or sides to notify the user that more tabs are available.

  Tab scroll buttons have another purpose: to give the user the means to scroll into view, one at a time, the tabs that are not displayed. The user does this by selecting a forward or backward tab scroll button, which causes the next tab to scroll into view, but does not change the location of the selection cursor. Once the tab is in view, the user can display that tab's page by selecting the tab.

  A maximum of four tab scroll buttons can be displayed: two for the major tab side and two for the minor tab side. Figure 9-15 on page 9-21 is an example of a notebook with two of its tab scroll buttons displayed on the bottom-left and bottom-right corners of the minor tab side.

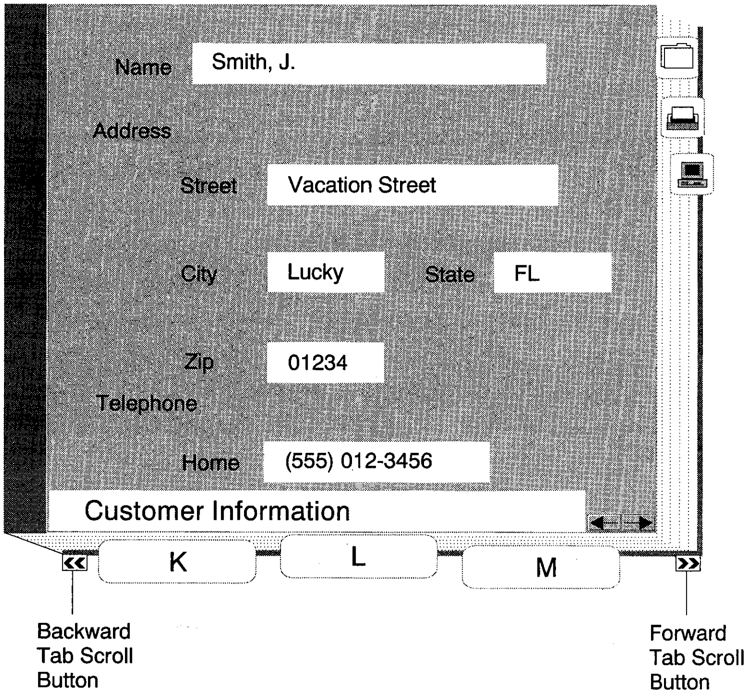

Figure 9-15. Notebook with Two Tab Scroll Buttons Displayed

In this example, only three minor tabs are displayed because the notebook is not wide enough to display more. Here, the user can display a previous minor tab by selecting the backward tab scroll button or a following minor tab by selecting the forward tab scroll button.

When the first tab in the notebook is displayed, the backward tab scroll button is deactivated. Unavailable-state emphasis is applied to it to show that no more tabs can be scrolled into view by using the backward tab scroll button. Unavailable-state emphasis is applied to the forward tab scroll button if the last tab in the notebook is displayed.

Keyboard Support

The users can utilize the keyboard to display and manipulate notebook pages and components.

Focus on Application Dialog or Window

If the application dialog page or window has the focus, the notebook handles the following keyboard interactions:

Keyboard Input	Description
Alt+PgDn or PgDn	Brings the next page to the top of the notebook. If the application uses the PgDn key, then it must be used in combination with the Alt key.
Alt+PgUp or PgUp	Brings the previous page to the top of the notebook. If the application uses the PgUp key, then it must be used in combination with the Alt key.
Alt+Up Arrow	Switch the focus to the notebook window.
Tab	Move the cursor to the next control within the top page window or dialog. If the cursor is currently on the last control within the top page window or dialog when the Tab key is pressed, the cursor is moved to the notebook major tab, if it exists; else to the minor tab, if it exists; else to the right page button.
Shift+Tab	Move the cursor to the previous control within the top page window or dialog. If the cursor is currently on the first control within the top page window or dialog when the Shift+Tab key is pressed, the cursor is moved to the previous control. If the previous control is the notebook, the cursor is moved to the right page button.

Focus on the Notebook Control

If the notebook control has the focus, it handles the following keyboard intereactions:

Keyboard Input	Description
Alt+Down Arrow	Switch the focus to the application's primary window.
Alt+PgDn or PgDn	Brings the next page to the top of the notebook.
Alt+PgUp or PgUp	Brings the previous page to the top of the notebook.
Left or Up Arrow	If the cursor is currently on a major tab, it is moved to the previous major tab. If the previous major tab is not visible, the tabs are scrolled to bring the previous major tab into view. If the first major tab is reached, scrolling ends. If the cursor is currently on a minor tab, it is moved to the previous minor tab. If the previous minor tab is not visible, the tabs are scrolled to bring the previous minor tab into view. If the first minor tab is reached, scrolling ends.

	If the cursor is currently on the right page button, the cursor moves to the left page button. If the cursor is currently on the left page button, no action is taken.
Right or Down Arrow	If the cursor is currently on a major tab, it is moved to the next major tab. If the next major tab is not visible, the tabs are scrolled to bring the next major tab into view. If the last major tab is reached, scrolling ends. If the cursor is currently on a minor tab, it is moved to the next minor tab. If the next minor tab is not visible, the tabs are scrolled to bring the next minor tab into view. If the last minor tab is reached, scrolling ends. If the cursor is currently on the right page button, no action is taken. If the cursor is currently on the left page button, the cursor moves to the right page button.
Tab	The cursor moves from the major tab, then to the minor tab, then to the right page button, and then to the last tab stop in the application dialog or window.
Shift+Tab	The cursor moves from the page button, to the minor tab, to the major tab, and then to the first tab stop in the application dialog or window.
Home	Brings the first page of the notebook to the top and sets the cursor on the associated tab.
End	Brings the last page of the notebook to the top and sets the cursor on the associated tab.
Enter or Spacebar	If the cursor is on a major or minor tab, the associated page is brought to the top of the notebook, and the selected tab is given the same background color as the notebook page area. The other tabs have their color specified in the BKM_SETNOTEBOOKCOLORS message. This helps the user distinguish the selected tab from the other tabs if different colors are used. If the cursor is currently on the right page button, the next page is brought to the top of the notebook. If the cursor is currently on the left page button, the previous page is brought to the top of the notebook.
Mnemonics	Mnemonics are underlined characters in the text of a tab that cause the tab's page to be selected. Coding a tilde (~) before a text character in the BKM_SETTABTEXT message causes that character to be underlined and activates it as a mnemonic-selection character. A user performs mnemonic selection by pressing a character key that corresponds to an underlined character. When this happens, the tab that contains the underlined character is

selected, and that tab's page is brought to the top of the notebook.

Note: Mnemonic selection is not case sensitive, so the user can type the underscored letter in either uppercase or lowercase.

Enhancing Notebook Controls Performance and Effectiveness

This section provides the following information to enable you to fine-tune a notebook control:

- Dynamic resizing and scrolling
- Tab painting and positioning.

Dynamic Resizing and Scrolling

The notebook control supports *dynamic resizing* by recalculating the size of the notebook's parts when either the user or the application changes the size of any of those parts. A BKN_NEWPAGESIZE notification code is sent from the notebook to the application whenever the notebook's size changes.

The notebook handles the sizing and positioning of each application page window if the BKA_AUTOPAGESIZE attribute is specified for the inserted notebook page. Otherwise, the application must handle this when it receives the BKN_NEWPAGESIZE notification code from the notebook.

If the size of the notebook window is decreased so that the page window is not large enough to display all the information the page contains, the information in the page window is clipped. If scroll bars are desired to enable the clipped information to be scrolled into view, they must be provided by the application. Tab scroll buttons are automatically displayed if the size of the notebook is decreased so that all the major or minor tabs cannot be displayed. For example, a notebook has major tabs on the right side, but the height of the notebook does not allow all the tabs to be displayed. In this case, tab scroll buttons are displayed on the upper- and lower-right corners of the notebook.

Tab Painting and Positioning

The tab pages provide a method for organizing the information in a notebook so that the user easily can see and navigate to that information. When a page is inserted with a major or minor tab attribute, the notebook displays a tab for that page, based on the orientation of the notebook. The contents of the tab can be painted either by the notebook control or the application.

If the notebook control is to paint the tabs, the application must associate a text string or bit map with the page whose tab is to be drawn. This is done by sending the BKM_SETTABTEXT or BKM_SETTABBITMAP message to the notebook control for the specified page. If neither of these messages is sent for an inserted page with a major or minor tab attribute, the application must draw the contents of the tab, through *ownerdraw*. The application receives a WM_DRAWITEM message whenever a tab page that has no text or bit map associated with it is to be drawn. The application can either draw the tab contents or return FALSE, in which case the notebook control fills the tab with the tab background color.

Positioning Tabs in Relation to the Top Tab:
There are seven page edges that define the back pages. The page attribute (BKA_MAJOR or BKA_MINOR) and the topmost page determine how the tabs are positioned. In most cases, the tabs must be drawn when their position changes. For example, this can happen when a page with a tab attribute is brought to the top of the notebook.

The new top major or minor tab will appear attached to the top page. The other tabs will appear as described in the following list. This information is provided to help you understand the relationship between the top tab and the other tabs so that you can organize the information you put into a notebook appropriately. The application has no control over tab positioning.

- When the top page is a major tab page:
  - Any major tabs prior to the top major tab are aligned on the last page of the notebook.
  - Any major tabs after the top major tab are incrementally cascaded from the topmost edge to the last page.
  - If the top major tab has minor tabs, no major tab is drawn on the page edge that immediately follows the top tab page. Instead, any major tabs that follow the top tab are incrementally cascaded, beginning on the second page, edge-down from the top tab. This is done to account for the minor tabs that are positioned between the top major tab and the major tab that follows it on the perpendicular notebook edge.

    The minor tabs are all positioned on the third page edge from the top, thereby giving the appearance of being between the top major tab and the next major tab.
- When the top page is a minor tab page:
  - Any minor tabs prior to the top minor tab are positioned on the third page edge from the top of the notebook.
  - Any minor tabs after the top minor tab are incrementally cascaded up to the third page edge from the top.

Related Functions

This section covers the functions that are related to notebook controls.

WinInvalidateRect

This function adds a rectangle to a window's update region.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinInvalidateRect  (HWND hwnd, PRECTL pwrc, BOOL fIncludeChildren)
```

Parameters

hwnd (HWND) – input
Handle of window whose update region is to be changed.

HWND_DESKTOP	This function applies to the whole screen (or desktop).
Other	Handle of window whose update region is to be changed.

pwrc (PRECTL) – input
Update rectangle.

NULL	The whole window is to be added into the window's update region.
Other	Rectangle to be added to the window's update region.

fIncludeChildren (BOOL) – input
Invalidation-scope indicator.

TRUE	Include the descendants of *hwnd* in the invalid rectangle.
FALSE	Include the descendants of *hwnd* in the invalid rectangle, but only if the parent does not have a WS_CLIPCHILDREN style.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetPresParam

This function sets a presentation parameter for a window.

Syntax

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

BOOL WinSetPresParam  (HWND hwnd, ULONG idAttrType,
                       ULONG cbAttrValueLen, PVOID pAttrValue)
```

Parameters

hwnd (HWND) – input
Window handle.

idAttrType (ULONG) – input
Attribute type identity.

PP_FOREGROUNDCOLOR	Foreground color (in RGB) attribute.
PP_BACKGROUNDCOLOR	Background color (in RGB) attribute.
PP_FOREGROUNDCOLORINDEX	Foreground color index attribute.
PP_BACKGROUNDCOLORINDEX	Background color index attribute.
PP_HILITEFOREGROUNDCOLOR	Highlighted foreground color (in RGB) attribute, for example for selected menu items.
PP_HILITEBACKGROUNDCOLOR	Highlighted background color (in RGB) attribute.
PP_HILITEFOREGROUNDCOLORINDEX	Highlighted foreground color index attribute.
PP_HILITEBACKGROUNDCOLORINDEX	Highlighted background color index attribute.
PP_DISABLEDFOREGROUNDCOLOR	Disabled foreground color (in RGB) attribute.
PP_DISABLEDBACKGROUNDCOLOR	Disabled background color (in RGB) attribute.
PP_DISABLEDFOREGROUNDCOLORINDEX	Disabled foreground color index attribute.
PP_DISABLEDBACKGROUNDCOLORINDEX	Disabled background color index attribute.
PP_BORDERCOLOR	Border color (in RGB) attribute.

PP_BORDERCOLORINDEX	Border color index attribute.
PP_FONTNAMESIZE	Font name and size attribute.
PP_ACTIVECOLOR	Active color value of data type RGB.
PP_ACTIVECOLORINDEX	Active color index value of data type LONG.
PP_INACTIVECOLOR	Inactive color value of data type RGB.
PP_INACTIVECOLORINDEX	Inactive color index value of data type LONG.
PP_ACTIVETEXTFGNDCOLOR	Active text foreground color value of data type RGB.
PP_ACTIVETEXTFGNDCOLORINDEX	Active text foreground color index value of data type LONG.
PP_ACTIVETEXTBGNDCOLOR	Active text background color value of data type RGB.
PP_ACTIVETEXTBGNDCOLORINDEX	Active text background color index value of data type LONG.
PP_INACTIVETEXTFGNDCOLOR	Inactive text foreground color value of data type RGB.
PP_INACTIVETEXTFGNDCOLORINDEX	Inactive text foreground color index value of data type LONG.
PP_INACTIVETEXTBGNDCOLOR	Inactive text background color value of data type RGB.
PP_INACTIVETEXTBGNDCOLORINDEX	Inactive text background color index value of data type LONG.
PP_SHADOW	Changes the color used for drop shadows on certain controls.
PP_USER	This is a user-defined presentation parameter.

cbAttrValueLen (ULONG) – input
Byte count of the data passed in the *pAttrValue* parameter.

pAttrValue (PVOID) – input
Attribute value.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Related Window Messages

This section covers the window messages that are related to notebook controls.

BKM_CALCPAGERECT

This message calculates an application page rectangle from a notebook rectangle or calculates a notebook rectangle from an application page rectangle, depending on the setting of the *bPage* parameter.

Parameters

param1

pRectl (PRECTL)
Pointer to the RECTL structure that contains the coordinates of the rectangle.

If the *bPage* parameter is TRUE, this structure contains the coordinates of a notebook window on input, and on return it contains the coordinates of an application page window.

If the *bPage* parameter is FALSE, this structure contains the coordinates of an application page window on input, and on return it contains the coordinates of a notebook window.

param2

bPage (BOOL)
Window specifier.

Specifies whether the window coordinates to calculate are for a notebook window or an application page window.

TRUE	An application page window is calculated.
FALSE	A notebook window is calculated.

Returns

rc (BOOL)
Success indicator.

TRUE	Coordinates were successfully calculated.
FALSE	Unable to calculate coordinates. This is returned if an invalid RECTL structure is specified in the *pRectl* parameter.

BKM_DELETEPAGE

This message deletes the specified page or pages from the notebook data list.

Parameters

param1

ulPageId (ULONG)
Page identifier.

Page identifier for deletion. This is ignored if the BKA_ALL attribute of the *usDeleteFlag* parameter is specified.

param2

usDeleteFlag (USHORT)
Page range attribute.

Attribute that specifies the range of pages to be deleted.

BKA_SINGLE — Delete a single page.

BKA_TAB — If the page ID specified is that of a page with a major tab attribute, delete that page and all subsequent pages up to the next page that has a major tab attribute.

If the page ID specified is that of a page with a minor tab attribute, delete that page and all subsequent pages up to the next page that has either a major or minor tab attribute.

This attribute should only be specified for pages that have major or minor tab attributes. If a page with neither of these attributes is specified, FALSE is returned and no pages are deleted.

BKA_ALL — Delete all pages in the notebook.

Returns

rc (BOOL)
Success indicator.

TRUE — Pages were successfully deleted.
FALSE — Unable to delete the page or pages. This is returned if an invalid page ID is specified for the *ulPageId* parameter or if the BKA_TAB attribute is specified for a page that has neither a major nor a minor tab attribute.

BKM_INSERTPAGE

This message inserts the specified page into the notebook data list.

Parameters

param1

ulPageId (ULONG)
Page ID for placement.

Page identifier used for the placement of the inserted page. This identifier is ignored if the BKA_FIRST or BKA_LAST attribute of the *usPageOrder* parameter is specified.

param2

usPageStyle (USHORT)
Style attributes.

Attributes that specify the style to be used for an inserted page. You can specify one attribute from each of the following groups by using logical OR operators (|) to combine attributes.

- Specify the following for automatic page position and size:

BKA_AUTOPAGESIZE	Notebook handles the positioning and sizing of the application page window specified in the BKM_SETPAGEWINDOWHWND message.

- Specify the following to display status area text:

BKA_STATUSTEXTON
Page is to be displayed with status area text. If this attribute is not specified, the application cannot associate a text string with the status area of the page being inserted.

- Specify one of the following if the page is to have a major or minor tab attribute:

BKA_MAJOR	Inserted page will have a major tab attribute.
BKA_MINOR	Inserted page will have a minor tab attribute.

usPageOrder (USHORT)
Order attributes.

Placement of page relative to the previously inserted pages. You can specify one of the following attributes:

BKA_FIRST	Insert page at the front of the notebook. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.
BKA_LAST	Insert page at the end of the notebook. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.
BKA_NEXT	Insert page after the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, NULL is returned and no page is inserted.
BKA_PREV	Insert page before the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, NULL is returned and no page is inserted.

Returns

ulPageId (ULONG)
Page ID for insertion.

NULL	The page was not inserted into the notebook. An invalid page ID was specified for the *ulPageId* parameter for *param1* or not enough space was available to allocate the page data.
Other	Identifier for the inserted page.

BKM_INVALIDATETABS

This message repaints all of the tabs in the notebook.

Parameters

param1

ulReserved (ULONG)
Reserved value, should be 0.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Tabs painted successfully.
FALSE	Tabs were not painted.

BKM_QUERYPAGECOUNT

This message queries the number of pages.

Parameters

param1

ulPageId (ULONG)
Page ID or 0.

Page identifier from which to start the query, or 0. If this parameter is set to 0, the query begins with the first page.

param2

usQueryEnd (USHORT)
Query end attribute.

Attribute that ends the page count query.

BKA_MAJOR	Query the number of pages between the page ID specified in the *ulPageId* parameter and the next page that has the BKA_MAJOR attribute. The page that has the BKA_MAJOR attribute is not included in the page count.
BKA_MINOR	Query the number of pages between the page ID specified in the *ulPageId* parameter and the next page that has the BKA_MINOR attribute. The page that has the BKA_MINOR attribute is not included in the page count.
BKA_END	Query the number of pages between the page ID specified in the *ulPageId* parameter and the last page. When this attribute is specified, the page count includes the last page plus the notebook's back cover.

Returns

pageCount (SHORT)
Number of pages.

BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the *ulPageId* parameter.
Other	Number of pages for the specified range. If the notebook is empty or no pages are found in the range, this value is 0.

BKM_QUERYPAGEDATA

This message queries the 4 bytes of application reserved storage associated with the specified page.

Parameters

param1

ulPageId (ULONG)
Page ID.

The page identifier of the page from which to retrieve the 4 bytes of data.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulPageData (ULONG)
Page data.

BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the *ulPageId* parameter.
0	No page data was set for the page specified in the *ulPageId* parameter.
Other	Application-defined page data.

BKM_QUERYPAGEID

This message queries the page identifier for the specified page.

Parameters

param1

ulPageId (ULONG)
Location page ID.

Page identifier used for locating the requested page. This identifier is ignored if the BKA_FIRST, BKA_LAST, or BKA_TOP attribute is specified.

param2

usQueryOrder (USHORT)
Page ID query order.

Order in which to query the page identifier.

BKA_FIRST	Get the page identifier for the first page. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.
BKA_LAST	Get the page identifier for the last page. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.
BKA_NEXT	Get the page identifier for the page after the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, BOOKERR_INVALID_PARAMETERS is returned.
BKA_PREV	Get the page identifier for the page before the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, BOOKERR_INVALID_PARAMETERS is returned.
BKA_TOP	Get the page identifier for the page currently visible in the notebook. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.

usPageStyle (USHORT)
Page style.

Page style for which to query the page identifier. If neither of these attributes is specified, the *usPageStyle* parameter is ignored.

BKA_MAJOR	Query page with major tab attribute.
BKA_MINOR	Query page with minor tab attribute. If a major tab page is found before the minor tab page, the search is ended and 0 is returned.

Returns

ulPageId (ULONG)
Retrieved page ID.

BOOKERR_INVALID_PARAMETERS	Returned if the page ID specified for the *ulPageId* parameter for *param1* is invalid when specifying either the BKA_PREV or BKA_NEXT attribute in the *usQueryOrder* parameter.
0	Requested page not found. This could be an indication that the end or front of the list has been reached, or that the notebook is empty.
Other	Retrieved page identifier.

BKM_QUERYPAGEINFO

This message queries the page information associated with a notebook page.

Parameters

param1

ulPageId (ULONG)
Id of the notebook page whose information is to be queried.

param2

pPageInfo (PPAGEINFO)
Pointer to a notebook page information structure.

Returns

returns

rc (BOOL)
Success indicator.

Possible values are described in the following list:

TRUE	Message was processed.
FALSE	Message was ignored.

BKM_QUERYPAGESTYLE

This message queries the style that was set when the specified page was inserted.

Parameters

param1

ulPageId (ULONG)
Page ID.

Page identifier of the page from which to query the style setting.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

usPageStyle (USHORT)
Page style data.

BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the *ulPageId* parameter.
Other	Page style data.

BKM_QUERYPAGEWINDOWHWND

This message queries the application page window handle associated with the specified page.

Parameters

param1

ulPageId (ULONG)
Page ID.

Page identifier of the page whose window handle is requested.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

hwndPage (HWND)
Window handle.

BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the *ulPageId* parameter.
NULLHANDLE	No application page window handle is associated for the page specified in the *ulPageId* parameter.
Other	Handle of the application page window associated with the specified page identifier.

BKM_QUERYSTATUSLINETEXT

This message queries the status line text, text size, or both for the specified page.

Parameters

param1

ulPageId (ULONG)
Page ID.

Page identifier of the page whose status line text is requested.

param2

pBookText (PBOOKTEXT)
Pointer to a BOOKTEXT data structure. See "BOOKTEXT" on page 9-51 for definitions of this structure's fields as they apply to the BKM_QUERYSTATUSLINETEXT message.

Returns

statusTextLen (USHORT)
String length.

BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the *ulPageId* parameter or the structure specified for the *pBookText* parameter is invalid.
0	No text data has been set (BKM_SETSTATUSLINETEXT) for the page specified in the *ulPageId* parameter.
Other	Length of the returned status line text string.

BKM_QUERYTABBITMAP

This message queries the bit-map handle associated with the specified page.

Parameters

param1

ulPageId (ULONG)
Page ID.

Page identifier of the page whose bit-map handle is requested. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

hbm (HBITMAP)
Bit-map handle.

BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the *ulPageld* parameter.
NULLHANDLE	No bit-map handle is associated with the page specified in the *ulPageld* parameter.
Other	Handle of the bit map associated with the specified page identifier.

BKM_QUERYTABTEXT

This message queries the text, text size, or both for the specified page.

Parameters

param1

ulPageId (ULONG)
Page ID.

Page identifier of the page whose tab text is requested. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

param2

pBookText (PBOOKTEXT)
Pointer to a BOOKTEXT data structure.

See "BOOKTEXT" on page 9-51 for definitions of this structure's fields as they apply to the BKM_QUERYTABTEXT message.

Returns

tabTextLen (USHORT)
Length of the tab text string.

BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the *ulPageId* parameter or the structure specified for the *pBookText* parameter is invalid.
0	No text data has been set (BKM_SETTABTEXT) for the page specified in the *ulPageId* parameter.
Other	Length of the returned tab text string.

BKM_SETDIMENSIONS

This message sets the height and width for the major tabs, minor tabs, or page buttons.

Parameters

param1

usWidth (USHORT)
Width value to set.

usHeight (USHORT)
Height value to set.

param2

usType (USHORT)
Notebook region.

Notebook region for which the dimensions are to be set. Valid values are:

- BKA_MAJORTAB
- BKA_MINORTAB
- BKA_PAGEBUTTON.

Returns

rc (BOOL)
Success indicator.

TRUE Dimensions were successfully set.
FALSE Unable to set dimensions. Returned if an invalid value is specified for the *usType* parameter or if the dimensions are invalid.

BKM_SETNOTEBOOKCOLORS

This message sets the colors for the major tab text and background, the minor tab text and background, and the notebook page background.

Parameters

param1

ulColor (ULONG)
Color value to set.

param2

usBookAttr (USHORT)
Notebook region.

Notebook region whose color is to be set. Valid values are:

BKA_BACKGROUNDPAGECOLOR or BKA_BACKGROUNDPAGECOLORINDEX
Page background. This color is initially set to SYSCLR_PAGEBACKGROUND.

BKA_BACKGROUNDMAJORCOLOR or BKA_BACKGROUNDMAJORCOLORINDEX
Major tab background. This color is initially set to SYSCLR_PAGEBACKGROUND.

BKA_BACKGROUNDMINORCOLOR or BKA_BACKGROUNDMINORCOLORINDEX
Minor tab background. This color is initially set to SYSCLR_PAGEBACKGROUND.

BKA_FOREGROUNDMAJORCOLOR or BKA_FOREGROUNDMAJORCOLORINDEX
Major tab text. This color is initially set to SYSCLR_WINDOWTEXT.

BKA_FOREGROUNDMINORCOLOR or BKA_FOREGROUNDMINORCOLORINDEX
Minor tab text. This color is initially set to SYSCLR_WINDOWTEXT.

Returns

rc (BOOL)
Success indicator.

TRUE	Colors were successfully set.
FALSE	Unable to set colors. Returned if an invalid notebook attribute is specified for the *usBookAttr* parameter.

BKM_SETPAGEDATA

This message sets the 4 bytes of application reserved storage associated with the specified page.

Parameters

param1

ulPageId (ULONG)
Page ID.

The page identifier of the page from which to set the 4 bytes of data.

param2

ulPageData (ULONG)
Page data.

Application-defined page data.

Returns

rc (BOOL)
Success indicator.

TRUE	Page data was successfully set.
FALSE	Unable to set page data. This value is returned if the page ID specified in the *ulPageId* parameter is invalid.

BKM_SETPAGEINFO

This message sets the page information associated with notebook page which contains a single message.

Parameters

param1

ulPageId (ULONG)
Id of the notebook page whose information is to be set.

param2

pPageInfo (PPAGEINFO)
Pointer to a notebook page information structure.

Returns

returns

rc (BOOL)
Success indicator.

Possible values are described in the following list:

TRUE	Message was processed.
FALSE	Message was ignored.

BKM_SETPAGEWINDOWHWND

This message associates an application page window handle with the specified notebook page.

Parameters

param1

ulPageId (ULONG)
Page ID.

The page ID of the notebook page with which the application page window is to be associated.

param2

hwndPage (HWND)
Window handle.

The handle of the application page window that is to be associated with the notebook page identified in the *ulPageId* parameter.

Returns

rc (BOOL)
Success indicator.

TRUE	Application page window handle was successfully set.
FALSE	Unable to set application page window handle. This value is returned if the page ID specified for the *ulPageId* parameter is invalid.

BKM_SETSTATUSLINETEXT

This message associates a text string with the specified page's status line.

Parameters

param1

ulPageId (ULONG)
Page ID.

The page identifier with which to associate the text string.

param2

pString (PSZ)
Pointer to a text string that ends in a null character.

Returns

rc (BOOL)
Success indicator.

TRUE	Status line text was successfully set.
FALSE	Unable to set status line text. This value is returned if the page ID specified in the *ulPageId* parameter is invalid or if the page was inserted without specifying the BKA_STATUSTEXTON attribute.

BKM_SETTABBITMAP

This message associates a bit-map handle with the specified page.

Parameters

param1

ulPageId (ULONG)
Page ID.

The page identifier with which to associate the bit-map handle. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

param2

hbm (HBITMAP)
Bit-map handle.

Returns

rc (BOOL)
Success indicator.

TRUE Tab bit map was successfully set.

FALSE Unable to set tab bit map. If the page ID specified in the *ulPageId* parameter is invalid or if it identifies a page that does not have a BKA_MAJOR or BKA_MINOR attribute, FALSE is returned and no bit map is associated with the page.

BKM_SETTABTEXT

This message associates a text string with the specified page.

Parameters

param1

ulPageId (ULONG)
Page ID.

The page identifier with which to associate the text string. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

param2

pString (PSZ)
Pointer to a text string that ends with a null character.

Returns

rc (BOOL)
Success indicator.

TRUE	Tab text was successfully set.
FALSE	Unable to set tab text. If the page ID specified in the *ulPageId* parameter is invalid or if it identifies a page that does not have a BKA_MAJOR or BKA_MINOR attribute, FALSE is returned and no text string is associated with the page.

BKM_TURNTOPAGE

This message brings the specified page to the top of the notebook.

Parameters

param1

ulPageId (ULONG)
Page ID.

The page identifier that is to become the top page.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

fSuccess (BOOL)
Success indicator.

TRUE	The page was successfully moved to the top of the notebook.
FALSE	Unable to move the page to the top of the notebook. This value is returned if the page ID specified in the *ulPageId* parameter is invalid.

WM_PRESPARAMCHANGED (in Notebook Controls)

For the cause of this message, see WM_PRESPARAMCHANGED.

Parameters

param1

attrtype (ULONG)
Attribute type.

Presentation parameter attribute identity.

PP_BACKGROUNDCOLOR or PP_BACKGROUNDCOLORINDEX
Sets the background color of the notebook window. This color is initially set to SYSCLR_FIELDBACKGROUND.

PP_BORDERCOLOR or PP_BORDERCOLORINDEX
Sets the color of the notebook outline. This color is initially set to SYSCLR_WINDOWFRAME.

PP_FOREGROUNDCOLOR or PP_FOREGROUNDCOLORINDEX
Sets the color of text on the status line. This color is initially set to SYSCLR_WINDOWTEXT.

PP_HILITEBACKGROUNDCOLOR or PP_HILITEBACKGROUNDCOLORINDEX
Sets the color of the selection cursor. This color is initially set to SYSCLR_HILITEBACKGROUND.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

Related Notification Messages

This section covers the notification messages that are related to notebook controls.

WM_CONTROL (in Notebook Controls)

For the cause of this message, see WM_CONTROL.

Parameters

param1

id (USHORT)
Control-window identity.

notifycode (USHORT)
Notify code.

The notebook control uses these notification codes:

BKN_HELP	Indicates the notebook control has received a WM_HELP message.
BKN_NEWPAGESIZE	Indicates the dimensions of the application page window have changed.
BKN_PAGEDELETED	Indicates a page has been deleted from the notebook.
BKN_PAGESELECTED	Indicates a new page has been brought to the top of the notebook. This notification is sent after the page is turned.
BKN_PAGESELECTEDPENDING	Indicates a new page is about to be brought to the top of the notebook. This notification is sent before the page is actually turned. If the application does not want the page to be turned, it sets the *ulPageIdNew* field of the PAGESELECTNOTIFY structure to NULL before returning.

param2

notifyinfo (ULONG)
Notify code information.

The value of this parameter depends on the value of the *notifycode* parameter. When the value of the *notifycode* parameter is BKN_HELP, this parameter is the ID of the notebook page (*ulPageId*) whose tab contains the selection cursor.

When the value of the *notifycode* parameter is BKN_PAGESELECTED or BKN_PAGESELECTEDPENDING, this parameter is a pointer to the PAGESELECTNOTIFY structure.

When the value of the *notifycode* parameter is BKN_PAGEDELETED, this parameter is a pointer to the DELETENOTIFY structure.

Otherwise, this parameter is the notebook control window handle.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DRAWITEM (in Notebook Controls)

This notification message is sent to the owner of a notebook control each time a tab's content is to be drawn by the owner of the notebook. The tab's content is drawn by the owner unless the owner sets the tab text or bit map by sending a BKM_SETTABTEXT or BKM_SETTABBITMAP message, respectively, to the notebook control.

Parameters

param1

id (USHORT)
Window identifier.

The window identifier of the notebook control sending this notification message.

param2

powneritem (POWNERITEM)
Pointer to an OWNERITEM data structure.

The following list defines the OWNERITEM data structure fields that apply to the notebook control. See "OWNERITEM" on page 8-111 for the default field values.

hwnd (HWND)
Notebook window handle.

hps (HPS)
Presentation-space handle.

fsState (ULONG)
Notebook window style flags. See "Notebook Styles" on page 9-1 for descriptions of these style flags.

fsAttribute (ULONG)
Page attribute flags for the tab page. See BKM_INSERTPAGE for descriptions of these attribute flags.

fsStateOld (ULONG)
Reserved.

fsAttributeOld (ULONG)
Reserved.

rclItem (RECTL)
Tab rectangle to be drawn in window coordinates.

idItem (LONG)
Reserved.

hItem (ULONG)
Current page ID (*ulPageId*) for which the content of a tab is to be drawn.

Returns

rc (BOOL)
Content-drawn indicator.

TRUE	The owner draws the tab's content.
FALSE	If the owner does not draw the tab's content, the owner returns this value and the notebook control draws the tab's content.

Related Data Structures

This section covers the data structures that are related to notebook controls.

BOOKTEXT

Notebook data structure that contains text strings for notebook status lines and tabs. This data structure is used with the BKM_QUERYSTATUSLINETEXT and the BKM_QUERYTABTEXT messages only. See "BKM_QUERYSTATUSLINETEXT" on page 9-38 and "BKM_QUERYTABTEXT" on page 9-39 for information about those messages.

Syntax

```
typedef struct _BOOKTEXT {
PSZ         pString;
ULONG       textLen;
 } BOOKTEXT;

typedef BOOKTEXT *PBOOKTEXT;
```

Fields

pString (PSZ)
Pointer to a string buffer.

Buffer in which the text string is to be placed. For the BKM_QUERYSTATUSLINETEXT message, this is the buffer in which the status line text is placed.

For the BKM_QUERYTABTEXT message, this is the buffer in which the tab text is placed.

textLen (ULONG)
String length.

Length of the text string. For the BKM_QUERYSTATUSLINETEXT message, this is the length of the status line text string.

For the BKM_QUERYTABTEXT message, this is the length of the tab text string.

DELETENOTIFY

Structure that contains information about the application page that is being deleted from a notebook.

Syntax

```
typedef struct _DELETENOTIFY {
HWND          hwndBook;
HWND          hwndPage;
ULONG         ulAppPageData;
HBITMAP       hbmTab;
 } DELETENOTIFY;

typedef DELETENOTIFY *PDELETENOTIFY;
```

Fields

hwndBook (HWND)
Notebook window handle.

hwndPage (HWND)
Application page window handle.

ulAppPageData (ULONG)
Application-specified page data.

hbmTab (HBITMAP)
Application-specified tab bit map.

PAGESELECTNOTIFY

Structure that contains information about the application page being selected.

Syntax

```
typedef struct _PAGESELECTNOTIFY {
HWND        hwndBook;
ULONG       ulPageIdCur;
ULONG       ulPageIdNew;
 } PAGESELECTNOTIFY;

typedef PAGESELECTNOTIFY *PPAGESELECTNOTIFY;
```

Fields

hwndBook (HWND)
Notebook window handle.

ulPageIdCur (ULONG)
Current top page identifier.

ulPageIdNew (ULONG)
New top page identifier.

Summary

Following are tables that describe the OS/2 functions, window messages, notification messages, notification codes, and data structures used with notebook controls:

Table 9-2. Notebook Control Functions

Function Name	Description
WinInvalidateRect	Adds a rectangle to a window's update region.
WinSetPresParam	Sets a presentation parameter for a window.

Table 9-3 (Page 1 of 2). Notebook Control Window Messages

Message Name	Description
BKM_CALCPAGERECT	Calculates a window rectangle from a notebook rectangle or a notebook rectangle from a window rectangle, depending on the setting of the *fPage* parameter.
BKM_DELETEPAGE	Deletes the specified page or pages from the notebook data list.
BKM_INSERTPAGE	Inserts the specified page into the notebook data list.
BKM_INVALIDATETABS	Repaints all the tabs in the notebook.
BKM_QUERYPAGECOUNT	Queries the number of pages.
BKM_QUERYPAGEDATA	Queries the 4 bytes of application-reserved storage associated with the specified page.
BKM_QUERYPAGEID	Queries the page identifier for the specified page.
BKM_QUERYPAGEINFO	Queries any of the page information associated with a notebook page.
BKM_QUERYPAGESTYLE	Queries the style that was set when the specified page was inserted.
BKM_QUERYPAGEWINDOWHWND	Queries the notebook page window handle associated with the specified page.
BKM_QUERYSTATUSLINETEXT	Queries the status-line text, text size, or both, for the specified page.
BKM_QUERYTABBITMAP	Queries the bit-map handle associated with the specified page. If this message is sent for a page having both a major and a minor attribute, the notebook returns the bitmap that is associated with the major tab.
BKM_QUERYTABTEXT	Queries the text, text size, or both, for the specified page. If this message is sent for a page having both a major and a minor attribute, the notebook returns the text that is associated with the major tab.
BKM_SETDIMENSIONS	Sets the height and width for the major tabs, minor tabs, or page buttons.

Table 9-3 (Page 2 of 2). Notebook Control Window Messages	
Message Name	**Description**
BKM_SETNOTEBOOKCOLORS	Sets the colors for the major tab text and background, minor tab text and background, and notebook page background.
BKM_SETPAGEDATA	Sets the 4 bytes of application-reserved storage associated with the specific page.
BKM_SETPAGEINFO	Allows an application to set any of the page information associated with a page in the notebook which contains a single message.
BKM_SETPAGEWINDOWHWND	Associates a notebook page window handle with the specified notebook page.
BKM_SETSTATUSLINETEXT	Associates a text string with the status line on the specified page.
BKM_SETTABBITMAP	Associates a bitmap handle with the specified page. If this message is sent for a page having both a major and a minor tab attribute, the notebook sets both the major and minor tab bitmap to be the bitmap that is passed in.
BKM_SETTABTEXT	Associates a text string with the specified page. If this message is sent for a page having both a major and a minor attribute, the notebook sets both the major and minor tab text to be the text that is passed in.
BKM_TURNTOPAGE	Brings the specified page to the top of the notebook.
WM_CHAR	Occurs when the user presses a key.
WM_PRESPARAMCHANGED	Occurs when a presentation parameter is set or removed dynamically from a window instance.
WM_SIZE	Occurs when the size of the notebook window changes.

Table 9-4. Notebook Control Notification Messages	
Message Name	**Description**
WM_CONTROL	Occurs when a control has a significant event to notify to its owner.
WM_CONTROLPOINTER	Sent to the notebook control's owner window when the pointing device pointer moves over the notebook control window, thereby enabling the owner to set the pointing device pointer.
WM_DRAWITEM	Sent to the owner of the notebook control each time an item is to be drawn.

Table 9-5. Notebook Control Notification Codes

Code Name	Description
BKN_HELP	Indicates that the notebook control has received a WM_HELP message.
BKN_NEWPAGESIZE	Indicates that the dimensions of the notebook page window have changed.
BKN_PAGEDELETED	Indicates that a page has been deleted from the notebook.
BKN_PAGESELECTED	Indicates that a new page has been brought to the top of the notebook.
BKN_PAGESELECTEDPENDING	Indicates that a new page is about to be brought to the top of the notebook.

Table 9-6. Notebook Control Data Structures

Data Structure Name	Description
BOOKTEXT	Contains text strings for notebook status lines and tabs.
DELETENOTIFY	Contains information about the page being deleted from a notebook.
PAGESELECTNOTIFY	Contains information about the page being selected in a notebook.
PPAGEINFO	Contains a pointer to the notebook page information data structure.

Chapter 10. File Dialog Controls

File dialog controls provide basic functions that enable users to do the following:

- Display and select from a list of drives, directories, and files
- Enter a file name directly
- Filter the file names before they are displayed
- Display active network connections
- Specify .TYPE EA extended attributes
- Interact with a single-selection or multiple-selection file dialog
- Interact with a modal or modeless file dialog.

These basic functions can be extended to meet the requirements of PM applications.

About File Dialog Controls

The file dialog control enables you to implement *Open* or *SaveAs* dialogs. Figure 10-1 and Figure 10-2 on page 10-2 illustrate an example of these two dialogs.

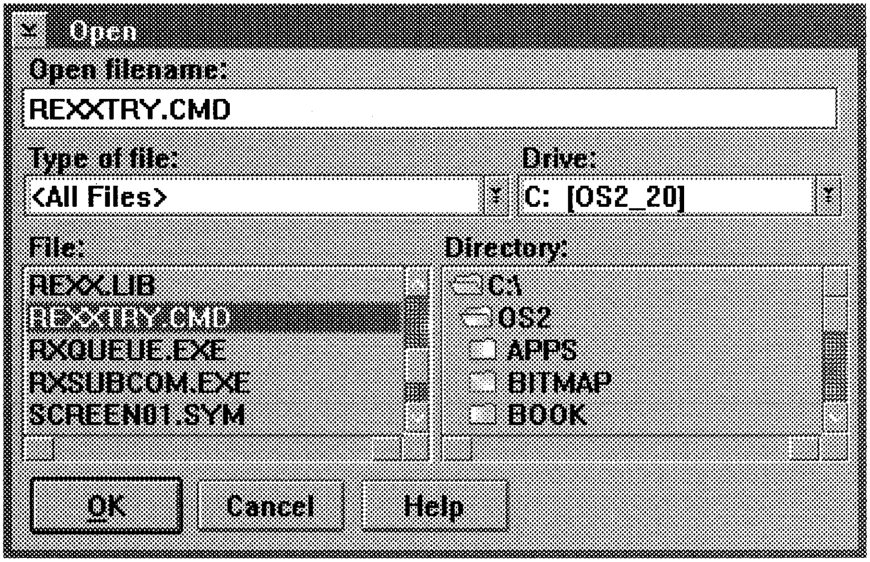

Figure 10-1. Open Dialog Example

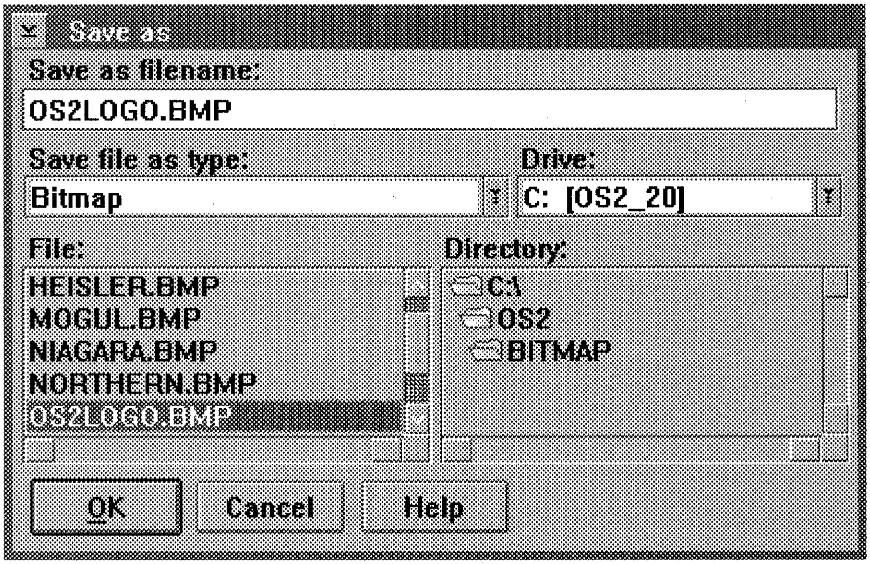

Figure 10-2. SaveAs Dialog Example

Customizing the File Dialog

You can customize the File Dialog control by using the standard controls and adding any of your own design. Specify a standard control by including the control name, ID, and style in the dialog. The standard control list is included in "Summary" on page 10-16.

Using File Dialog Controls

This section describes how to create:

- A file dialog
- An Open dialog
- A SaveAs dialog.

Creating a File Dialog

To present a file dialog to users, your application must do the following:

1. Allocate storage for a FILEDLG data structure and set all fields to NULL.
2. Initialize the fields in the FILEDLG data structure.

   The application must do the following:

   a. Set the *cbSize* field to the size of the data structure.

   b. Set the *fl* field to indicate the type of dialog. You must set the FDS_OPEN_DIALOG or FDS_SAVEAS_DIALOG flags.

The application can set the following:

a. An application-specific title. Pass the pointer to a null-terminated string in the *pszTitle* field.

b. An application-specific text for the *OK* push button. Pass the pointer to a null-terminated string in the *pszOKButton* field.

c. A custom dialog procedure to provide application-specific function. Pass the pointer to a window procedure in the *pfnDlgProc* field.

d. Set other FDS_* flags in the *fl* field to customize the dialog style.

e. Pass the initial position of the dialog in the *x* and *y* fields.

3. Initialize the FILEDLG data structure with any values that users should see when they invoke the dialog for the first time. For example, you can:

   a. Pass the name of the first drive from which file information will be displayed in the *pszIDrive* field.

   b. If you want to limit user selections, pass a list of drives from which the user can choose in the *papszIDriveList* field. Otherwise, the system defaults to showing all available drives.

   c. Pass the name of an extended-attribute filter to be used to filter file information in the *pszIType* field.

   d. Pass a list of extended attributes in the *papszITypeList* field. By selecting from this list, users can filter file information.

   e. Pass the name of the initial file to be used by the dialog in the *szFullFile* field. This can be a file name or a string filter, such as *.dat, to filter the initial file information. This field can be fully qualified to select the initial drive and directory.

4. Invoke the file dialog. Call WinFileDlg and pass the dialog's owner window handle and a pointer to the initialized FILEDLG data structure.
5. Verify the return value from WinFileDlg. If it is successful, the application can create the file dialog (either Open or SaveAs) by using the file name or file names returned from the dialog.

Creating an Open Dialog

When the Open dialog is invoked, the fields in the dialog box are updated with the fields passed in the FILEDLG data structure. The values passed in the *szFullFile* field of the data structure are displayed in the *File Name* field, the Directory list box, and the *Drive* field. The value passed in the *pszIType* field is displayed in the *Type* field.

Creating a SaveAs Dialog

The SaveAs dialog is identical to the Open dialog with these exceptions:

- By default, the file names in the file list box are grayed and cannot be selected, although the list box can be scrolled.
- When the user clicks on the *OK* push button or presses the Enter key, the file name in the *File Name* field is passed to the application, and the application saves, rather than opens, the file.
- The titles of the file name, filter, and dialog are SaveAs rather than Open.

Graphical User Interface Support for File Dialog Controls

This section provides information about the file dialog user interface.

Name Field

The *File Name* field is a single-line entry (SLE) field used to display the name of a file that was selected from the file list box or entered directly by the user. As the user types, the file or files matching the user entry are scrolled into view in the file list box. The first file name that most closely matches the file name typed by the user is placed at the top of the list box. When the user types a character that causes a mismatch, the file at the top of the list is displayed.

When the user presses the Enter key, the dialog returns the selected file name to the application. The application then initiates the default action of opening the file. When a file name is not valid, such as when the file does not exist, the application displays an error message.

The *File Name* field displays the currently selected file name or the current string filter. When a filter is specified in the *szFullFile* field of the FILEDLG data structure, the string filter is displayed without the path information. The string filter remains in the field until a file is selected or the user types over the data in the field.

When a file name is not specified, the *File Name* field is blank.

File List Box

The File list box is a single- or multiple-selection list box that is scrollable both horizontally and vertically. It contains all the files that meet the filter criteria, sorted by name.

When the file dialog is a single-selection dialog, the selected file name is placed in the *File Name* field. When the file dialog is a multiple-selection dialog, the topmost selected file name is placed in the *File Name* field. When the user double-clicks on a file name, the dialog exits and returns the selected file or files to the application for opening.

Directory List Box

The Directory list box is a single-selection list box that is scrollable both horizontally and vertically.

The Directory list box displays the path in the *szFullFile* field of the FILEDLG data structure as a list of each parent subdirectory. Any subdirectories of the selected directory also are displayed. Each directory level is indented to show the path, and the current working directory level is indicated by an arrow. The top entry is always the root directory, with the drive specification preceding it. When the *szFullFile* field is NULL, the current path of the current drive is displayed. The user selects a new subdirectory by double-clicking on the subdirectory name. This action updates the Directory list box.

Drive Field

The *Drive* field contains a drop-down list of the logical drives. This field cannot be edited by the user.

The *Drive* field displays the value passed in the *papszIDriveList* field of the FILEDLG data structure. If the application does not specify a drive list, all drives currently available on the system are displayed. When the drop-down list is displayed, the current drive is highlighted. When the user selects a drive, the display is refreshed. When either the user-specified drive or the default drive has a volume label, the volume label is displayed also.

Users can access networked files by associating logical disks with remote servers, or they can enter the name and ID of the server in the *File Name* field. When the server name entered is not found in the Drive drop-down list, it is added to the list and displayed in the *Drive* field.

Type Field

The *Type* field contains a drop-down list of extended-attribute filters.

The *Type* field displays the value passed in the *pszIType* field of the FILEDLG data structure. The current setting is highlighted when the drop-down list is displayed.

When a type filter is not specified by the application, `<All Files>` is displayed and no extended-attribute type filtering is used with the initial display.

All files affected by the string filter and the extended-attribute type filter criteria are displayed, based on how the filters are to be used. The default is that all file names meeting the intersection of the two filters are shown. When users change the value in the *Type* field, the File list box is updated to display a list of files that meet the new type filter criteria. Files that meet both the string filter and extended-attribute type filter are displayed.

Standard Push Button and Default Action

The *OK* push button initiates the default action.

When a subdirectory is selected, the *File Name* field is empty. When the user clicks on the *OK* push button or presses the Enter key, the subdirectory is opened and the displayed values in the File list box and the Directory list box are refreshed.

When a file name is selected, selection of subdirectories is canceled and the *File Name* field is updated with the name of the selected file. When the user clicks on the *OK* push button or presses the Enter key, the file displayed in the *File Name* field is returned to the application for opening.

Subclassing the Default File Dialog Procedure

The name of the dialog procedure is assigned to the *pfnDlgProc* field of the FILEDLG data structure.

Related Functions

This section covers the functions that are related to file dialog controls.

WinDefFileDlgProc

This function is the default dialog procedure for the file dialog.

Syntax

```
#define INCL_winstdfile

#include <os2.h>

MRESULT WinDefFileDlgProc  (HWND hwnd, ULONG msg, MPARAM mp1,
                            MPARAM mp2)
```

Parameters

hwnd (HWND) – input
Dialog-window handle.

msg (ULONG) – input
Message identity.

mp1 (MPARAM) – input
Parameter 1.

mp2 (MPARAM) – input
Parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

WinFileDlg

This function creates and displays the file dialog and returns the user's selection or selections.

Syntax

```
#define INCL_winstdfile

#include <os2.h>

HWND WinFileDlg  (HWND hwndP, HWND hwndO, PFILEDLG pfild)
```

Parameters

hwndP (HWND) – input
  Parent-window handle.

HWND_DESKTOP	The desktop window.
Other	Specified window.

hwndO (HWND) – input
  Requested owner-window handle.

pfild (PFILEDLG) – input
  Pointer to a FILEDLG structure.

Returns

hwndDlg (HWND) – returns
  File dialog window handle.

WinFreeFileDlgList

This function frees the storage allocated by the file dialog when the FDS_MULTIPLESEL dialog flag is set.

Syntax

```
#define INCL_winstdfile

#include <os2.h>

BOOL WinFreeFileDlgList  (PAPSZ  papszFQFilename)
```

Parameters

papszFQFilename (PAPSZ) – input
  Pointer to a table of pointers of fully-qualified file names returned by the dialog.

Returns

rc (BOOL) – returns
  Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Related Window Messages

This section covers the window messages that are related to file dialog controls.

FDM_ERROR

This message is sent whenever the file dialog is going to display an error message window. This allows an application to display its own message, if desired, instead of messages provided by the system.

Parameters

param1

usErrorId (USHORT)
Error message ID.

This is the ID of the message that is displayed by the file dialog if the default file dialog procedure processes the message.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

usUserReply (USHORT)
User's reply.

0	The file dialog presents the error message for this ID.
MBID_OK	The file dialog processes the reply as if the OK push button was pressed in its message window.
MBID_CANCEL	The file dialog processes the reply as if the Cancel push button was pressed in its message window.
MBID_RETRY	The file dialog processes the reply as if the Retry push button was pressed in its message window.

FDM_FILTER

This message is sent before a file that meets the current filter criteria is added to the File list box.

Parameters

param1

pFilename (PSZ)
Pointer to the file name.

param2

pEAType (PSZ)
Pointer to the .TYPE EA extended attribute.

Returns

rc (BOOL)
Success indicator.

TRUE	Add the file.
FALSE	Do not add the file.

FDM_VALIDATE

This message is sent when the user selects a file and presses Enter or clicks on the OK button, or double-clicks on a file name in the file list box.

Parameters

param1

pFileName (PSZ)
Pointer to the fully-qualified file name.

param2

usSeltype (USHORT)
Selection type.

rc (BOOL)
Validity indicator.

TRUE	File name is valid.
FALSE	File name is not valid.

Related Data Structure

This section covers the data structure that is related to file dialog controls.

FILEDLG

File-dialog structure.

Syntax

```
typedef struct _FILEDLG {
ULONG         cbSize;
ULONG         fl;
ULONG         ulUser;
LONG          lReturn;
LONG          lSRC;
PSZ           pszTitle;
PSZ           pszOKButton;
PFNWP         pfnDlgProc;
PSZ           pszIType;
PAPSZ         papszITypeList;
PSZ           pszIDrive;
PAPSZ         papszIDriveList;
HMODULE       hMod;
CHAR          szFullFile[CCHMAXPATH];
PAPSZ         papszFQFilename;
ULONG         ulFQFCount;
USHORT        usDlgID;
SHORT         x;
SHORT         y;
SHORT         sEAType;
 } FILEDLG;

typedef FILEDLG *PFILEDLG;
```

Fields

cbSize (ULONG)
Structure size.

Size of the structure. This field allows future expansion of the structure and must be initialized with the size of the FILEDLG structure.

fl (ULONG)
FDS_* flags.

Several flags can be specified to alter the behavior of the dialog.

Note: The dialog must be either an "Open" or a "Save As" dialog. If neither the FDS_OPEN_DIALOG nor the FDS_SAVEAS_DIALOG flag is set, or if both are set, the dialog will return an error.

FDS_APPLYBUTTON	An Apply push button is added to the dialog. This is useful in a modeless dialog.
FDS_CENTER	The dialog is positioned in the center of its parent window, overriding any specified *x*, *y* position.
FDS_CUSTOM	A custom dialog template is used to create the dialog. The *hMod* and *usDlgID* fields must be initialized.
FDS_ENABLEFILELB	When this flag is set, the Files list box on a Save As dialog is enabled. When this flag is not set, the Files list box is not enabled for a Save As dialog. This is the default.
FDS_FILTERUNION	When this flag is set, the dialog uses the union of the string filter and the extended-attribute type filter when filtering files for the Files list box. When this flag is not set, the list box, by default, uses the intersection of the two.
FDS_HELPBUTTON	A Help push button of style (BS_HELP|BS_NOPOINTERFOCUS) with an ID of DID_HELP_PB is added to the dialog. When this push button is pressed, a WM_HELP message is sent to *hwndO*.
FDS_INCLUDE_EAS	If this flag is set, the dialog will always query extended attribute information for files as it fills the Files list box. The default is to not query the information unless an extended attribute type filter has been selected.
FDS_MODELESS	When this flag is set, the dialog is modeless; WinFileDlg returns immediately after creating the dialog window and returns the window handle to the application. The application should treat the dialog as if it were created with WinLoadDlg. As in the modal (default) dialog case, the return value is found in the *lReturn* field of the FILEDLG structure passed to WinFileDlg.
FDS_MULTIPLESEL	When this flag is set, the Files list box for the dialog is a multiple selection list box. When this flag is not set, the default is a single-selection list box.
FDS_OPEN_DIALOG	The dialog is an “Open” dialog when this flag is set.
FDS_PRELOAD_VOLINFO	If this flag is set, the dialog will preload the volume information for the drives and will preset the current default directory for each drive. The default behavior is for the volume label to be blank and the initial directory will be the root directory for each drive.
FDS_SAVEAS_DIALOG	The dialog is a “Save As” dialog when this flag is set.

ulUser (ULONG)
Used by the application.

This field can be used by an application that is subclassing the file dialog to store its own state information.

lReturn (LONG)
Result code.

Result code from dialog dismissal. This field contains the ID of the push button pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplies additional push buttons in its template. If an error occurs on dialog invocation, this field is set to zero.

lSRC (LONG)
System return code.

This field contains an FDS_ERR return code. When a dialog fails, this field is used to tell the application the reason for the failure.

pszTitle (PSZ)
Dialog title string.

When this field is NULL, the dialog title defaults to the name of the dialog currently running.

pszOKButton (PSZ)
OK push button text.

This string is used to set the text of the OK push button. The default text is OK.

pfnDlgProc (PFNWP)
Custom dialog procedure.

NULL unless the caller is subclassing the file dialog. When non-NULL, it points to the dialog procedure of the application.

pszIType (PSZ)
Extended-attribute type filter.

This field contains a pointer to the initial extended-attribute type filter that is applied to the initial dialog screen. This filter is not required to be in *papszITypeList*.

papszITypeList (PAPSZ)
Pointer to a table of pointers to extended-attribute types.

Each pointer in the table points to a null-terminated string, and each string is an extended-attribute type. These types are sorted in ascending order in the Type drop-down box. The end of the table is marked by a null pointer. To specify an empty table, the application sets this field to NULL, or it specifies a table containing only a null pointer.

pszIDrive (PSZ)
The initial drive.

This field contains a pointer to a string that specifies the initial drive applied to the initial dialog screen. This drive is not required to be in *papszIDriveList*.

papszIDriveList (PAPSZ)
Pointer to a table of pointers to drives.

Each pointer in the table points to a null-terminated string, and each string is a valid drive or network identifier. These drives and network IDs will be sorted in ascending order in the Drive drop-down box. The end of the table is marked by a null pointer. To specify an empty table, the application sets this field to NULL, or it specifies a table containing only a null pointer.

hMod (HMODULE)
Module for custom dialog resources.

If FDS_CUSTOM is set, this is the HMODULE from which the custom file dialog template is loaded. NULLHANDLE causes the dialog resource to be pulled from the module of the current EXE.

szFullFile[CCHMAXPATH] (CHAR)
Character array.

An array of characters where CCHMAXPATH is a system-defined constant. On initialization, this field contains the initial fully-qualified path and file name. On completion, this field contains the selected fully-qualified path and file name. The simple file name can be replaced with a string filter, such as *.DAT. When the dialog is invoked, all drive and path information is stripped from the entry and moved to the corresponding fields in the dialog.

When a file name is specified, the Files list box is scrolled to the matching file name. When there is no exact match, the closest match is used.

When a string filter is specified, the dialog is initially refreshed using the results of this filter intersected with the results of *pszIType*. After the dialog is initially shown, the string filter remains in the file name field until a file is selected, or the user overtypes the value.

When a file is selected, **szFullFile** is returned to the calling application and is set to the selected fully-qualified file name.

When more than one file is selected in a multiple file selection dialog, only the topmost selected file name is returned in this field.

papszFQFilename (PAPSZ)
Pointer to a table of pointers to fully-qualified file names.

Returned to multiple file selection dialogs when the user selects one or more files from the list box. If the user types the file name in the file name entry field, the file name will be in **szFullFile** and this pointer will be NULL. When one or more selections are made, the count of items in this array will be returned in *ulFQFCount*.

This table of pointers is storage allocated by the file dialog. When the application completes opening or saving all of the files specified, the application must call WinFreeFileDlgList to free the storage allocated by the file dialog.

ulFQFCount (ULONG)
Number of file names.

Number of file names selected in the dialog. In a single file selection dialog, this value is 1. In a multiple file selection dialog, this value will be the number of files selected by the user.

usDlgID (USHORT)
Custom dialog ID.

The ID of the dialog window. When FDS_CUSTOM is set, this field contains the ID of the resource containing the custom dialog template.

x (SHORT)
X-axis dialog position.

This, along with *y* and *hwndP*, is used to position the dialog. It is updated in the structure if the user moves the dialog to a new position. If the FILEDLG structure is reused, the dialog appears in the position at which it was left each time it is invoked. The FDS_CENTER flag overrides this position and automatically centers the dialog in its parent.

y (SHORT)
Y-axis dialog position.

This, along with *x* and *hwndP*, is used to position the dialog. It is updated in the structure if the user moves the dialog to a new position. If the FILEDLG structure is reused, the dialog appears in the position at which it was left each time it is invoked. The FDS_CENTER flag overrides this position and automatically centers the dialog in its parent.

sEAType (SHORT)
Selected extended-attribute type.

Returns a selected extended-attribute type to assign to the file name returned in **szFullFile**. This field is a zero-based offset into the *papszITypeList* and is returned only when the Save As dialog is used. A –1 value is returned when the Open dialog is used.

Summary

Following are tables that describe the OS/2 functions, window messages, data structure, and minimum set of standard controls in file dialog controls:

Table 10-1. File Dialog Control Functions

Function Name	Description
WinDefFileDlgProc	The default dialog procedure for the file dialog.
WinFileDlg	Creates and displays the file dialog and returns the user's selections.
WinFreeFileDlgList	Frees the storage allocated by the file dialog when the FDS_MULTIPLESEL dialog flag is set.

Table 10-2. File Dialog Control Window Messages

Message Name	Description
FDM_ERROR	Sent before the file dialog displays a message notifying the user of an error.
FDM_FILTER	Sent before a file that meets the current filter criteria is added to the File list box.
FDM_VALIDATE	Sent when the user selects a file and presses the Enter key or clicks on the OK push button, or when the user double-clicks on a file name in the File list box.

Table 10-3. File Dialog Control Data Structure

Data Structure Name	Description
FILEDLG	File-dialog data structure.

Table 10-4 (Page 1 of 2). File Dialog Standard Controls			
Standard Control Name	**ID**	**Class/Style**	**Remarks**
DID_APPLY_PB	268	WC_BUTTON, BS_PUSHBUTTON | WS_VISIBLE	Button control. Used to apply selection for a modeless dialog.
DID_CANCEL_PB	DID_CANCEL	WC_BUTTON, BS_PUSHBUTTON | WS_VISIBLE	Button control. Used as a Cancel push button.
DID_DIRECTORY_LB	264	WC_LISTBOX, LS_HORZSCROLL | LS_OWNERDRAW | WS_TABSTOP | WS_VISIBLE	List-box control. Used to display and select the directories on the system.
DID_DIRECTORY_TXT	263	WC_STATIC, DT_LEFT | DT_TOP | SS_TEXT | WS_GROUP | WS_VISIBLE	Static-text control. Label for the Directory list box.
DID_DRIVE_CB	260	WC_COMBOBOX, CBS_DROPDOWNLIST | WS_TABSTOP | WS_VISIBLE	Combination-box control. Used to display and select drive names.
DID_DRIVE_TXT	259	WC_STATIC, DT_LEFT | DT_TOP | SS_TEXT | WS_GROUP | WS_VISIBLE	Static-text control. Label for the Drive field.
DID_FILE_DIALOG	256	DIALOG, FS_DLGBORDER | FS_NOBYTEALIGN | WS_CLIPSIBLINGS | WS_SAVEBITS, FCF_DLGBORDER | FCF_SYSMENU | FCF_TITLEBAR	Dialog control ID.
DID_FILENAME_ED	258	WC_ENTRYFIELD, ES_AUTOSCROLLBAR | ES_LEFT | ES_MARGIN | WS_TABSTOP | WS_VISIBLE	Static entry field. Fully-qualified file name entry field for parsing or selecting.

Table 10-4 (Page 2 of 2). File Dialog Standard Controls			
Standard Control Name	**ID**	**Class/Style**	**Remarks**
DID_FILENAME_TXT	257	WC_STATIC, DT_LEFT | DT_TOP | SS_TEXT | WS_GROUP | WS_VISIBLE	Static-text control. Label for the File Name field.
DID_FILES_LB	266	WC_LISTBOX, LS_HORZSCROLL | WS_TABSTOP | WS_VISIBLE	List-box control. Used to display and select the files in a directory.
DID_FILES_TXT	265	WC_STATIC, DT_LEFT | DT_TOP | SS_TEXT | WS_GROUP | WS_VISIBLE	Static-text control. Label for the Files list box.
DID_FILTER_CB	262	WC_COMBOBOX, CBS_DROPDOWNLIST | WS_TABSTOP | WS_VISIBLE	Combination-box control. Used to display and select extended-attribute type filters.
DID_FILTER_TXT	261	WC_STATIC, DT_LEFT | DT_TOP | SS_TEXT | WS_GROUP | WS_VISIBLE	Static-text control. Label for the Type field.
DID_HELP_PB	267	WC_BUTTON, BS_HELP | BS_NOPOINTERFOCUS | BS_PUSHBUTTON | WS_VISIBLE	Button control. Used to request help from the application.
DID_OK_PB	DID_OK	WC_BUTTON, BS_DEFAULT | BS_PUSHBUTTON | WS_GROUP | WS_TABSTOP | WS_VISIBLE	Button control. Used as an OK push button.

Chapter 11. Font Dialog Controls

Font dialog controls provide basic functions that give users the ability to display and select from a list of:

- Font family names installed on the system
- Available styles for each font
- Available sizes for each font
- Emphasis styles available for each font.

Users can view their selections, using a sample character string in a preview area, and interact with a modal or modeless font dialog. This chapter explains how font dialog controls can be extended to meet the requirements of PM applications.

About Font Dialog Controls

In the font dialog control, *family face* is defined as the name of the typeface. Figure 11-1 shows an example of a font dialog.

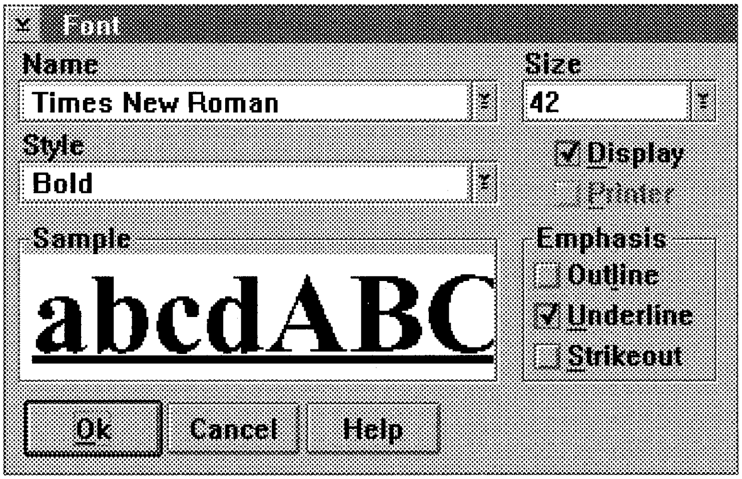

Figure 11-1. Font Dialog Example

Courier, Times New Roman**, and Helvetica** are examples of commonly used family faces. Type styles include normal, **bold**, *italic*, and ***bold italic***. *Size* is the point size, or vertical measurement, of the type. Font emphasis styles include outline, underline, and strikeout.

Customizing the Font Dialog

You can create a font dialog by customizing the font dialog control, using the standard controls and adding any controls of your own design. Specify a standard control by including a control of the same class, ID, and style as in the font dialog.

The minimum set of controls required for the font dialog are:

- DID_CANCEL_BUTTON
- DID_DISPLAY_FILTER
- DID_NAME
- DID_OK_BUTTON
- DID_OUTLINE
- DID_PRINTER_FILTER
- DID_SAMPLE
- DID_SIZE
- DID_STRIKEOUT
- DID_STYLE
- DID_UNDERSCORE

The complete set of standard controls is included in "Summary" on page 11-25.

Even if your dialog does not use all of the required controls, you must include them. You can make the unused controls invisible so that your application users are not confused.

Using Font Dialog Controls

This section describes how to create a font dialog.

Creating a Font Dialog

To present a font dialog to users, your application must do the following:

1. Allocate storage for a FONTDLG data structure and set all fields to NULL.
2. Initialize the fields in the FONTDLG data structure.

   The application must do the following:

   a. Set the *cbSize* field to the size of the data structure.

   b. Set either the *hpsScreen* or the *hpsPrinter* presentation space field, or both. You must have a valid presentation space from which to query fonts.

   c. Pass the pointer to a buffer in which to return the family name selected (*pszFamilyname*) and the size of the buffer (*usFamilyBufLen*). If the application requires a default font, pass the family name of the font in this buffer.

   The application can choose to set the following:

   a. An application-specific title. Pass the pointer to a null-terminated string in the *pszTitle* field.

   b. An application-specific preview string. Pass the pointer to a null-terminated string in the *pszPreview* field.

   c. Application-specific available font sizes for outline fonts. Pass the pointer to a null-terminated string containing point sizes, separated by spaces in the *pszPtSizeList* field.

d. A custom dialog procedure to provide application-specific function. Pass the pointer to a window procedure in the *pfnDlgProc* field.

e. Set the appropriate FNTS_* flags in the *fl* field to customize the dialog style.

f. Set the FNTF_NOVIEWPRINTERFONTS or FNTF_NOVIEWSCREENFONTS flags to customize the dialog style when working with printer fonts in the *flFlags* field. These filter flags should be initialized only when both the *hpsScreen* and the *hpsPrinter* presentation space fields are non-NULL.

g. Pass the initial position of the dialog in the *x* and *y* fields.

3. Initialize the FONTDLG data structure with any values that users should see when they invoke the dialog for the first time. For example, you can:

   a. Pass the characteristics of the default font in the *usWeight*, *usWidth*, *flType*, and *sNominalPointSize* fields.

   b. Pass any display options of the default font in the *flStyle* field.

   c. Pass the color options for displaying the font sample in the *clrFore* and *clrBack* fields.

4. Invoke the font dialog. Call WinFontDlg and pass the dialog's parent window handle, owner window handle, and a pointer to the initialized FONTDLG data structure.

5. Check the return value from WinFontDlg. If it is successful, the selected font can be used by the application. The information returned in the *fAttrs* field of the FONTDLG data structure is used.

Graphical User Interface Support for Font Dialog Controls

This section contains information about the graphical user interface support.

Name Field

The *Name* field is a drop-down list that displays a font family name. When the font dialog is invoked, the value displayed in this field is either an application-supplied family name or the default system font.

When users select a family name from the drop-down list, the *Name* field display is refreshed with the selected family name. The preview area is updated to show the sample character string in the selected family face, using the font style, size, and emphasis currently in effect.

Style Field

The *Style* field is a drop-down list that displays a font style. When the font dialog is invoked, the value displayed in this field is either an application-specified font style or the system default.

When users select a font style from the drop-down list, the *Style* field display is refreshed with the selected style name. The preview area is updated to show the sample character string in the selected font style, using the family name, size, and emphasis currently in effect.

Size Field

The *Size* field is a drop-down combination box that displays available font sizes. Users can display and select from a list of available sizes for a font, or they can type a font size directly into the entry field.

When users select a font size from the drop-down list, the *Size* field display is refreshed with the selected size. The preview area is updated to show the character string in the selected font size, using the family name, font style, and emphasis currently in effect.

The font sizes included in the drop-down list are dependent on the character definition of the font. For image or raster fonts, all available sizes are listed. For outline fonts, the default sizes are 8, 10, 12, 14, 18, and 24 points. If required, the application can specify the available sizes for outline fonts.

When users type a font size in the entry field, the preview area is updated immediately. The *Size* field will accept a fixed point number, such as 24.25, with up to four places saved after the decimal.

Emphasis Group Box

The *Emphasis group box* is a multiple-selection field that contains a list of emphasis styles (*Outline, Underline, Strikeout*) available for each font.

When users select an emphasis style, the preview area is updated immediately. The Outline selection is not available for image fonts.

Preview Area

The *Preview* area enables users to view their font family, style, size and emphasis selections as they make them. It contains a sample character string that is defined by the application. The default character string is abcdABCD. The Preview area displays font sizes as large as 48 points. As the size of the font increases, the sample displayed is clipped by the borders of the area.

Filter Check Box

The *Filter* check box enables users to limit the font family name drop-down list to select from fonts that are displayable only, printable only, or a merged list. The initial setting of the *Filter* check box is specified by the application.

Standard Push Button and Default Action

The dialog can be dismissed with either the *OK* or *Cancel* push buttons.

Subclassing the Default Font Dialog Procedure

The name of the dialog procedure is assigned to the *pfnDlgProc* field of the FONTDLG data structure.

Related Functions

This section covers the functions that are related to font dialog controls.

WinDefFontDlgProc

This function is the default dialog procedure for the font dialog.

Syntax

```
#define INCL_winstdfont

#include <os2.h>

MRESULT WinDefFontDlgProc  (HWND hwnd, ULONG msg, MPARAM mp1,
                            MPARAM mp2)
```

Parameters

hwnd (HWND) – input
Dialog-window handle.

msg (ULONG) – input
Message identity.

mp1 (MPARAM) – input
Parameter 1.

mp2 (MPARAM) – input
Parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

WinFontDlg

This dialog allows the user to select a font.

Syntax

```
#define INCL_winstdfont

#include <os2.h>

HWND WinFontDlg  (HWND hwndP, HWND hwndO, PFONTDLG pfntd)
```

Parameters

hwndP (HWND) – input
Parent-window handle.

HWND_DESKTOP	The desktop window.
Other	Specified window.

hwndO (HWND) – input
Requested owner-window handle.

pfntd (PFONTDLG) – input
Pointer to an initialized FONTDLG structure.

Returns

hwnd (HWND) – returns
Font dialog window handle.

Related Window Messages

This section covers the window messages that are related to font dialog controls.

FNTM_FACENAMECHANGED

This message notifies the subclassing application whenever the font family name is changed by the user.

Parameters

param1

pFamilyname (PSZ)
Pointer to the currently-selected face name.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

FNTM_FILTERLIST

This message is sent whenever the Font Dialog is preparing to add a font family name, font style type, or point size entry to the combination box fields that contain these parameters.

Parameters

param1

pFontname (PSZ)
Pointer to the text string that is being added to the combination box.

param2

usFieldId (USHORT)
Field identifier.

The identifier of the field to which the text string is being added. The identifier can be one of the following:

FNTI_FAMILYNAME	The text string is an addition to the family name combination box.
FNTI_STYLENAME	The text string is an addition to the style combination box.
FNTI_POINTSIZE	The text string is an addition to the size combination box.

usFontType (USHORT)
Font information.

The family name, style, or point size that is being added to the combination box. Use one of the following to identify the font information that is being added:

FNTI_BITMAPFONT	A bit-map font is being added or a point size of a bit-map font is being added.
FNTI_VECTORFONT	A vector font is being added.
FNTI_SYNTHESIZED	A synthesized font is being added. This value is valid for the style field only.
FNTI_FIXEDWIDTHFONT	A fixed width (monospace) font is being added.
FNTI_PROPORTIONALFONT	A proportionally spaced font is being added.
FNTI_DEFAULTLIST	A point size from the default list (or the application-supplied list) is being added.

Returns

rc (BOOL)
Filter indicator.

TRUE	Add the text string to the combination box.
FALSE	Do not add the text string to the combination box.

FNTM_POINTSIZECHANGED

This message notifies subclassing applications when the point size of the font is changed by the user.

Parameters

param1

pPointSize (PSZ)
Pointer to the text in the point-size entry field.

param2

fxPointSize (FIXED)
Point size.

The *fxPointSize* field in FONTDLG stated in fixed-point notation.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

FNTM_STYLECHANGED

This message notifies subclassing applications when the user changes any of the attributes in the STYLECHANGE structure.

Parameters

param1

styc (STYLECHANGE)
Style changes.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

FNTM_UPDATEPREVIEW

This message notifies subclassing applications before the preview window is updated. This occurs when the font selection is modified.

Parameters

param1

hwndPreview (HWND)
Window handle.

Window handle the preview image is drawn into. This is a static text field.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

Related Notification Message

This section covers the notification message that is related to font dialog controls.

WM_DRAWITEM (in Font Dialog)

If the FNTS_OWNERDRAWPREVIEW style is set for a font dialog, this notification message is sent to that dialog's owner whenever the preview window area (sample text) is to be drawn.

Parameters

param1

id (USHORT)
Window identifier.

The window ID of the sample area (DID_SAMPLE).

param2

pOwnerItem (POWNERITEM)
Pointer to an OWNERITEM data structure.

The following list defines the OWNERITEM data structure fields as they apply to the font dialog. See "OWNERITEM" on page 8-111 for the default field values.

hwnd **(HWND)**
Window handle of the sample area.

hps **(HPS)**
Presentation-space handle.

fsState **(ULONG)**
Reserved.

fsAttribute **(ULONG)**
Reserved.

fsStateOld **(ULONG)**
Reserved.

fsAttributeOld **(ULONG)**
Reserved.

rclItem **(RECTL)**
Item rectangle to be drawn in window coordinates.

idItem **(LONG)**
Reserved.

hItem **(CNRDRAWITEMINFO)**
Reserved.

Returns

rc (BOOL)

Item-drawn indicator.

TRUE	The owner draws the item.
FALSE	If the owner does not draw the item, the owner returns this value and the font dialog draws the item.

Related Data Structures

This section covers the data structures that are related to font dialog controls.

FACENAMEDESC

Face-name description structure. See GpiQueryFaceString.

Syntax

```
typedef struct _FACENAMEDESC {
USHORT       usSize;
USHORT       usWeightClass;
USHORT       usWidthClass;
USHORT       usReserved;
ULONG        flOptions;
 } FACENAMEDESC;

typedef FACENAMEDESC *PFACENAMEDESC;
```

Fields

usSize (USHORT)
Length of structure.

usWeightClass (USHORT)
Weight class.

Indicates the visual weight (thickness of strokes) of the characters in the font:

FWEIGHT_DONT_CARE	Any font weight satisfies the request.
FWEIGHT_ULTRA_LIGHT	Ultra-light.
FWEIGHT_EXTRA_LIGHT	Extra-light.
FWEIGHT_LIGHT	Light.
FWEIGHT_SEMI_LIGHT	Semi-light.
FWEIGHT_NORMAL	Medium (normal) weight.
FWEIGHT_SEMI_BOLD	Semi-bold.
FWEIGHT_BOLD	Bold.
FWEIGHT_EXTRA_BOLD	Extra-bold.
FWEIGHT_ULTRA_BOLD	Ultra-bold.

usWidthClass (USHORT)
Width class.

Indicates the relative aspect ratio of the characters of the font in relation to the normal aspect ratio for this type of font:

FWIDTH_DONT_CARE	Any font width satisfies the request.
FWIDTH_ULTRA_CONDENSED	Ultra-condensed (50% of normal).
FWIDTH_EXTRA_CONDENSED	Extra-condensed (62.5% of normal).
FWIDTH_CONDENSED	Condensed (75% of normal).

FWIDTH_SEMI_CONDENSED	Semi-condensed (87.5% of normal).
FWIDTH_NORMAL	Medium (normal).
FWIDTH_SEMI_EXPANDED	Semi-expanded (112.5% of normal).
FWIDTH_EXPANDED	Expanded (125% of normal).
FWIDTH_EXTRA_EXPANDED	Extra-expanded (150% of normal).
FWIDTH_ULTRA_EXPANDED	Ultra-expanded (200% of normal).

usReserved (USHORT)
Reserved.

flOptions (ULONG)
Other characteristics of the font.

FTYPE_ITALIC	Italic font required. If not specified, non-italic font required.
FTYPE_ITALIC_DONT_CARE	Italic and non-italic fonts can satisfy the request. If this option is specified, FTYPE_ITALIC is ignored.
FTYPE_OBLIQUE	Oblique font required. If not specified, non-oblique font required.
FTYPE_OBLIQUE_DONT_CARE	Oblique and non-oblique fonts can satisfy the request. If this option is specified, FTYPE_OBLIQUE is ignored.
FTYPE_ROUNDED	Rounded font required. If not specified, non-rounded font required.
FTYPE_ROUNDED_DONT_CARE	Rounded and non-rounded fonts can satisfy the request. If this option is specified, FTYPE_ROUNDED is ignored.

FATTRS

Font-attributes structure.

Syntax

```
typedef struct _FATTRS {
USHORT      usRecordLength;
USHORT      fsSelection;
LONG        lMatch;
CHAR        szFacename[FACESIZE];
USHORT      idRegistry;
USHORT      usCodePage;
LONG        lMaxBaselineExt;
LONG        lAveCharWidth;
USHORT      fsType;
USHORT      fsFontUse;
 } FATTRS;

typedef FATTRS *PFATTRS;
```

Fields

usRecordLength (USHORT)
Length of record.

fsSelection (USHORT)
Selection indicators.

Flags causing the following features to be simulated by the system.

Note: If an italic flag is applied to a font that is itself defined as italic, the font is slanted further by italic simulation.

Underscore or strikeout lines are drawn using the appropriate attributes (for example, color) from the character bundle (see the CHARBUNDLE datatype), not the line bundle (see LINEBUNDLE). The width of the line, and the vertical position of the line in font space, are determined by the font. Horizontally, the line starts from a point in font space directly above or below the start point of each character, and extends to a point directly above or below the escapement point for that character.

For this purpose, the start and escapement points are those applicable to left-to-right or right-to-left character directions (see GpiSetCharDirection in GPI), even if the string is currently being drawn in a top-to-bottom or bottom-to-top direction.

For left-to-right or right-to-left directions, any white space generated by the character extra and character break extra attributes (see GpiSetCharExtra and GpiSetCharBreakExtra in GPI), as well as increments provided by the vector of increments on GpiCharStringPos and GpiCharStringPosAt, are also underlined/overstruck, so that in these cases the line is continuous for the string.

FATTR_SEL_ITALIC	Generate *italic* font.
FATTR_SEL_UNDERSCORE	Generate underscored font.
FATTR_SEL_BOLD	Generate **bold** font. (Note that the resulting characters are wider than those in the original font.)
FATTR_SEL_STRIKEOUT	Generate font with ~~overstruck~~ characters.
FATTR_SEL_OUTLINE	Use an outline font with hollow characters. If this flag is not set, outline font characters are filled. Setting this flag normally gives better performance, and for sufficiently small characters (depending on device resolution) there may be little visual difference.

lMatch (LONG)
Matched-font identity.

szFacename[FACESIZE] (CHAR)
Typeface name.

The typeface name of the font, for example, Tms Rmn.

idRegistry (USHORT)
Registry identifier.

Font registry identifier (zero if unknown).

usCodePage (USHORT)
Code page.

If zero, the current Gpi code page (see GpiSetCp in GPI) is used. A subsequent GpiSetCp function changes the code page used for this logical font.

lMaxBaselineExt (LONG)
Maximum baseline extension.

For raster fonts, this should be the height of the required font, in world coordinates.

For outline fonts, this should be zero.

lAveCharWidth (LONG)
Average character width.

For raster fonts, this should be the width of the required font, in world coordinates.

For outline fonts, this should be zero.

fsType (USHORT)
Type indicators.

FATTR_TYPE_KERNING	Enable kerning (PostScript** only).
FATTR_TYPE_MBCS	Font for mixed single- and double-byte code pages.
FATTR_TYPE_DBCS	Font for double-byte code pages.
FATTR_TYPE_ANTIALIASED	Antialiased font required. Only valid if supported by the device driver.

fsFontUse (USHORT)
Font-use indicators.

These flags indicate how the font is to be used. They affect presentation speed and font quality.

FATTR_FONTUSE_NOMIX	Text is not mixed with graphics and can be written without regard to any interaction with graphics objects.
FATTR_FONTUSE_OUTLINE	Select an outline (vector) font. The font characters can be used as part of a path definition. If this flag is not set, an outline font might or might not be selected. If an outline font is selected, however, character widths are rounded to an integral number of &pel.s.
FATTR_FONTUSE_TRANSFORMABLE	Characters can be transformed (for example, scaled, rotated, or sheared).

FONTDLG

Font-dialog structure.

Syntax

```
typedef struct _FONTDLG {
ULONG          cbSize;
HPS            hpsScreen;
HPS            hpsPrinter;
PSZ            pszTitle;
PSZ            pszPreview;
PSZ            pszPtSizeList;
PFNWP          pfnDlgProc;
PSZ            pszFamilyname;
FIXED          fxPointSize;
ULONG          fl;
ULONG          flFlags;
ULONG          flType;
ULONG          flTypeMask;
ULONG          flStyle;
ULONG          flStyleMask;
LONG           clrFore;
LONG           clrBack;
ULONG          ulUser;
LONG           lReturn;
LONG           lSRC;
LONG           lEmHeight;
LONG           lXHeight;
LONG           lExternalLeading;
HMODULE        hMod;
FATTRS         fAttrs;
SHORT          sNominalPointSize;
USHORT         usWeight;
USHORT         usWidth;
SHORT          x;
SHORT          y;
USHORT         usDlgId;
USHORT         usFamilyBufLen;
USHORT         usReserved;
 } FONTDLG;

typedef FONTDLG *PFONTDLG;
```

Fields

cbSize (ULONG)

Structure size.

This field allows for future expansion of the structure, and must be initialized with the size of the FONTDLG structure.

hpsScreen (HPS)
Screen presentation space.

If not NULLHANDLE, the screen presentation space from which screen fonts are queried.

hpsPrinter (HPS)
Printer presentation space.

If not NULLHANDLE, the printer presentation space from which printer font are queried.

pszTitle (PSZ)
Dialog title string.

Application-provided dialog title. If NULL, it defaults to "Font."

pszPreview (PSZ)
Font-preview window string.

String to show in font-preview window. If NULL, it defaults to "abcdABCD."

Note: Care is necessary when choosing the string to put in this field. Using many different characters causes excess memory to be used by the font cache.

pszPtSizeList (PSZ)
Application-provided point size list.

String which contains a list of point sizes to be used as the default list for outline fonts in the point-size drop-down area. Point sizes are separated by spaces. If NULL, the point size drop down defaults to 8, 10, 12, 14, 18, and 24.

pfnDlgProc (PFNWP)
Custom dialog procedure.

NULL unless the caller is subclassing the font dialog. When non-NULL, it points to the dialog procedure of the application.

pszFamilyname (PSZ)
Family name buffer.

Buffer provided by the application for passing the family name of the font. The font family name used by the application to select a font. When the first character in this string is NULL, no family name was initially selected, and the dialog defaults to the system font.

A buffer must be passed to the font dialog to allow the dialog to return the selected font family name. The size of this buffer is placed in the *usFamilyBufLen* field.

fxPointSize (FIXED)
Point size of the font.

If FNTS_OWNERDRAWPREVIEW is set, 0 means the user wants to leave the font size unchanged and the application must update the preview area.

fl (ULONG)
FNTS_* flags.

FNTS_APPLYBUTTON	An Apply push button is added to the dialog. This is useful in a modeless dialog.
FNTS_BITMAPONLY	The dialog presents bit-map fonts only. An application that changes fonts by using the presentation parameters (PP_* values) could use this flag.
FNTS_CENTER	The dialog is positioned in the center of its parent window, overriding any specified x,y position.
FNTS_CUSTOM	A custom dialog template is used to create the dialog. The *hMod* and *usDlgId* fields must be initialized.
FNTS_FIXEDWIDTHONLY	The dialog presents fixed-width (monospace) fonts only.
FNTS_HELPBUTTON	A Help push button of style (BS_HELP|BS_NOPOINTERFOCUS) with an ID of DID_HELP_BUTTON is added to the dialog. If the push button is pressed, a WM_HELP message is sent to the *hwndO* parameter of the WinFontDlg function call.
FNTS_INITFROMFATTRS	The dialog initializes itself from the font attribute structure (FATTRS) that is passed.
FNTS_MODELESS	The dialog is modeless; WinFontDlg returns immediately after creating the dialog window and returns the window handle to the application. The application should treat the dialog as if it were created with WinLoadDlg. As in the modal (default) dialog case, the return value is found in the *lReturn* field of the FONTDLG structure passed to WinFontDlg.
FNTS_NOSYNTHESIZEDFONTS	The dialog does not synthesize any fonts.
FNTS_OWNERDRAWPREVIEW	This flag makes the check boxes in the font dialog three-state check boxes, enabling the user to leave certain style attributes unchanged. Additionally, a WM_DRAWITEM message will be sent to the owner, providing the owner an opportunity to draw the preview window itself.
FNTS_PROPORTIONALONLY	The dialog presents proportionally spaced fonts only.
FNTS_RESETBUTTON	A Reset push button is added to the dialog. When this push button is pressed, the values for the dialog are restored to their initial values.
FNTS_VECTORONLY	The dialog presents vector fonts only.

flFlags (ULONG)
FNTF_* flags.

FNTF_NOVIEWPRINTERFONTS	This flag is initialized only when both *hpsScreen* and *hpsPrinter* are not NULLHANDLE. On input, this parameter determines whether the printer fonts are to be included in the font list box. The user controls this with a check box.
FNTF_NOVIEWSCREENFONTS	This flag is initialized only when both *hpsScreen* and *hpsPrinter* are not NULLHANDLE. On input, this parameter determines whether the screen fonts should be included in the font list box. The user controls this with a check box.
FNTF_PRINTERFONTSELECTED	This determines if a printer-specific font is selected by the user. The application should make an approximation of this printer font when outputting to the screen. This is an output-only flag and is ignored on input.
FNTF_SCREENFONTSELECTED	This determines if a screen-specific font is selected by the user. The application should make an approximation of this screen font when outputting to the screen. This is an output-only flag and is ignored on input.

flType (ULONG)
The selected type bits.

These flags specify what additional attributes the user specified for the font. This field is used as the *flOptions* field in the FACENAMEDESC structure for GpiQueryFaceString.

flTypeMask (ULONG)
Mask of type bits to use.

This field is used only if FNTS_OWNERDRAWPREVIEW is specified. It tells which flags of the *flTypeMask* field the user wants to change, and is relevant only if the text for which the font is selected has different faces and styles.

flStyle (ULONG)
Selected style bits.

Flags for any additional selections the user specified for the font. This field is used as the *fsSelection* field in the FATTRS structure passed to GpiCreateLogFont.

flStyleMask (ULONG)
Mask of style bits to use.

This field is used only if FNTS_OWNERDRAWPREVIEW is specified. It tells which flags of the *flStyle* field the user wants to change and is relevant only if the text for which the font is selected has different faces and styles.

clrFore (LONG)
Font foreground color.

Foreground color of the font. This color is a value used for the color mode that *hpsScreen* is in. If FNTS_OWNERDRAWPREVIEW is specified, this value can be CLR_NOINDEX, leaving the foreground color "as is."

clrBack (LONG)
Font background color.

Background color of the font. This color is a value used for the color mode that *hpsScreen* is in. If FNTS_OWNERDRAWPREVIEW is specified, this value can be CLR_NOINDEX leaving the background color "as is."

ulUser (ULONG)
Application-defined.

A ULONG that an application uses to store its state information when it is subclassing the font dialog.

lReturn (LONG)
Return value.

Return value from WinFontDlg. This value is the ID of the push button pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplied additional push buttons in its template.

lSRC (LONG)
System return code.

This field contains an FNTS_ERR return code. When a dialog fails, this field is used to tell the application the reason for the failure.

lEmHeight (LONG)
Em height.

The Em height of the current font. This is the same as in the FONTMETRICS structure. It is an output-only parameter and its value has no effect on the behavior of the font dialog, but is updated when the user dismisses the dialog.

lXHeight (LONG)
X height.

The x height of the current font. This is the same as in the FONTMETRICS structure. It is an output-only parameter and its value has no effect on the behavior of the font dialog, but is updated when the user dismisses the dialog.

lExternalLeading (LONG)
External leading.

The external leading of the font. This is the same as in the FONTMETRICS structure. It is an output-only parameter and its value has no effect on the behavior of the font dialog, but is updated when the user dismisses the dialog.

hMod (HMODULE)
Module for custom dialog resources.

If FNTS_CUSTOM is set, this is the HMODULE from which the custom font dialog template is loaded. NULLHANDLE causes the dialog resource to be pulled from the module of the current EXE.

fAttrs (FATTRS)
Font-attribute structure.

Font-attribute structure of selected font. The FATTRS for the selected font. This is output-only for all fields except *usCodePage*, which is input/output, and the initial code page value passed is used for font selection. The value returned is the one for the matching font.

sNominalPointSize (SHORT)
Font point size.

The nominal point size of the font. This is the same as in the FONTMETRICS structure. It is an output-only parameter and its value has no effect on the behavior of the font dialog, but is updated when the user dismisses the dialog.

usWeight (USHORT)
Font weight.

The weight of the font. This is the weight-class/boldness the user selects for the font. This field is used as the *usWeightClass* field in the FACENAMEDESC structure for GpiQueryFaceString. When FNTS_OWNERDRAWPREVIEW is set, 0 causes the application to leave the font weight "as is" and the application must update the preview area.

usWidth (USHORT)
Font width.

The width of the font. This is the width-class the user selects for the font. This field is used as the *usWidthClass* field in the FACENAMEDESC structure for GpiQueryFaceString. When FNTS_OWNERDRAWPREVIEW is set, 0 causes the application to leave the font width "as is" and the application must update the preview area.

x (SHORT)
The x-axis dialog position.

This, along with *y* and *hwndP*, is used to position the dialog. It is updated in the structure if the user moves the dialog to a new position. This way, the dialog appears in the position at which it was left each time it is invoked. The FNTS_CENTER flag overrides this position and automatically centers the dialog in its parent.

y (SHORT)
The y-axis dialog position.

This, along with *x* and *hwndP*, is used to position the dialog. It is updated in the structure if the user moves the dialog to a new position. This way, the dialog appears in the position at which it was left each time it is invoked. The FNTS_CENTER flag overrides this position and automatically centers the dialog in its parent.

usDlgId (USHORT)
Dialog ID.

This sets the ID of the dialog window. If FNTS_CUSTOM is set, this is the ID of the resource that contains the custom dialog template.

usFamilyBufLen (USHORT)
Buffersize.

Size of the buffer passed in the *pszFamilyname* resource that contains the custom dialog template.

usReserved (USHORT)
Reserved.

This is a reserved field.

STYLECHANGE

Style-change structure. This structure is returned by the FNTM_STYLECHANGED message.

All “old” fields describe the style attributes before the user made a change. The other, or “new”, parameters describe the style that will be in effect after this is passed to WinDefFontDlgProc. When the “old” and “new” values are the same, the user made no change.

For further details of the parameters, see FONTDLG.

Syntax

```
typedef struct _STYLECHANGE {
USHORT       usWeight;
USHORT       usWeightOld;
USHORT       usWidth;
USHORT       usWidthOld;
ULONG        flType;
ULONG        flTypeOld;
ULONG        flTypeMask;
ULONG        flTypeMaskOld;
ULONG        flStyle;
ULONG        flStyleOld;
ULONG        flStyleMask;
ULONG        flStyleMaskOld;
 } STYLECHANGE;

typedef STYLECHANGE *PSTYLECHANGE;
```

Fields

usWeight (USHORT)
New weight of font.

usWeightOld (USHORT)
Old weight of font.

usWidth (USHORT)
New width of font.

usWidthOld (USHORT)
Old width of font.

flType (ULONG)
New type of font.

flTypeOld (ULONG)
Old type of font.

flTypeMask (ULONG)
New type mask.

flTypeMaskOld (ULONG)
Old type mask.

flStyle (ULONG)
New selected style bits.

flStyleOld (ULONG)
Old selected style bits.

flStyleMask (ULONG)
New mask of style bits to use.

flStyleMaskOld (ULONG)
Old mask of style bits to use.

Summary

Following are tables that describe the OS/2 functions, window messages, notification message, data structures, and standard controls in font dialog controls:

Table 11-1. Font Dialog Control Functions

Function Name	Description
WinDefFontDlgProc	The default dialog procedure for the font dialog.
WinFontDlg	Allows the user to select a font.

Table 11-2. Font Dialog Control Window Messages

Message Name	Description
FNTM_FACENAMECHANGED	Notifies the subclassing application whenever the font family name is changed by the user.
FNTM_FILTERLIST	Sent whenever the font dialog is preparing to add a font family name, font style type, or point size entry to the combination-box fields that contain these parameters.
FNTM_POINTSIZECHANGED	Notifies subclassing applications when the point size of the font is changed by the user.
FNTM_STYLECHANGED	Notifies subclassing applications when the user changes any of the attributes in the STYLECHANGE data structure.
FNTM_UPDATEPREVIEW	Notifies subclassing applications before the preview window is updated.

Table 11-3. Font Dialog Control Notification Message

Message Name	Description
WM_DRAWITEM	Sent to the owner of the font dialog control each time an item is to be drawn.

Table 11-4. Font Dialog Control Data Structures

Data Structure Name	Description
FACENAMEDESC	Face-name description data structure.
FATTRS	Font-attributes data structure.
FONTDLG	Font-dialog data structure.
STYLECHANGE	Style-change data structure returned by the FNTM_STYLECHANGED message.

Table 11-5 (Page 1 of 3). Font Dialog Standard Controls			
Standard Control Name	**ID**	**Class/Style**	**Remarks**
DID_APPLY_BUTTON	311	WC_BUTTON, BS_PUSHBUTTON | WS_VISIBLE	Button control *provided by the application*. Used as an Apply push button in modeless applications.
DID_CANCEL_BUTTON	DID_CANCEL	WC_BUTTON, BS_PUSHBUTTON | WS_VISIBLE	Button control. Used as a Cancel push button.
DID_DISPLAY_FILTER	303	WC_BUTTON, BS_AUTOCHECKBOX | WS_GROUP | WS_TABSTOP | WS_VISIBLE	Button control. Used to filter the Font Name field.
DID_EMPHASIS_GROUPBOX	317	WC_STATIC, SS_GROUPBOX | WS_GROUP | WS_VISIBLE	Group box around the emphasis check boxes.
DID_FONT_DIALOG	300	DIALOG, FCF_SYSMENU | FCF_TITLEBAR | FS_BORDER | FS_DLGBORDER | FS_NOBYTEALIGN | WS_CLIPSIBLINGS | WS_SAVEBITS	Dialog control ID.
DID_HELP_BUTTON	310	WC_BUTTON, BS_HELP | BS_NOPOINTERFOCUS | BS_PUSHBUTTON | WS_VISIBLE	Button control. Used to request help from the application.
DID_NAME	301	WC_COMBOBOX, CBS_DROPDOWNLIST | WS_TABSTOP | WS_VISIBLE	Combination-box control. Used to display and select font family names.
DID_NAME_PREFIX	313	WC_STATIC, DT_LEFT | DT_TOP | SS_TEXT | WS_GROUP | WS_VISIBLE	Static-text control. Label for the font Family Name field.
DID_OK_BUTTON	DID_OK	WC_BUTTON, BS_DEFAULT | BS_PUSHBUTTON | WS_GROUP | WS_TABSTOP | WS_VISIBLE	Button control. Used as an OK push button.

Table 11-5 (Page 2 of 3). Font Dialog Standard Controls			
Standard Control Name	**ID**	**Class/Style**	**Remarks**
DID_OUTLINE	307	WC_BUTTON, BS_AUTOCHECKBOX | WS_TABSTOP | WS_VISIBLE	Check-box control. Used to select the outline emphasis of the selected font.
DID_PRINTER_FILTER	304	WC_BUTTON, BS_AUTOCHECKBOX | WS_TABSTOP | WS_VISIBLE	Button control. Used to filter the Font Name field.
DID_RESET_BUTTON	312	WC_BUTTON, BS_PUSHBUTTON | WS_VISIBLE	Button control *provided by the application*. Used as a Reset push button.
DID_SAMPLE	306	WC_STATIC, DT_CENTER | DT_VCENTER | SS_TEXT | WS_GROUP | WS_VISIBLE	Static-text control. Used to display the preview string in the selected font.
DID_SAMPLE_GROUPBOX	316	WC_STATIC, SS_GROUPBOX | WS_GROUP | WS_VISIBLE	Group box around a sample field.
DID_SIZE	305	WC_COMBOBOX CBS_DROPDOWN | WS_TABSTOP | WS_VISIBLE	Combination-box control. Used to display, select, and enter the type size of the selected font.
DID_SIZE_PREFIX	315	WC_STATIC, DT_LEFT | DT_TOP | SS_TEXT | WS_GROUP | WS_VISIBLE	Static-text control. Label for the font Type Size field.
DID_STRIKEOUT	309	WC_BUTTON, BS_AUTOCHECKBOX | WS_VISIBLE	Check-box control. Used to select strikeout emphasis of the selected font.
DID_STYLE	302	WC_COMBOBOX, CBS_DROPDOWNLIST | WS_TABSTOP | WS_VISIBLE	Combination-box control. Used to display and select font style names.

Table 11-5 (Page 3 of 3). Font Dialog Standard Controls			
Standard Control Name	**ID**	**Class/Style**	**Remarks**
DID_STYLE_PREFIX	314	WC_STATIC, DT_LEFT | DT_TOP | SS_TEXT | WS_GROUP | WS_VISIBLE	Static-text control. Label for the font Style Name field.
DID_UNDERSCORE	308	WC_BUTTON, BS_AUTOCHECKBOX | WS_VISIBLE	Check box control. Used to select the underscore emphasis of the selected font.

Chapter 12. Direct Manipulation

Direct manipulation is the act of moving graphical representations such as OS/2 icons around the screen using a pointing device, such as a mouse. This chapter explains how to use direct manipulation in PM applications.

About Direct Manipulation

The direct manipulation protocol enables the user to select an object in a window, drag it to another location, and drop it on another object or in another window. *Dragging* is the act of moving an object as though it were attached to the pointer; it is performed by pressing and holding the drag button and moving the pointer. *Dropping* is the act of fixing the position of the dragged object by releasing the drag button on the pointer. This causes interaction (data exchange) between the window from which the selected object is dragged and the window containing the object on which the selected object is dropped. Figure 12-1 shows an example of an object being dragged to a printer.

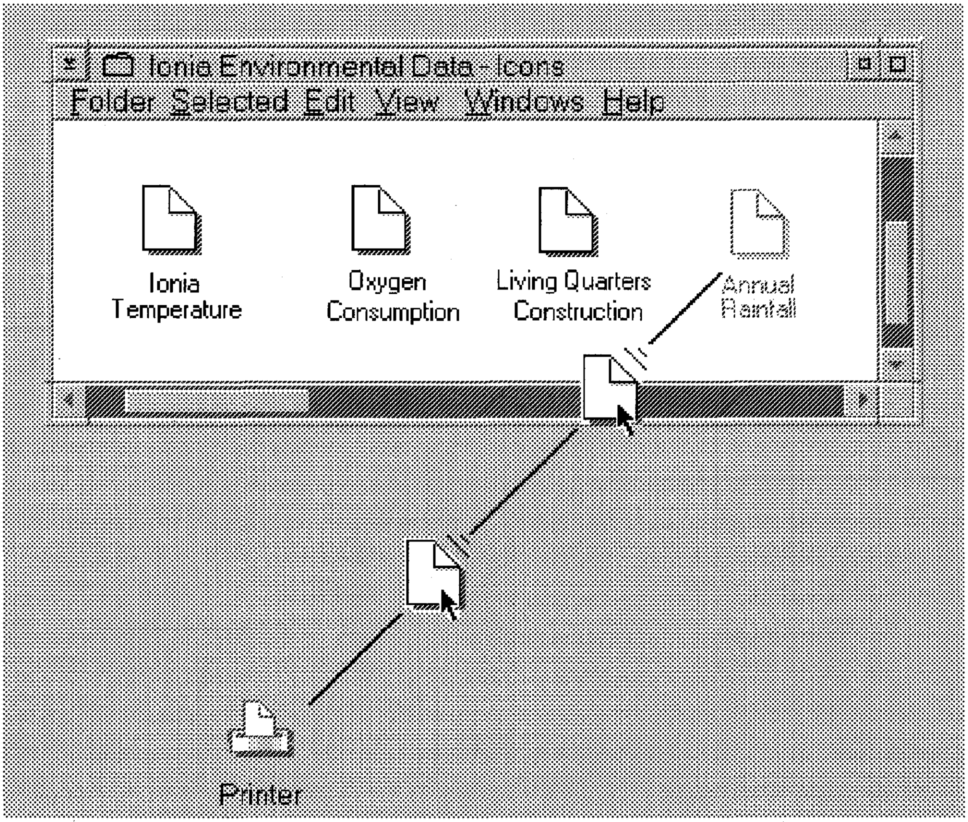

Figure 12-1. Dragging Data to a Printer

The window containing the dragged object is the *source*. The window containing the object that was dropped on is the *target*. The source and target can be the same window, different windows within the same application, or windows belonging to different applications. The dragged object can be either the only visible object in the source window or one of many objects. The target object can be either the only visible object in the target window or one of many objects. A source or target window that contains multiple objects is a *container window*.

The data exchange that occurs between the source and target after a direct manipulation operation enables applications that support the protocol to integrate easily, while providing a simple user interface.

Application-Defined Drag Operations

At times it may be useful for an application to define its own drag operation to facilitate functions between two windows in the same application or between closely related applications. For example, an application implementing a keyboard remapping function may want to provide a method of redefining keys with direct manipulation. This application could define an operation whereby dragging one key to another exchanges the definitions of the two keys. The protocol provides the extendability to enable this kind of function.

Rendering Mechanism and Format

The *rendering mechanism* represents the way in which you want to exchange the data, for example, dynamic data exchange (DDE). The *rendering format* identifies the actual type or true type of the data, for example, text. To exchange data, both the source and target must know how to communicate with each other through the rendering mechanism and understand the particular format of the data.

The *native rendering mechanism* and format of the object is the mechanism that most naturally conveys the data, either where it is now, or where it can be put most easily. The format conveys all information about the data. For example, a spreadsheet cell has a location in a row and column of a spreadsheet. Rendering the spreadsheet cell in a simple text format would cause this information to be lost, so a more appropriate format should be chosen for its native rendering format.

A source application may be able to exchange data with a target through several mechanisms, such as:

- Dynamic Data Exchange (DDE)
- OS/2 File
- Print.

Additionally, the source application might be able to *render* the data in various formats, that is, into various types. For example, a spreadsheet application could render its contents in a spreadsheet format or into a simple text format. The ability of the source application to render the data in some format might, itself, depend on the exchange mechanism used. The rendering mechanisms and formats that a source application can support, for each object dropped, are provided to the target through the *hstrRMF* field in the DRAGITEM data structure.

The target application may also be able to exchange data with the source through several different combinations of mechanism and format. The target is responsible for obtaining the data from the source in the format that they both support and that provides the highest level of information about the data.

While making this determination, the target must consider the exchange capabilities offered by the mechanism. For example, an OS/2 File exchange mechanism can provide only a snapshot of the data at the time the direct manipulation operation occurred. An exchange using DDE, on the other hand, offers the target an opportunity to remain informed about changes to the data.

Non-Standard Rendering Mechanisms

Some standard rendering mechanisms are already defined, but the system lets the set of rendering mechanisms be expanded, allowing for:

- Additional standard rendering mechanisms to be defined in the future
- Application definition of private or nonstandard rendering mechanisms.

An application can elect to support some, all, or none of the standard rendering mechanisms defined by the system. Applications that do not support any of the standard rendering mechanisms are not precluded from using direct manipulation. However, support of the standard rendering mechanisms and formats increases the chances of a successful data transfer between applications.

An application that supports a particular rendering mechanism, whether or not it is a rendering mechanism defined by the system, must follow a specific set of guidelines defined by that rendering mechanism, including conversation-initiation procedures and naming conventions.

Responsibilities of a Source Application

The source is responsible for starting a direct manipulation operation. Startup can be accomplished only with a pointing device, such as a mouse. The operation starts when the application detects that a drag button has been pressed and the pointing device has moved. Dragging continues until terminated, which is usually when the button is released.

Although the direct manipulation protocol lets the application use any button for dragging, it is recommended that the system-defined drag button be used for direct manipulation operations.

The source has the following responsibilities in preparing for the actual drag of the objects across the screen:

- Allocate and initialize the DRAGINFO data structure that conveys the necessary information about each object to the target.
- Initialize a set of DRAGIMAGE data structures that describe the image to be displayed during the drag operation.

- Make the following information known to the system:
  - The type of each object being directly manipulated
  - The rendering mechanism and format for each object
  - The suggested name of the object at the target
  - The name of the container or folder containing the source object
  - The name of the object at the source
  - The *true* type of each object being directly manipulated
  - The native rendering mechanism and format for each object.

Responsibilites of a Target Application

The target in a direct manipulation operation is responsible for determining whether a particular set of objects can be dropped on it, and for providing the user with visible cues regarding the operation. A target is informed of the operation through messages sent to it as the pointer, provided by the source, is dragged across the screen.

When a set of objects is dropped on the target, the target is responsible for establishing the appropriate conversations with the source to accomplish the data transfer. The type of conversation for each object is based on the rendering mechanism and format of the object being dropped.

The target application is responsible for:

- Determining if data can be exchanged between source and target by verifying that both applications share knowledge of at least one rendering mechanism and format
- Providing visible feedback, or target emphasis, on whether a drop is allowed
- Defining the default state of a direct manipulation operation
- Initiating conversations with the source for data transfer.

Messages Sent to a Target Application

Table 12-1 describes the messages that are sent to each window whose boundaries are crossed as the user drags the object around the screen.

Table 12-1 (Page 1 of 2). Messages Sent to a Target Application

Message Name	Description
DM_DRAGOVER	Sent to the window under the pointer as the pointer is dragged across it. A single DM_DRAGOVER message is sent each time the pointer moves and each time a key is pressed or released, and it contains a pointer to the DRAGINFO data structure. The target can access this data structure with DrgAccessDraginfo.
DM_DRAGLEAVE	Sent whenever the DM_DRAGOVER message has been sent to a window, and the pointer is moved outside the bounds of that window. If the target or an object in the window had been emphasized as a target, it should be de-emphasized.

Table 12-1 (Page 2 of 2). Messages Sent to a Target Application

Message Name	Description
DM_DROP	Sent to the target to provide it with the information necessary to establish a conversation for data exchange with the source. The target should immediately remove any target emphasis. The data transfers must not be done before responding to the DM_DROP message.
DM_DROPHELP	Posted to a target to indicate that the user requested help for the drag operation while over that target.

Response to Messages Sent to a Target Application

Table 12-2 shows the four possible responses available to the target when it receives a DM_DRAGOVER message. The target sends these values to the window handle specified in the DRAGINFO data structure.

Table 12-2. Target Responses to DM_DRAGOVER

Message Name	Description
DOR_DROP	Sent if the objects being dragged are acceptable. A drop does not occur unless DOR_DROP is returned.
DOR_NODROP	Sent if the objects being dragged are acceptable and the target supports the current operation, but the objects cannot be dropped on the current location in the target window. For example, a list box might return DOR_NODROP if it contains objects that can be dropped on, but the pointer is over an object that cannot be dropped on. If the target response is DOR_NODROP, the DM_DRAGOVER message continues to be sent to the target when: • The pointer is moved • A keyboard key is pressed • The pointer is moved out of and back into the window.
DOR_NODROPOP	Sent if the objects being dragged are acceptable, but the target does not support the current operation. This response implies that the drop may be valid if the drag operation changes. For example, copying a file to a shredder would not be valid, but moving a file to a shredder would be. Once the target has sent DOR_NODROPOP, no further DM_DRAGOVER messages is sent to the target until: • A keyboard key is pressed • The pointer is moved out of and back into the window.
DOR_NEVERDROP	Sent when the objects being dragged are not acceptable, and the target will never accept them. Once the target has sent DOR_NEVERDROP, no further DM_DRAGOVER messages are sent to that target until the pointer is moved out of and back into the target window.

If a reply other than DOR_DROP is received from a target, the augmentation emphasis is automatically changed to indicate that no drop is allowed. This gives the user a visible cue that a drop cannot occur. The emphasis is reverted to *drop allowed* when a DOR_DROP reply is received from some target.

Two-Object Drag Operation

Figure 12-2 represents the sequence of functions and message flows for a typical direct manipulation operation. The flow shows a two-object drag from App1 to App3, dragging over App2.

The direct manipulation operation is started by the source window procedure after the user selects the objects to be manipulated and the source receives a WM_BEGINDRAG message.

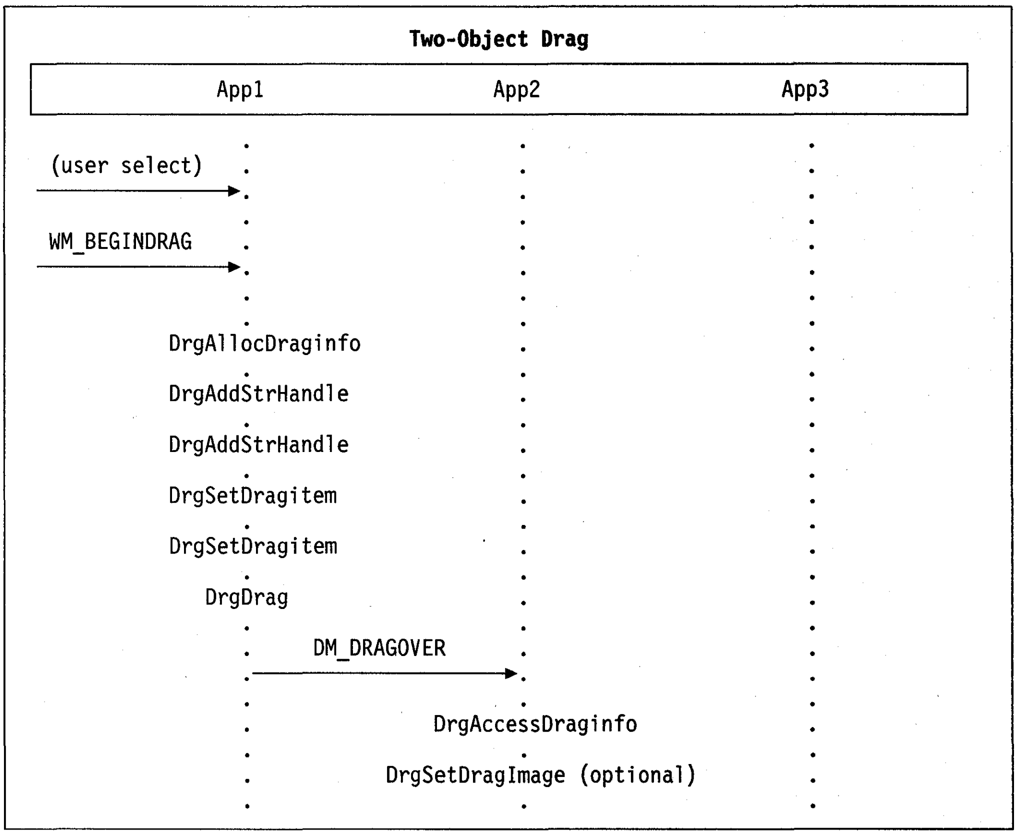

Figure 12-2 (Part 1 of 3). Diagram Showing Sequence of Function and Message Flows

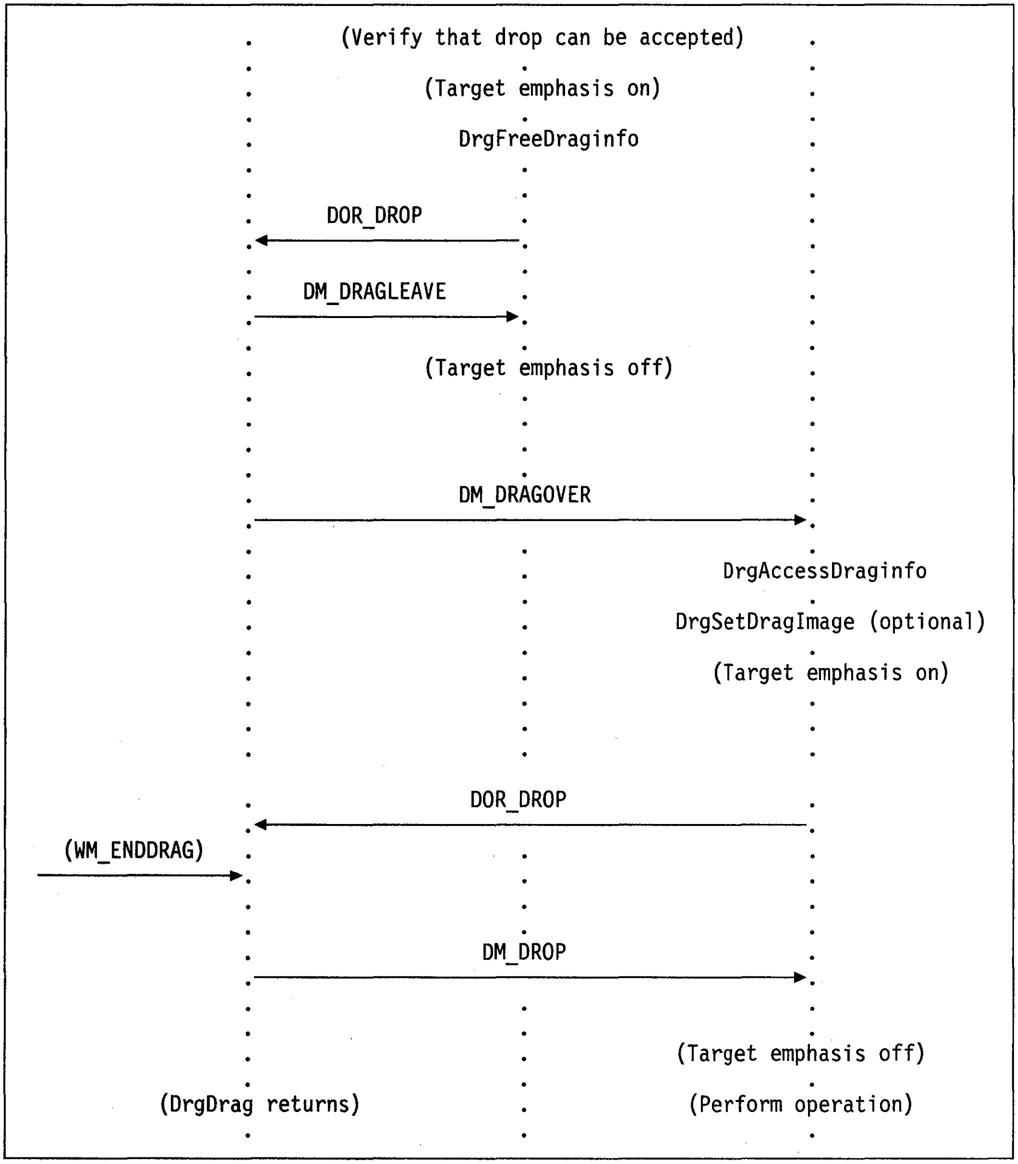

Figure 12-2 (Part 2 of 3). Diagram Showing Sequence of Function and Message Flows

```
        DrgFreeDraginfo          .        DrgDeleteDraginfoStrHandles
               .                 .                    .
               .                 .             DrgFreeDraginfo
               .                 .                    .
               .                 .                    .
```

Figure 12-2 (Part 3 of 3). Diagram Showing Sequence of Function and Message Flows

Conversation after the Drop

Figure 12-3 represents the sequence of message flows for a typical direct manipulation data-transfer operation. The flow describes a single-object move from source to target. The user dropped on white space in the target container.

For this example, assume that the rendering mechanism selected is DRM_OS2FILE and that the source does not initially provide the target with the source item's file name. Also assume that the source and target items exist on different drives.

```
              Source                              Target

                .                                   .
                .                                   .
                .                         DrgAllocDragtransfer
                .                                   .
                .                          DrgSendTransferMsg
                .                                   .
                .            DM_RENDER              .
                .<----------------------------------.
                .                                   .
Verify the rendering mechanism and format           .
                .                                   .
         DrgAddStrHandle                            .
         (hstrSourceName)                           .
                .                                   .
                .          DM_RENDER(reply)         .
                .---------------------------------->.
                .                                   .
```

Figure 12-3 (Part 1 of 2). Diagram Showing Sequence of Message Flows

```
           DosCopy                                 .
              .                                    .
     DrgFreeDragtransfer                           .
              .                                    .
              .         DM_RENDERCOMPLETE          .
              .----------------------------------->.
              .                                    .
              .         DM_ENDCONVERSATION         .
              .<-----------------------------------.
              .                                    .
          DosDelete                       DrgFreeDragtransfer
              .                                    .
       DrgFreeDraginfo                    DrgDeleteStrHandle
              .                                    .
              .                             DrgFreeDraginfo
              .                                    .
              .                                    .
```

Figure 12-3 (Part 2 of 2). Diagram Showing Sequence of Message Flows

Canceling a Drag Operation

The user can end a direct manipulation operation in one of the three following ways:

- Pressing the Esc key to cancel the operation
- Releasing the drag button when the pointer is over a target that cannot accept the drop

  This action is equivalent to pressing the Esc key. When the pointer is over a target that can accept the drop, the target is informed of the drop, and the source is given the window handle of the target.
- Pressing the F1 key to request help

  A DM_DROPHELP message is posted to the target. This enables the target to provide the user with assistance regarding:

  – What would happen if the user dropped the object on that target
  – Why the target cannot accept a particular drop.

  The source sees this termination of the direct manipulation operation as a cancelation.

About Pickup and Drop

Pickup and Drop (also known as Lazy Drag) enables a drag operation to occur without requiring that the drag button be pressed for the duration of the operation (as in the standard direct manipulation operation). Pickup and Drop is non-modal in nature, allowing the user to interrupt the drag operation with other processes, and eliminating the requirement that both the source and target objects be visible prior to initiation of the drag operation (as in standard protocol). Pickup and Drop does not replace the standard, modal direct manipulation operation; it offers a more flexible alternative data transfer option.

Pickup and Drop is composed of one or more source object Pickup operations followed by a single Drop operation on a target object. Pickup and Drop is initiated by the first Pickup and is terminated by a Drop or Cancel Drag operation. When Pickup and Drop is initiated, the mouse pointer is augmented with the system Pickup pointer icon, as shown in Figure 12-4.

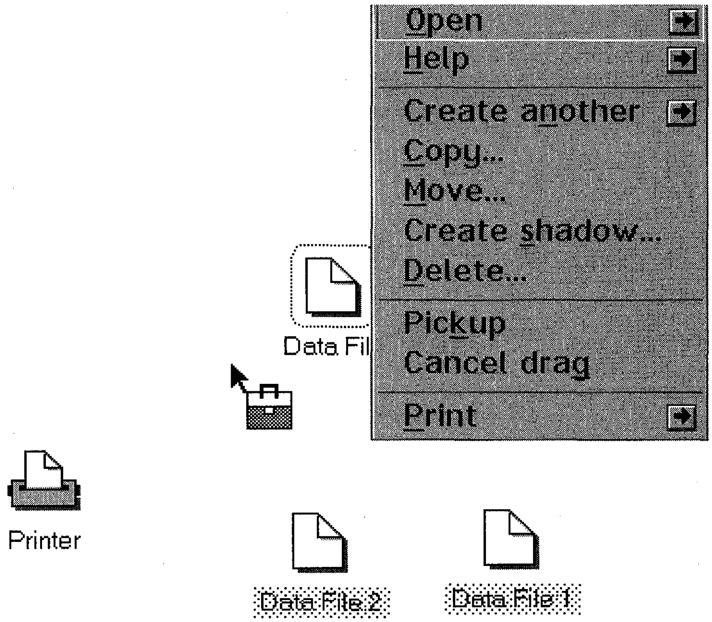

Figure 12-4. Pickup Mouse Icon, Popup Menu and Pickup Emphasis

The drag images seen in a standard direct manipulation operation are not displayed. As additional items are selected, they are added to the system Pickup set, and pickup emphasis is displayed for each item. The Pickup set is currently limited to a single source window or folder. While the operation is in progress, all other operations are valid with the exception of a standard direct manipulation operation.

Pickup and Drop is initiated by DrgLazyDrag in response to a WM_PICKUP message generated when the user presses Alt+mouse button 2 on a source object. As the pointer moves over a potential target, DM_DRAGOVER and DM_DRAGLEAVE messages are sent when the user presses a key indicating the intention to drop the object. The target emphasis is not displayed until the user attempts to drop the object. Each time items are added to the Pickup set in response to a WM_PICKUP message, DrgReallocDraginfo must be called to reallocate the DRAGINFO data structure. The Pickup and Drop operation is then re-initiated by another DrgLazyDrag call. DrgLazyDrag returns upon initialization for the operation. The pointing device remains active during the operation and may be used as if no drag operation were in effect. If the pointer is over a valid target when a Drop is invoked, a DM_DROP message is sent to the target, and a DM_DROPNOTIFY message posted to the source window.

DrgCancelLazyDrag is called to cancel the operation, and similarly posts a DM_DROPNOTIFY message to the source window, but with a target window handle of zero in the *mp2* parameter.

DrgLazyDrop can be used to programmatically invoke a drop operation; for example, from a menu choice.

DrgQueryDraginfoPtrFromHwnd and DrgQueryDraginfoPtrFromDragitem are called to query the DRAGINFO pointer at any time during the course of the operation.

DrgQueryDragStatus is called to determine whether a Pickup and Drop operation is currently in progress.

Data Structure Handling

Prior to initiating a Pickup and Drop operation (via DrgLazyDrag), DrgAllocDraginfo must be called to allocate a DRAGINFO data structure. As additional objects are added to the Pickup set, the DRAGINFO and DRAGITEM data structures must be reallocated using DrgReallocDraginfo. This function unconditionally frees the existing DRAGINFO data structure passed to it, reallocates a new DRAGINFO data structure, and returns the pointer to the new data structure. The Pickup and Drop operation is then re-inititiated by another DrgLazyDrag call.

The DRAGIMAGE array is passed to DrgLazyDrag, so that compatibility with the drag operation is maintained. This allows the application to support Pickup and Drop, and standard drag operation with the same code. However, the drag images in the data structure are not used for display during Pickup and Drop, as the mouse pointer is augmented with a Pickup icon during the operation. As soon as DrgLazyDrag returns, the DRAGIMAGE array can be freed.

Message Handling

In the standard direct manipulation protocol, DrgDrag does not return until the drag set is dropped on a target window. Pickup and Drop is slightly different, and requires a change in the handling of a Drop. Since the operation is non-modal, DrgLazyDrag returns as soon as it has completed drag initialiation and before a drop is performed. In the Pickup and Drop protocol, DM_DROPNOTIFY is posted to the source window as notification of a drop. The parameters of this message contain the pointer to the DRAGINFO data structure allocated by the source window and the handle of the target window. The source window should examine the *mp2* parameter to determine if the target window and the source window are the same; if not, the source should free the DRAGINFO upon receipt of this message. Where the target and source are the same, the target window frees DRAGINFO after completing the post-drop conversation. The implementation of Pickup and Drop does not affect any of the existing post-drop conversation messages.

The DM_DROPHELP message is not supported for the Pickup and Drop protocol, since help could be requested for any subject at any point during the operation. If the application is to provide Drop help, it must do so from a menu choice and explicitly code the support to be provided.

About Rendering Mechanisms

The following sections describe the standard rendering mechanisms used by various containers and applications for direct manipulation.

OS/2 File Rendering Mechanism

This rendering mechanism can be used by various containers, including file folders and trash cans. These containers allow objects to be dragged and dropped on white space in the container to accomplish a Move or Copy operation. They also can allow objects in the same or another container to be dragged and dropped on objects within the container to accomplish an operation.

Mechanism Name

The string for this rendering mechanism is DRM_OS2FILE.

Messages

The following messages are used by the DRM_OS2FILE:

- DM_RENDER

  This message is sent by a target to a source to request a rendering for an object. When this message is received, the source determines if it understands the rendering mechanism and format selected by the target for the object. It also confirms that it allows the operation selected by the user for that object. The source must respond to this message before proceeding with the rendering operation.

- DM_RENDERCOMPLETE

  This message is posted by a source to a target to notify the target that the rendering operation has been completed by the source, either successfully or unsuccessfully. The source can elect to let the target retry a successful or an unsuccessful operation. In this case, it should return to its state at the time of the drop for that object and indicate, in the message, that a retry is allowed.

  Support for this message by a source is optional. If this message is not supported, then:

  - The source must convey all necessary information to the target in order to allow it to handle the rendering operation.
  - It must always indicate that native rendering is allowed when replying to a DM_RENDER message.

- DM_ENDCONVERSATION

  This message is sent by a target to a source to notify the source that the rendering operation is complete and that the conversation is terminated. When this message is received, the entire drop operation for the object is complete. The source can now release any resources it had allocated to the drop and rendering operations. When the reply is received, the target can release the resources it had allocated to the operation.

Native Mechanism Actions

If the target understands the native rendering mechanism and format of the object, it may be possible to render the object without any involvement on the part of the source, provided the source has given the target sufficient information to do so. In order for the rendering to be performed by the target, the source must fill in, at a minimum, the *hstrContainerName* and *hstrSourceName* fields. The *hstrContainerName* field represents the subdirectory that the file indicated by *hstrSourceName* is in. For the target to do the rendering on its own, the true type of the object must be DTYP_OS2FILE. When these conditions are met, the target may proceed with the operation. When the operation is complete, the target must send a DM_ENDCONVERSATION message to the window indicated by *hwndItem* in the DRAGITEM data structure.

Preventing a Target from Rendering an Item

A source can prevent a target from doing the rendering operation on its own by not providing the source name for the object. This may be a necessary action for sources that implement some type of security, or that may not allow particular operations to be performed for an object move. When a source takes this course, it must fill in the *hstrSourceName* in the DRAGITEM data structure before replying to a DM_RENDER message. The target deletes the *hstrSourceName* string handle prior to freeing the DRAGINFO data structure, just as it would if the information had been passed to it at the time of the drop.

Requesting the Source to Render the Item

Whenever the conditions for a target to do the rendering operation without source participation are not met, the target must request the source to carry out the rendering by posting a DM_RENDER message to the source. Of course, the target can do this even if it is able to carry out the rendering mechanism on its own.

Allocating and Freeing a DRAGTRANSFER Data Structure

The data in a drag transfer message is carried in a DRAGTRANSFER data structure. DRAGTRANSFER data structures are allocated when the target calls DrgAllocDragtransfer.

When the conversation is completed, both the source and the target must call DrgFreeDragtransfer to free the shared memory. The target should do it immediately after sending a DM_ENDCONVERSATION message. The source should do it immediately after sending a DM_RENDERCOMPLETE message.

Operation Specifics

Regardless of the operation being performed, the target must fill in the *hstrRenderToName* field in the DRAGTRANSFER data structure before sending a DM_RENDER message. This is the fully qualified drive, path, and file name of the file that will contain the data when the rendering operation is complete. When the source has completed the operation, it must post a DM_RENDERCOMPLETE message to the target. The target then must complete the direct manipulation operation for that object by posting a DM_ENDCONVERSATION message to the source. Once the conversations for all of the objects involved in the drop are complete, the target can delete the string handles and free the DRAGINFO data structure.

Non-Native Mechanism Actions

The target may select the DRM_OS2FILE rendering mechanism when it is not the native rendering mechanism for an object, as long as the source supports it. In this case, the target must always request that the source carry out the rendering operation as described above. The source should render the data in the requested format to the file specified by the *hstrRenderToName* field. If the requested operation is a Move, the source should take whatever action is necessary to remove its knowledge of the object as long as no information regarding the object was lost in the transfer.

Naming Conventions

The naming conventions for this rendering mechanism are as follows:

- hstrContainerName

  Contains the fully qualified drive and path name for the source file, for example:

  ```
  C:\
  C:\MYSUBDIR\
  A:\SUBDIR1\SUBDIR2\
  \\NETWORK\SHARED\SUBDIRA\SUBDIRB\
  ```

- hstrSourceName

  Contains the name of the source file or subdirectory, for example:

  ```
  MYSOURCE.C
  MYSOURCE.H
  MYSOURCE IS A LONG FILE NAME
  SUBDIR3
  ```

  If you specify a subdirectory, the action is applied to all files in the subdirectory.

- hstrRenderToName

  Contains the fully qualified file or subdirectory name that is to be used at the target, for example:

  ```
  C:\MYSUBDIR\MYSOURCE.C
  \\NETWORK\SHARED\SUBDIRA\SUBDIRB\MYSOURCE.H
  C:\SUBDIR1\SUBDIR2\SUBDIR3
  ```

Types

Any type that is allowed as a *.TYPE* extended attribute is allowed in the *hstrType* field of the DRAGITEM data structure. The type for a file can be obtained using DosQFileInfo; the type can be set by using DosSetFileInfo.

Print Rendering Mechanism

A common object that might be provided by a container is a printer. This object would allow objects to be dragged and dropped on it to accomplish a print operation.

Mechanism Name

The string for this rendering mechanism is DRM_PRINTOBJECT.

Messages
To support this rendering mechanism, a source must be able to receive and process a DM_PRINTOBJECT message. The target posts this message to the source. When the message is received, the source prints the current view of the object identified in the message to the printer. The second message parameter (of type PRINTDEST) gives all the parameters necessary to call DevPostDeviceModes and DevOpenDC.

Native Mechanism Actions
There are no native mechanism actions for this rendering mechanism, because the act of printing an object is considered a transform from the native rendering mechanism to the print mechanism.

Naming Conventions
None.

Dynamic Data Exchange (DDE) Rendering Mechanism
This rendering mechanism can be used by various containers and applications. The containers allow objects to be dragged and dropped on white space in the container to accomplish a Move or Copy operation. They can also allow objects in the same or another container to be dragged and dropped on objects within the container to accomplish some operation.

Mechanism Name
The string for this rendering mechanism is DRM_DDE.

Messages
To support this rendering mechanism, a source must be able to receive and process the following messages:

- WM_DDE_REQUEST

  This message is posted by the target to the window indicated by the *hwndItem* field in the DRAGITEM data structure to request information regarding the object. Note that WM_DDE_INITIATE is not required because the target already has the handle of the window it wants to converse with. This message is sent for all Move and Copy operations.

- WM_DDE_ADVISE

  This message is posted by the target to the window indicated by the *hwndItem* field in the DRAGITEM data structure order to maintain a *hot link* to the object.

- WM_DDE_UNADVISE

  This message is posted by the target to the window indicated by the *hwndItem* field in the DRAGITEM data structure to terminate a hot link to the object.

- WM_DDE_TERMINATE

  This message is posted by the target to the window indicated by the *hwndItem* field in the DRAGITEM data structure to terminate a conversation.

To support this rendering mechanism, a target must be able to receive and process the following messages:

- WM_DDE_DATA

  This message is posted to the target by the source to deliver the requested information regarding the object.

- WM_DDE_ACK

  This message is posted to the target by the source to acknowledge a WM_DDE_ADVISE or WM_DDE_UNADVISE message.

- WM_DDE_TERMINATE

  This message is posted to the target by the source to end a conversation.

Native Mechanism Actions

Prior to establishing a DDE conversation, the target should determine the source-supported formats in which it wants to have the object rendered. It should register this format in the system atom table and use the resulting atom in the *usFormat* field of the DDESTRUCT used in the conversation.

The target should establish the DDE conversation by posting a WM_DDE_REQUEST message to the window indicated by the *hwndItem* field in the DRAGITEM data structure. The target acts as the client, and the source acts as the server in the conversation.

Operation Specifics

The following actions should be taken by the source, depending on the operation being performed:

Copy Send the data to the target.

Move Remove knowledge of the object after receiving confirmation that the target has successfully completed its portion of the rendering operation.

Non-Native Mechanism Actions

The target and source proceed in the same way, regardless of whether DDE was the native rendering mechanism or an alternate rendering mechanism.

Naming Conventions

The naming conventions for the DRM_DDE rendering mechanism follow:

- hstrSourceName

  Contains the object name to be used in the DDE conversation.

- hstrRMF

  The format portion of the list of ordered pairs in the format *<DRM_DDE,format>* identifies the formats supported by the source for the object. The non-standard DDE formats that these formats map to must be registered in the system atom table by both the source and the target.

Types

Any type that is allowed as a *.TYPE* extended attribute is allowed in the *hstrType* field of the DRAGITEM data structure.

Application-Defined Rendering Mechanisms

An application can choose to define a new rendering mechanism. However, if an application intends to provide renderings from this extended rendering mechanism to existing rendering mechanisms, it should publish enough information so that other application developers can use the new mechanism. An application must address several distinct areas of definition. These areas are described below, in general, and also are addressed under the definition for the system mechanisms.

Mechanism Name

The string name of the rendering mechanism should be defined by the application. This string name is specified in the mechanism/format pair of the DRAGITEM data structure.

Native Mechanism Actions

When both a source and target application store the data in the same native mechanism, a transform is not required. Instead, the native Move and Copy actions for that mechanism can be performed by the target. An application must completely define the proper procedure for performing that action. In the case of files, the native Move action is defined as a DosMove or DosCopy/DosDelete. The native Copy action is DosCopy. An application need not support all of the basic actions; it can choose to define additional native mechanism actions, indicated by the DO_UNKNOWN action in the DRAGINFO data structure.

Naming Conventions

An application that is defining a new mechanism must completely specify the naming conventions for objects rendered in that mechanism. This information typically includes both the name of the data and preceding information describing the exact location of the data. Any special rules concerning uppercase and lowercase or character sets to be used in naming also must be specified. The semantics for using these mechanism names, as well as an algorithm for deriving location information, also must be defined.

An application that is defining a new rendering mechanism must completely define the set of messages that a target and source application must support, and must specify the appropriate action to be taken for each message. The message IDs (above WM_USER) for the messages must be published.

Performance Considerations

If an application provides or defines transforms from the newly defined mechanism to existing mechanisms, performance information about the transform between mechanisms should be provided. This aids the application developer in choosing the appropriate transform when it encounters an application that transforms from an unknown native mechanism to several different known mechanisms.

Using Direct Manipulation

This section shows the sequence of functions and message flows for a typical direct manipulation operation. This section also describes the activities that must be performed by the applications during direct manipulation.

Note: Much of the sample codes in this section are part of a complete program illustrated in "Sample Code for Direct Manipulation" on page 12-32.

Allocating Memory for the Drag Operation

To prepare for the drag operation, the source must invoke DrgAllocDraginfo to allocate memory for the DRAGINFO data structure. DrgAllocDraginfo initializes the DRAGINFO data structure as follows:

cbDraginfo	The size, in bytes, of the entire DRAGINFO data structure, including the DRAGITEM array
cbDragitem	The size, in bytes, of each DRAGITEM data structure
usOperation	DO_DEFAULT
xDrop **and** *yDrop*	The current mouse-pointer location, in desktop coordinates
cditem	The count of objects being dragged, as specified in DrgAllocDraginfo.

Initializing DRAGITEM Data Structure

After allocating memory for the DRAGINFO data structure, the source initializes a DRAGITEM data structure, as appropriate, for each of the objects to be dragged. This is accomplished either by using DrgSetDragitem or by obtaining a pointer to each DRAGITEM data structure with DrgQueryDragitemPtr, and initializing it directly.

The first step the source takes to initialize the DRAGITEM data structure is to create the appropriate drag string handles. String handles must be created for:

- Object type
- Supported rendering mechanisms and formats for the object
- Suggested name of the object at the target
- Name of the container holding the object (whether a container or folder)
- Name of the object at the source when the source allows the target to carry out the operation for the object.

Type

To directly manipulate an object, both the source and the target must support the object *type*, which describes the format of the object. For example, the input to a C compiler could have the type *Plain Text* (DRT_TEXT). The *hstrType* field in the DRAGITEM data structure conveys this information for each object being dragged. The type is represented by a string handle. The target should check to see if it supports the type before allowing the user to drop the object.

Several DTYP_* constants are defined as *notational conveniences* for common types of data. An application can extend these types by defining its own character strings and then creating string handles for them using DrgAddStrHandle.

True Type

The *true type* of an object is the type that most accurately describes the object. For example, the input to a C compiler could have the type *Plain Text* (DRT_TEXT), but would be more accurately described as *C Code* (DRT_C). *C Code* would be the true type of this object. Multiple types can be conveyed by using a comma to separate strings. Figure 12-5 shows the format to use to convey multiple types.

```
"type,type..."
```

Figure 12-5. Format to Use to Convey Multiple Types

The true type should appear first in the list of types, so the type string for the example object would be "C Code, Plain Text".

Rendering Mechanism and Format

The rendering mechanism and format are passed as a string handle in the DRAGITEM data structure. The string handle must be created using DrgAddStrHandle. Figure 12-6 shows the string handle format.

```
"elem {,elem,elem...}"
```

Figure 12-6. String Handle Format

where *elem* is an ordered pair in the form:

```
"<mechanism,format>"
```

or a cross product in the form:

```
"(mechanism{,mechanism...}) X (format{,format...})"
```

Multiple cross products are permitted in a single rendering mechanism and format string handle, as are combinations of ordered pairs and cross products. When cross-product notation is used, the rendering mechanism is the left operand. When ordered-pair notation is used, the rendering mechanism is the left element in the ordered pair.

Several constants are defined for common rendering mechanisms and formats. For example, the rendering mechanism and format for a:

- C source file might be "<DRM_OS2FILE,CF_OEMTEXT>"
- Spreadsheet file might be "<DRM_OS2FILE,CF_SYLK>"

An application can extend these by defining its own "*<mechanism,format>*" strings and creating string handles for these using DrgAddStrHandle. For example, if an application understands and can generate an LU 6.2 data stream, it can define its own rendering format, "DRF_LU62," and use it in direct manipulation operations. If an application wishes to use its own rendering mechanisms or formats to communicate with other applications, it should publish the protocol for the mechanisms, the format of the data streams, or both.

Native Rendering Mechanism and Format

The native rendering mechanism and format of the object is the mechanism that most naturally conveys the data and its current format. For example, the native rendering mechanism and format for a:

- C source file might be "<DRM_OS2FILE,CF_OEMTEXT>"
- Spreadsheet file might be "<DRM_OS2FILE,CF_SYLK>"

In some direct-manipulation operations, it might be possible for the target to carry out the necessary action on the source object without the source's participation. However, this is possible only when the target supports both the true type and the native rendering mechanism and format of the object. Even when the target is not performing the necessary action on the source object, it is still important to know the native rendering mechanism and format. In determining the rendering mechanism and format to be used in the data exchange after the drop, the target might select the native format because, generally, performance is better when the native rendering mechanism and format are used.

The native rendering mechanism and format are conveyed to the target by making it the first ordered pair, or the first ordered pair to result from a cross product, in the list of rendering mechanisms and formats passed in the DRAGINFO data structure.

Suggested Name at Target

When dragging an object, for example, a file, from one container to another, it is important to know the name the object should have at the target. This may or may not be the same name it had at the source. This name enables the target to check if another object with the same name already exists at the target and to take the appropriate action. For example, a target container might not allow the user to drop the object if an object by that same name already exists at the target.

Container Name

Sometimes it is necessary for a target container to know the name of the source container. This name could carry some location information. For example, the default operation when dragging objects between containers is a Move. However, in the case of file folders on different drives, this default would be changed to a Copy operation. Thus, a file folder would fill this field with the drive and path information for a file, for example, A:\SUBDIR1\SUBDIR2\. A database container, on the other hand, might fill this field with the fully qualified OS/2 file name of the database.

Source Name

In some direct-manipulation operations, it is possible for the target to perform the necessary action on the source object without the source's participation. If the source allows this, the target name should be filled in with the name of the source object. For example, a file folder would put the name of the source file into this field, such as AUTOEXEC.BAT. A database manager, on the other hand, might fill this field with some location information so the target could find a particular record or field within the database.

Sample Code for Initializing DRAGITEM Data Structure

The sample code fragment illustrated in Figure 12-7 shows how to initialize the DRAGITEM array.

```
/**********************************************************************/
/*  Get our current directory for the container name.                 */
/**********************************************************************/
   dirlen                      = CCHMAXPATH-1;
   DosQueryCurrentDir(0, szDir, &dirlen);
   sprintf(szContainer, "\\%s\\", szDir);
   hstrContainer               = DrgAddStrHandle(szContainer);
   Dragitem.hwndItem           = hListWnd;
   Dragitem.hstrType           = hstrType;
   Dragitem.hstrRMF            = hstrRMF;
   Dragitem.hstrContainerName = hstrContainer;
   Dragitem.fsControl          = 0;
   Dragitem.fsSupportedOps     = DO_COPYABLE | DO_MOVEABLE;
   Dragitem.hstrSourceName     = DrgAddStrHandle (szBuffer);
   Dragitem.hstrTargetName     = Dragitem.hstrSourceName;
   Dragitem.ulItemID           = index;

/**********************************************************************/
/*  Set info, prepare for drag.                                        */
/**********************************************************************/
   DrgSetDragitem(pSourceDraginfo,
                  &Dragitem,
                  sizeof(DRAGITEM),
                  0);
```

Figure 12-7. Sample Code for Initializing the DRAGITEM Array

Initializing DRAGIMAGE Data Structure

As part of the preparation for the actual drag, an application intializes a DRAGIMAGE data structure. The sample code illustrated in Figure 12-8 on page 12-22 shows how to initialize the DRAGIMAGE data structure.

```
/**********************************************************************/
/*  Initialize the drag image.                                        */
/**********************************************************************/
   dimg.cb       = sizeof (DRAGIMAGE);
   dimg.hImage   = WinQuerySysPointer (HWND_DESKTOP, SPTR_FILE, FALSE);
   dimg.fl       = DRG_ICON | DRG_TRANSPARENT;
   dimg.cxOffset = 0;
   dimg.cyOffset = 0;
```

Figure 12-8. Sample Code for Initializing the DRAGIMAGE Data Structure

Starting the Drag Operation

Once initialization is complete, the source object calls DrgDrag to start the direct manipulation operation. The sample code illustrated in Figure 12-9 shows how to start the drag operation.

```
/**********************************************************************/
/*  Start drag operation.                                             */
/**********************************************************************/
  DrgDrag(hFrameWnd,
          pSourceDraginfo,
          &dimg,
          1L,
          VK_BUTTON2,
          NULL);
```

Figure 12-9. Sample Code for Starting the Drag Operation

Responding to the DM_DRAGOVER Message

The DM_DRAGOVER message is sent to a target whenever the user drags the pointer into the window. To assess whether a drop can be accepted, the target must use DrgAccessDraginfo to get access to the DRAGINFO data structure. It then determines whether a drop can be accepted for each object. The object must meet the following minimum requirements to exchange data:

- The source and target must share knowledge of at least one common type for the object. The target can make this determination by using DrgVerifyTypeSet or DrgVerifyType.
- The source and target must share at least one common rendering mechanism and format for that type object. The target can make this determination by using DrgVerifyRMF.

DOR_DROP, DOR_NODROP, DOR_NODROPOP, and DOR_NEVERDROP are the four possible responses available to the target when it receives a DM_DRAGOVER message.

The target sends these values to the window handle specified in the DRAGINFO data structure.

The sample code illustrated in Figure 12-10 shows how the target determines its response to the DM_DRAGOVER message.

```
/**********************************************************************/
/*  Someone's dragging an object over us.                             */
/**********************************************************************/
        case DM_DRAGOVER:
         dragInfo = (PDRAGINFO)mp1;

         /* Get access to the DRAGINFO data structure */
         DrgAccessDraginfo(dragInfo);

         /* Can we accept this drop? */
         switch (dragInfo->usOperation)
         {

          /* Return DOR_NODROPOP if current operation */
          /* is link or unknown                       */
           case DO_UNKNOWN:
              DrgFreeDraginfo(dragInfo);
              return (MRFROM2SHORT(DOR_NODROPOP,0));
              break;

           /* Our default operation is Move */
           case DO_DEFAULT:
              dragItem = DrgQueryDragitemPtr(dragInfo,0);
              ulBytes  = DrgQueryStrName(dragItem->hstrContainerName,
                                                   sizeof(szDir),
                                                   szDir);
              if (!ulBytes)
                return (MRFROM2SHORT(DOR_NODROPOP,0));
              else
                usOp = DO_MOVE;
              break;
```

Figure 12-10 (Part 1 of 2). Sample Code Showing the Target's Response to DM_DRAGOVER

```
        /* Do the requested specific operation */
        case DO_MOVE:
        case DO_COPY:
           usOp = dragInfo->usOperation;
           break;
      }

      usIndicator = DOR_DROP;
      cItems = DrgQueryDragitemCount(dragInfo);

       /* Now, we need to look at each item in turn */
       for (i = 0; i < cItems; i++)
       {
         dragItem = DrgQueryDragitemPtr(dragInfo, i);

         /* Make sure we can move for a Move request */
         /* or copy for a Copy                       */
         if (((dragItem->fsSupportedOps & DO_COPYABLE)   &&
              (usOp == (USHORT)DO_COPY))                 ||
             ((dragItem->fsSupportedOps & DO_MOVEABLE)   &&
              (usOp == (USHORT)DO_MOVE)))
         {
            /* Check the rendering format */
            if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
               usIndicator = DOR_DROP;
            else
               usIndicator = DOR_NEVERDROP;
         }
         else
            usIndicator = DOR_NODROPOP;
       }

       /* Release the draginfo data structure */
       DrgFreeDraginfo(dragInfo);

       return (MRFROM2SHORT(usIndicator, usOp));
       break;
```

Figure 12-10 (Part 2 of 2). Sample Code Showing the Target's Response to DM_DRAGOVER

Providing Target Emphasis

The target should provide target emphasis so the user knows exactly where the drop occurs or, if the drop is not allowed, the boundaries of the region where the drop is not allowed.

A container window should emphasize a target object by drawing a thin, black rectangle around it. The application should use DrgGetPS and DrgReleasePS to obtain the presentation space in which to draw target emphasis.

Providing Customized Images

The target can provide a customized pointer to be displayed while it is the target of the drop by calling DrgSetDragPointer after it starts processing the DM_DRAGOVER message but before it sends a response. It also can provide a customized image (icon, bit map, and so forth) to be displayed while it is the target by calling DrgSetDragImage. This capability may be used by a target to provide additional visible feedback to the user. The pointer is reverted to the default when it is moved to a new target.

Responding to the DM_DRAGLEAVE Message

DM_DRAGLEAVE is sent whenever the DM_DRAGOVER message is sent to a window, and the pointer is moved outside the bounds of that window. If the target or an object in the window had been emphasized as a target, it should be de-emphasized.

Container windows monitor the position of the pointer on DM_DRAGOVER messages and simulate the DM_DRAGLEAVE message when the pointer moves on or off a contained object.

A DM_DRAGLEAVE message is not sent if the user drops the objects being dragged within the window. Therefore, when DM_DROP is received, the application de-emphasizes any target that was emphasized as a valid target.

If the user drags the pointer outside the target window, resulting in a new target, a DM_DRAGLEAVE message is sent to the former target. The receiver of a DM_DRAGLEAVE message should use it to de-emphasize the target, thus providing the user with visible feedback that this is no longer the target.

Responding to the DM_DROP Message

When the user drops the objects, a DM_DROP message is sent to the target, providing it with the information necessary to process the objects that were dropped. The target application uses the information provided to exchange data with the source. The target is responsible for establishing the appropriate conversations, and the source must cooperate in establishing the necessary conversations to achieve the actual data exchange. After completing the direct manipulation operation, including the post-drop conversation with the source, the target uses DrgDeleteStrHandle or DrgDeleteDraginfoStrHandles to delete the string handles in the DRAGINFO data structure, and DrgFreeDraginfo to release the storage. The target should immediately remove any target emphasis. The data transfers must not be done before responding to the DM_DROP message.

The sample code illustrated in Figure 12-11 on page 12-26 shows how a target processes an object that has been dropped on it. This code fragment is part of a complete program which is illustrated in "Sample Code for Direct Manipulation" on page 12-32.

```
/* Drop the object on us (receive the object) */
case DM_DROP:

 /* Get access to the DRAGINFO data structure */
 DrgAccessDraginfo(dragInfo);

 /* Can we accept this drop? */
 switch (dragInfo->usOperation)
 {

   /* Return DOR_NODROPOP if current */
   /* operation is link or unknown.  */
   case DO_UNKNOWN:
      DrgFreeDraginfo(dragInfo);
      return (MRFROM2SHORT (DOR_NODROPOP, 0));
      break;

   /* Our default operation is Move */
   case DO_DEFAULT:
      dragItem = DrgQueryDragitemPtr(dragInfo, 0);
      ulBytes = DrgQueryStrName(dragItem->hstrContainerName,
                                sizeof(szDir),
                                szDir);
      if(!ulBytes)
        return (MRFROM2SHORT (DOR_NODROPOP, 0));
      usOp = (USHORT)DO_MOVE;
      break;

   /* Do the requested specific operation */
   case DO_MOVE:
   case DO_COPY:
      usOp = dragInfo->usOperation;
      break;
  }

  usIndicator = DOR_DROP;
  cItems = DrgQueryDragitemCount(dragInfo);

  /* Now, we need to look at each item in turn */
  for (i = 0; i < cItems; i++)
  {
    dragItem = DrgQueryDragitemPtr(dragInfo, i);
```

Figure 12-11 (Part 1 of 2). Sample Code Showing the Drop of an Object on a Target

```
        /* Make sure we can move for a Move request */
        /* or copy for a Copy                       */
        if (((dragItem->fsSupportedOps & DO_COPYABLE)   &&
             (usOp == (USHORT)DO_COPY))                  ||
            ((dragItem->fsSupportedOps & DO_MOVEABLE)   &&
             (usOp == (USHORT)DO_MOVE)))

        {
           /* Check the rendering format */
           if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
              usIndicator = DOR_DROP;
           else
              usIndicator = DOR_NEVERDROP;
        }
        else
           usIndicator = DOR_NODROPOP;

/**********************************************************************/
/*  This is where we would actually move or copy the file,            */
/*  but we just display the name instead.                             */
/**********************************************************************/
        DrgQueryStrName(dragItem->hstrSourceName, 255, szBuffer);
        WinMessageBox(HWND_DESKTOP,
                      HWND_DESKTOP,
                      szBuffer,
                      "Dropped",
                      0,
                      MB_OK);
      }
      /* Release the draginfo data structure */
      DrgFreeDraginfo(dragInfo);

      return (MRFROM2SHORT(usIndicator, usOp));
      break;
```

Figure 12-11 (Part 2 of 2). Sample Code Showing the Drop of an Object on a Target

Exchanging Data

Direct manipulation offers various ways for source and target applications to exchange data. To accomplish the exchange, a separate conversation must be established to transfer each data object from the source to the target. The target must inform the source about the rendering mechanism it is using and the format in which the data is to be exchanged. The target can establish the conversations to run in parallel, or it can initiate the conversations in a serial fashion.

The target determines which rendering mechanism and format to use in the following manner:

1. Uses the native rendering mechanism and format whenever possible.

   This rendering conveys all information about the data. A target can determine if it supports the native rendering mechanism and format by using the following functions:

   - DrgVerifyNativeRMF
   - DrgQueryNativeRMFLen
   - DrgQueryNativeRMF.

   Even if it can use the native rendering mechanism and format supported by the source, the target can elect to exchange the data in a rendering mechanism and format that conveys less information about the object.

2. Uses the next best rendering mechanism and format.

   This is especially good for a Copy operation, because the user does not lose data about the object as occurs when the object is moved.

   The target can determine the next best rendering mechanism and format to use through repeated calls to DrgVerifyRMF. The calls are made starting with the most desirable rendering mechanism and format pair and progressing to the least desirable pair. Once a pair that the source supports has been found, the target can exchange the data.

The sample code illustrated in Figure 12-12 shows how the target checks the rendering mechanism and format.

```
/* Now, we need to look at each item in turn */
for (i = 0; i < cItems; i++)
{
  dragItem = DrgQueryDragitemPtr(dragInfo, i);

  /* Make sure we can move for a Move request */
  /* or copy for a Copy                       */
  if (((dragItem->fsSupportedOps & DO_COPYABLE)   &&
       (usOp == (USHORT)DO_COPY))                 ||
      ((dragItem->fsSupportedOps & DO_MOVEABLE)   &&
       (usOp == (USHORT)DO_MOVE)))
  {
```

Figure 12-12 (Part 1 of 2). Sample Code Showing how the Target Checks the Rendering Mechanism

```
            /* Check the rendering format */
            if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
               usIndicator = DOR_DROP;
            else
               usIndicator = DOR_NEVERDROP;
         }
         else
            usIndicator = DOR_NODROPOP;
      }
```

Figure 12-12 (Part 2 of 2). Sample Code Showing how the Target Checks the Rendering Mechanism

Performance Considerations

When context information about an object might be lost because of using a less-desirable rendering mechanism and format, the target can elect to pick a common mechanism and format that achieves the best performance. This is done the same way that the next best rendering mechanism and format is selected, proceeding from the best-performing rendering to the worst.

Regardless of the rendering mechanism used, the target might need to prepare the source for the rendering of the object. This is necessary when the source needs to create a window in order to handle the conversation. This preparation is done by sending a DM_RENDERPREPARE message to the *hwndSource* window in the DRAGINFO data structure. This message need be sent only when the DC_PREPARE flag is on in the *fsControl* field of the DRAGITEM data structure. When the source receives this message, it performs any necessary preparation for the rendering and fills in the *hwndItem* field in the DRAGITEM data structure, thereby allowing the target to establish conversation with that window.

Using Pickup and Drop

The sample code illustrated in Figure 12-13 on page 12-30 shows a Pickup and Drop operation after the user has selected an object and pressed mouse button 2 while holding down the Pickup and Drop augmentation key (Alt).

```
#define INCL_WINSTDDRAG #include <os2.h>

PDRAGINFO pdinfo;            /* Pointer to a DRAGINFO data structure    */
HWND hwndSource;             /* Handle of the Source window             */
DRAGITEM ditem;              /* DRAGITEM data structure                 */
PDRAGIMAGE pdimg;            /* Pointer to DRAGIMAGE data structure     */
HBITMAP hbm;                 /* Bit-map handle passed to DrgLazyDrag    */

case WM_PICKUP:

/*********************************************************************/
/*  Initialize the DRAGITEM data structure.                          */
/*********************************************************************/
ditem.hwndItem=hwndSource;         /* Handle of the source window       */
ditem.ulItemID=ID_ITEM;            /* App. defined id of item           */

ditem.hstrType          = DrgAddStrHandle("DRT_TEXT");  /* Text item   */
ditem.hstrRMF           = DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
ditem.hstrContainerName = DrgAddStrHandle("C:\\");
ditem.hstrSourceName    = DrgAddStrHandle("C:\\CONFIG.SYS");
ditem.hstrTargetName    = DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");

ditem.cxOffset         = 0;    /* Offset of the origin of the image    */
ditem.cyOffset         = 0;    /* from the pointer hotspot             */
ditem.fsControl        = 0;    /* Source item control flags            */
ditem.fsSupportedOps   = 0;

/*********************************************************************/
/*  Create the DRAGINFO data structures                              */
/*********************************************************************/
pdinfo=DrgAllocDraginfo(1);

/* Return FALSE if initialization fails */
if(!pdinfo) return FALSE;
```

Figure 12-13 (Part 1 of 2). Sample Code for a Pickup and Drop Operation

```
/************************************************************************/
/*  Initialize the DRAGIMAGE data structure.                            */
/************************************************************************/
pdimg=AllocMem(sizeof(DRAGIMAGE));

pdimg->cb=sizeof(DRAGIMAGE);       /* Size of the dragimage structure   */
pdimg->cptl=0;                     /* Image is not a polygon            */
pdimg->hImage=hbm;                 /* Handle of image to display        */
pdimg->sizlStretch.cx=20L          /* Size to stretch icon or bit map   */
pdimg->fl=DRG_BITMAP |             /* Flags passed to DrgLazyDrag       */
          DRG_STRETCH;

pdimg->cxOffset=0;                 /* Offset of the origin of image     */
pdimg->cyOffset=0;                 /* from the pointer hotspot          */

/************************************************************************/
/*  Set the DRAGITEM data structure.                                    */
/************************************************************************/
DrgSetDragitem(pdinfo, &ditem,(ULONG)sizeof(ditem, 0);

/************************************************************************/
/*  Begin the Lazy Drag operation.                                      */
/************************************************************************/

if (DrgLazyDrag(hwndSource,                /* Source of the drag        */
                pdinfo,                    /* Pointer to the DRAGINFO   */
                pdimg,                     /* DRAGIMAGE array           */
                1,                         /* Size of the DRAGIMAGE     */
                NULL))                     /* Reserved                  */
{
/* Free DRAGIMAGE if successful */
FreeMem (pdimg);
}
```

Figure 12-13 (Part 2 of 2). Sample Code for a Pickup and Drop Operation

Graphical User Interface Support for Direct Manipulation

This section describes the support the direct manipulation provides for graphical user interfaces (GUIs). Except where noted, this support conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

Keyboard Augmentation

A direct manipulation operation begins in a *default state*, which means that, when the user drops objects on a target, the target is informed that it should perform its default operation. The target is responsible for defining its default operation. For a container window, the default should be a Move operation, if it is supported. The default for a device, such as a printer, should be a Copy operation.

As the user drags the object, the default operation can be overridden by pressing and holding one of the following augmentation keys:

Ctrl Changes the operation to a Copy

Shift Changes the operation to a Move

Ctrl+Shift Changes the operation to a Link.

The last key pressed and held at the time of the drop determines the operation to be performed. The target can determine the defined augmentation key that was pressed at the time of the drop by inspecting the *usOperation* field of the DRAGINFO data structure.

A target can define additional augmentation keys for its own use. In this case, *usOperation* would indicate that the operation is unknown, and the target needs to use WinGetKeyState to determine the actual augmentation key that was used.

As the user presses augmentation keys, the pointer currently being displayed is modified to provide the user with a visible cue as to the type of operation being performed.

Sample Code for Direct Manipulation

This section illustrates a complete sample program for the drag portion of a drag-and-drop operation. Several parts of this program are explained in "Using Direct Manipulation" on page 12-18.

Source Application Sample Code

The source application includes the following files:

- Dragfrom.C
- Dragfrom.H
- Dragfrom.DEF
- Dragfrom.LNK
- Dragfrom.MAK

Figure 12-14 on page 12-33 shows the source application sample code.

```
===============
DRAGFROM.C
===============

/**********************************************************************/
/*  DRAGFROM.C - Drag source program                                  */
/*                                                                    */
/*  This program displays a list of files in the current directory.   */
/*  Drag any file name to EPM, and drop, and the file will be         */
/*  displayed in the editor.                                          */
/**********************************************************************/
#define INCL_DOSFILEMGR
#define INCL_WIN
#define INCL_WINSTDDRAG
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR

#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dragfrom.h"

/**********************************************************************/
/*  Global variables.                                                 */
/**********************************************************************/
HAB      hab;
char     szFormats[]  = "<DRM_OS2FILE, DRF_UNKNOWN>";
char     szFileNames[50][CCHMAXPATH];
HWND     hFrameWnd;
HWND     hListWnd;
PFNWP    SysWndProc;
PFNWP    ListWndProc;
HPOINTER hptrFile;

/**********************************************************************/
/*  Function prototypes.                                              */
/**********************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY LocalListProc(HWND, ULONG, MPARAM, MPARAM);
BOOL DoDrag(void);
void LoadList(void);
```

Figure 12-14 (Part 1 of 10). Sample Code for a Source Application

```
/************************************************************************/
/*   Main() - program entry point.                                      */
/************************************************************************/
int main(void)
{
   FRAMECDATA   fcd;
   HMQ          hmq;
   QMSG         qmsg;

   if (!(hab = WinInitialize (0)))
     return FALSE;

   hmq = WinCreateMsgQueue (hab, 0);

   if (!hmq)
   {
      WinTerminate(hab);
      return FALSE;
   }

/************************************************************************/
/*  Setup the frame control data for the frame window.                  */
/************************************************************************/
   fcd.cb = sizeof(FRAMECDATA);
   fcd.flCreateFlags = FCF_TITLEBAR      |
                       FCF_SYSMENU       |
                       FCF_SIZEBORDER    |
                       FCF_SHELLPOSITION |
                       FCF_MINMAX        |
                       FCF_TASKLIST;

   fcd.hmodResources = NULLHANDLE;

/************************************************************************/
/*  Set our resource key (so PM can find menus, icons, etc).           */
/************************************************************************/
   fcd.idResources = DRAGFROM;
```

Figure 12-14 (Part 2 of 10). Sample Code for a Source Application

```
/**************************************************************************/
/*  Create the frame - it will hold the list box.                         */
/**************************************************************************/
   hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                               WC_FRAME,
                               "Drag Source",
                               0, 0, 0, 0, 0,
                               NULLHANDLE,
                               HWND_TOP,
                               DRAGFROM,
                               &fcd,
                               NULL);

/**************************************************************************/
/*  Verify that the frame was created; otherwise, stop.                   */
/**************************************************************************/
   if (!hFrameWnd)
     return FALSE;

/**************************************************************************/
/*  Set an icon for the frame window.                                     */
/**************************************************************************/
   WinSendMsg(hFrameWnd,
              WM_SETICON,
              (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                         SPTR_FOLDER,
                                         FALSE),
               NULL);

/**************************************************************************/
/*  Create a list window child - we willl list files in it.               */
/**************************************************************************/
   hListWnd = WinCreateWindow(hFrameWnd,
                              WC_LISTBOX,
                              NULL,
                              0, 0, 0, 0, 0,
                              hFrameWnd,
                              HWND_BOTTOM,
                              FID_CLIENT,
                              NULL,
                              NULL);
```
Figure 12-14 (Part 3 of 10). Sample Code for a Source Application

```
/*********************************************************************/
/*  We must intercept the frame window's messages.                   */
/*  We save the return value (the current WndProc),                  */
/*  so we can pass it all the other messages the frame gets.         */
/*********************************************************************/
   SysWndProc  = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);
   ListWndProc = WinSubclassWindow (hListWnd, (PFNWP)LocalListProc);

   WinShowWindow(hFrameWnd, TRUE);
   WinPostMsg(hFrameWnd, WM_LOAD_LIST, 0, 0);

/*********************************************************************/
/*  Main message loop.                                               */
/*********************************************************************/
   while (WinGetMsg (hab, &qmsg, 0L, 0, 0))
     WinDispatchMsg (hab, &qmsg);

   WinDestroyWindow (hFrameWnd);
   WinDestroyMsgQueue (hmq);
   WinTerminate (hab);
}

/*********************************************************************/
/*  LocalWndProc() - intercepts frame window messages.               */
/*********************************************************************/
MRESULT EXPENTRY LocalWndProc (HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
   switch (msg)
   {
      /* Post a message to fill the list box */
      case WM_LOAD_LIST:
         LoadList();
         break;

      case WM_DESTROY:
         WinDestroyPointer (hptrFile);
          break;
```
Figure 12-14 (Part 4 of 10). Sample Code for a Source Application

```
    case WM_STARTDRAG:
       DoDrag();
       break;

    default:
       return (*SysWndProc)(hwnd, msg, mp1, mp2);
       break;
  }
  return FALSE;
}

/************************************************************************/
/*  LocalListProc() - List box subclassing                             */
/*  (all we care about is starting a drag).                            */
/************************************************************************/
MRESULT EXPENTRY LocalListProc(HWND hwnd,
                               ULONG msg,
                               MPARAM mp1,
                               MPARAM mp2)

{
  if (msg == WM_BUTTON2DOWN)
  {
    WinPostMsg(hFrameWnd, WM_STARTDRAG, mp1, 0);
    return (MRESULT)FALSE;
  }
  else
    return (*ListWndProc)(hwnd, msg, mp1, mp2);
}

/************************************************************************/
/*  DoDrag() - the actual drag function.                               */
/************************************************************************/
BOOL DoDrag ()
{
  char            szBuffer[CCHMAXPATH];
  char            szDir[256];
  SHORT           index, len;
  HWND            hTargetWnd;
  LHANDLE         hImage;
```

Figure 12-14 (Part 5 of 10). Sample Code for a Source Application

```
   DRAGITEM         Dragitem;
   HSTR             hstrType, hstrRMF, hstrContainer;
   CHAR             szItemName[64];
   CHAR             szContainer[CCHMAXPATH];
   PDRAGINFO        pSourceDraginfo;
   DRAGIMAGE        dimg;
   ULONG            dirlen;

/**********************************************************************/
/*  Get the file name from the listbox.                               */
/**********************************************************************/
   index = WinQueryLboxSelectedItem(hListWnd);
   len   = WinQueryLboxItemTextLength(hListWnd, index);
   WinQueryLboxItemText(hListWnd,
                        index,
                        szBuffer,
                        len);
   szBuffer[len] = '\0';

/**********************************************************************/
/*  Allocate the DRAGINFO data structure.                             */
/**********************************************************************/
   pSourceDraginfo = DrgAllocDraginfo(1);

/**********************************************************************/
/*  Define file type as unknown.                                      */
/**********************************************************************/
   hstrType  = DrgAddStrHandle (DRT_UNKNOWN);
   hstrRMF   = DrgAddStrHandle (szFormats);         /* OS2file unknown */

/**********************************************************************/
/*  Get our current directory for the container name.                 */
/**********************************************************************/
   dirlen                     = CCHMAXPATH-1;
   DosQueryCurrentDir(0, szDir, &dirlen);
   sprintf(szContainer, "\\%s\\", szDir);
   hstrContainer              = DrgAddStrHandle(szContainer);
   Dragitem.hwndItem          = hListWnd;
   Dragitem.hstrType          = hstrType;
```

Figure 12-14 (Part 6 of 10). Sample Code for a Source Application

```
   Dragitem.hstrRMF            = hstrRMF;
   Dragitem.hstrContainerName = hstrContainer;
   Dragitem.fsControl          = 0;
   Dragitem.fsSupportedOps     = DO_COPYABLE | DO_MOVEABLE;
   Dragitem.hstrSourceName     = DrgAddStrHandle (szBuffer);
   Dragitem.hstrTargetName     = Dragitem.hstrSourceName;
   Dragitem.ulItemID           = index;

/*********************************************************************/
/*  Set info, prepare for drag.                                      */
/*********************************************************************/
   DrgSetDragitem(pSourceDraginfo,
                  &Dragitem,
                  sizeof(DRAGITEM),
                  0);

/*********************************************************************/
/*  Initialize the drag image.                                       */
/*********************************************************************/
   dimg.cb       = sizeof (DRAGIMAGE);
   dimg.hImage   = WinQuerySysPointer (HWND_DESKTOP, SPTR_FILE, FALSE);
   dimg.fl       = DRG_ICON | DRG_TRANSPARENT;
   dimg.cxOffset = 0;
   dimg.cyOffset = 0;

   pSourceDraginfo->hwndSource = hFrameWnd;

/*********************************************************************/
/*  Start drag operation.                                            */
/*********************************************************************/
   DrgDrag(hFrameWnd,
           pSourceDraginfo,
           &dimg,
           1L,
           VK_BUTTON2,
           NULL);

   return TRUE;
}
```

Figure 12-14 (Part 7 of 10). Sample Code for a Source Application

```
/**********************************************************************/
/*  LoadList().                                                       */
/**********************************************************************/
void LoadList(void)
{
   char          szDir[CCHMAXPATH];
   FILEFINDBUF3  ffbFile;
   HDIR          hDir;
   int           rc, x;
   ULONG         dirlen;
   ULONG         count;

/**********************************************************************/
/*  We use a DosFindFirst/DosFindNext loop to fill the list box.      */
/**********************************************************************/
   hDir  = HDIR_CREATE;
   count = 1;
   rc = DosFindFirst("*.*",
                     &hDir,
                     0,
                     &ffbFile,
                     sizeof(FILEFINDBUF3),
                     &count,
                     FIL_STANDARD);

   x = 0;
   do
   {
      sprintf(szFileNames[x], "%s", ffbFile.achName);

      WinPostMsg(hListWnd,
                 LM_INSERTITEM,
                 MPFROMSHORT(LIT_END),
                 szFileNames[x]);
      count = 1;
      x++;
```

Figure 12-14 (Part 8 of 10). Sample Code for a Source Application

```
      rc = DosFindNext(hDir,
                       &ffbFile,
                       sizeof(FILEFINDBUF3),
                       &count);
   }
   while (count && (x < 50));

   DosFindClose(hDir);
}

===============
DRAGFROM.H
===============
#define DRAGFROM      100
#define WM_STARTDRAG  WM_USER+100
#define WM_LOAD_LIST  WM_USER+110

===============
DRAGFROM.DEF
===============
NAME            DRAGFROM WINDOWAPI

PROTMODE
HEAPSIZE        8192
STACKSIZE       32768
EXPORTS         LocalWndProc
                LocalListProc

===============
DRAGFROM.LNK
===============
dragfrom.obj
dragfrom.exe
dragfrom.map
dragfrom.def

===============
DRAGFROM.MAK
===============

CC      = icc /c /Ge /Gd- /Se /Re /ss /Gm+
LINK    = link386
HEADERS = dragfrom.h
```

Figure 12-14 (Part 9 of 10). Sample Code for a Source Application

```
#------------------------------------------------------------------------
#  A list of all of the object files.
#------------------------------------------------------------------------
ALL_OBJ1 = dragfrom.obj

all: dragfrom.exe

dragfrom.obj: dragfrom.c $(HEADERS)

dragfrom.exe: $(ALL_OBJ1) dragfrom.def dragfrom.lnk
              $(LINK) @dragfrom.lnk
```

Figure 12-14 (Part 10 of 10). Sample Code for a Source Application

Target Application Sample Code

The target application includes the following files:

- Target.C
- Target.RC
- Target.H
- Target.DEF
- Target.LNK

Figure 12-15 shows the target application sample code.

```
================
TARGET.C
================
#define  INCL_WIN
#define  INCL_GPI

#include <os2.h>
#include "target.h"

#pragma  linkage (main,optlink)
INT      main(VOID);
```

Figure 12-15 (Part 1 of 9). Sample Code for a Target Application

```
/**********************************************************************/
/*  Main() - program entry point.                                     */
/*  This program accepts drops from EPM.                              */
/**********************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);

HAB     hab;
HWND    hFrameWnd;
PFNWP   SysWndProc;

INT main (VOID)
{
   HMQ          hmq;
   HPOINTER     hPtr;
   FRAMECDATA   fcd;
   QMSG         qmsg;

   if (!(hab = WinInitialize(0)))
     return FALSE;

   if (!(hmq = WinCreateMsgQueue(hab, 0)))
     return FALSE;

/**********************************************************************/
/*  Set up the frame control data for the frame window.               */
/**********************************************************************/
   fcd.cb = sizeof(FRAMECDATA);
   fcd.flCreateFlags = FCF_TITLEBAR |
                       FCF_SYSMENU |
                       FCF_SIZEBORDER |
                       FCF_SHELLPOSITION |
                       FCF_MINMAX |
                       FCF_TASKLIST;
   fcd.hmodResources = NULLHANDLE;
   fcd.idResources = 0;
```

Figure 12-15 (Part 2 of 9). Sample Code for a Target Application

```
/**********************************************************************/
/*  Create the frame window.                                          */
/**********************************************************************/
   hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                               WC_FRAME,
                               "Target",
                               0,
                               0, 0, 0, 0,
                               NULLHANDLE,
                               HWND_TOP,
                               0,
                               &fcd,
                               NULL);

/**********************************************************************/
/*  Verify that the frame was created; otherwise, stop.               */
/**********************************************************************/
   if (!hFrameWnd)
     return FALSE;
   hPtr = WinLoadPointer(HWND_DESKTOP,
                         NULLHANDLE,
                         TRASHCAN);

/**********************************************************************/
/*  Set an icon for the frame window.                                 */
/**********************************************************************/
   WinSendMsg(hFrameWnd,
              WM_SETICON,
              (MPARAM)hPtr,
              NULL);

/**********************************************************************/
/*  We must intercept the frame window's messages                     */
/*  (to capture any input from the container control).                */
/*  We save the return value (the current WndProc),                   */
/*  so we can pass it all the other messages the frame gets.          */
/**********************************************************************/
   SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

   WinShowWindow(hFrameWnd, TRUE);
```

Figure 12-15 (Part 3 of 9). Sample Code for a Target Application

```
/**************************************************************************/
/*  Standard PM message loop - get it, dispatch it.                      */
/**************************************************************************/
   while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
       WinDispatchMsg(hab, &qmsg);

/**************************************************************************/
/*  Clean up on the way out.                                              */
/**************************************************************************/
   WinDestroyMsgQueue(hmq);
   WinTerminate(hab);

   return TRUE;
}

/**************************************************************************/
/*  LocalWndProc() - window procedure for the frame window.              */
/*  Called by PM whenever a message is sent to the frame.                */
/**************************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
   char         szDir[CCHMAXPATH];
   char         szBuffer[256];
   PDRAGINFO    dragInfo;
   PDRAGITEM    dragItem;
   USHORT       usOp;
   USHORT       usIndicator, cItems, i;
   ULONG        ulBytes;

   switch(msg)
   {
```
Figure 12-15 (Part 4 of 9). Sample Code for a Target Application

```
/***********************************************************************/
/*  Someone's dragging an object over us.                              */
/***********************************************************************/
     case DM_DRAGOVER:
     dragInfo = (PDRAGINFO)mp1;

     /* Get access to the DRAGINFO data structure */
     DrgAccessDraginfo(dragInfo);

     /* Can we accept this drop? */
     switch (dragInfo->usOperation)
     {

        /* Return DOR_NODROPOP if current operation */
        /* is link or unknown                       */
        case DO_UNKNOWN:
           DrgFreeDraginfo(dragInfo);
           return (MRFROM2SHORT (DOR_NODROPOP, 0));
           break;

        /* Our default operation is Move */
        case DO_DEFAULT:
           dragItem = DrgQueryDragitemPtr(dragInfo, 0);
            ulBytes = DrgQueryStrName(dragItem->hstrContainerName,
                                      sizeof(szDir),
                                      szDir);
           if (!ulBytes)
             return (MRFROM2SHORT (DOR_NODROPOP, 0));
           else
             usOp =  DO_MOVE;
           break;

        /* Do the requested specific operation */
        case DO_MOVE:
        case DO_COPY:
           usOp = dragInfo->usOperation;
           break;
     }
```

Figure 12-15 (Part 5 of 9). Sample Code for a Target Application

```
usIndicator = DOR_DROP;
cItems = DrgQueryDragitemCount(dragInfo);

/* Now, we need to look at each item in turn */
for (i = 0; i < cItems; i++)

{
   dragItem = DrgQueryDragitemPtr(dragInfo, i);

   /* Make sure we can move for a Move request */
   /* or copy for a Copy                        */
   if (((dragItem->fsSupportedOps & DO_COPYABLE)   &&
        (usOp == (USHORT)DO_COPY))                  ||
       ((dragItem->fsSupportedOps & DO_MOVEABLE)   &&
        (usOp == (USHORT)DO_MOVE)))

   {
      /* Check the rendering format */
      if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
         usIndicator = DOR_DROP;
      else
         usIndicator = DOR_NEVERDROP;
   }
   else
      usIndicator = DOR_NODROPOP;
}

/* Release the draginfo data structure */
DrgFreeDraginfo(dragInfo);

return (MRFROM2SHORT(usIndicator, usOp));
break;

/* Dragged object just left */
case DM_DRAGLEAVE:
   return (MRESULT)FALSE;
   break;

/* Drop the object on us (receive the object) */
case DM_DROP:

/* Get access to the DRAGINFO data structure */
DrgAccessDraginfo(dragInfo);
```

Figure 12-15 (Part 6 of 9). Sample Code for a Target Application

```
/* Can we accept this drop? */
switch (dragInfo->usOperation)

{
   /* Return DOR_NODROPOP if current operation */
   /* is link or unknown                        */
   case DO_UNKNOWN:
      DrgFreeDraginfo(dragInfo);
      return (MRFROM2SHORT (DOR_NODROPOP, 0));
      break;

   /* Our default operation is Move */
   case DO_DEFAULT:
      dragItem = DrgQueryDragitemPtr(dragInfo, 0);
      ulBytes  = DrgQueryStrName(dragItem->hstrContainerName,
                                 sizeof(szDir),
                                 szDir);

      if (!ulBytes)
        return (MRFROM2SHORT (DOR_NODROPOP, 0));
      usOp = (USHORT)DO_MOVE;
      break;

   /* Do the requested specific operation */
   case DO_MOVE:
   case DO_COPY:
      usOp = dragInfo->usOperation;
      break;
}

usIndicator = DOR_DROP;
cItems = DrgQueryDragitemCount(dragInfo);
```

Figure 12-15 (Part 7 of 9). Sample Code for a Target Application

```
    /* Now, we need to look at each item in turn */
    for (i = 0; i < cItems; i++)
    {
       dragItem = DrgQueryDragitemPtr(dragInfo, i);

       /* Make sure we can move for a Move request */
       /* or copy for a Copy                      */
       if (((dragItem->fsSupportedOps & DO_COPYABLE)   &&
           (usOp == (USHORT)DO_COPY))                  ||
          ((dragItem->fsSupportedOps & DO_MOVEABLE)    &&
           (usOp == (USHORT)DO_MOVE)))

       {
          /* Check the rendering format */
          if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
             usIndicator = DOR_DROP;
          else
             usIndicator = DOR_NEVERDROP;
       }
       else
          usIndicator = DOR_NODROPOP;

/************************************************************************/
/*  This is where we would actually move or copy the file,             */
/*  but we just display the name instead.                              */
/************************************************************************/
       DrgQueryStrName(dragItem->hstrSourceName, 255, szBuffer);
       WinMessageBox(HWND_DESKTOP,
                     HWND_DESKTOP,
                     szBuffer,
                     "Dropped",
                     0,
                     MB_OK);
    }

    /* Release the draginfo data structure */
    DrgFreeDraginfo(dragInfo);

    return (MRFROM2SHORT(usIndicator, usOp));
    break;
```

Figure 12-15 (Part 8 of 9). Sample Code for a Target Application

```
     /* Send the message to the usual WC_FRAME WndProc */
     default:
        return (*SysWndProc)(hwnd, msg, mp1, mp2);
        break;
   }
   return (*SysWndProc)(hwnd, msg, mp1, mp2);
}

================
TARGET.RC
================
#include <os2.h>
#include "target.h"

ICON TRASHCAN    trashcan.ico

================
TARGET.H
================
#define TRASHCAN 100

================
TARGET.DEF
================
NAME    TARGET   WINDOWAPI

DESCRIPTION 'PM Drag and Drop Sample'

CODE      MOVEABLE
DATA      MOVEABLE MULTIPLE

STACKSIZE 24576
HEAPSIZE  10240

PROTMODE

================
TARGET.LNK
================
target.obj
target.exe
target.map
target.def
```

Figure 12-15 (Part 9 of 9). Sample Code for a Target Application

Related Functions

This section covers the functions that are related to direct manipulation.

DrgAcceptDroppedFiles

This function handles the file direct manipulation protocol for a given window.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

BOOL DrgAcceptDroppedFiles  (HWND Hwnd, PSZ pPath, PSZ pTypes,
                             ULONG ulDefaultOp, ULONG ulReserved)
```

Parameters

Hwnd (HWND) – input
Handle of calling window.

pPath (PSZ) – input
Directory in which to place the dropped files.

pTypes (PSZ) – input
List of types that are acceptable to the drop.

ulDefaultOp (ULONG) – input
Default drag operation for this window.

ulReserved (ULONG) – input
Reserved.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgAccessDraginfo

This function accesses a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

BOOL DrgAccessDraginfo  (PDRAGINFO  pDraginfo)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgAddStrHandle

This function creates a handle to a string.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

HSTR DrgAddStrHandle  (PSZ  pString)
```

Parameters

pString (PSZ) – input
String for which a handle is to be created.

Returns

hstr (HSTR) – returns
String handle.

NULLHANDLE	Error occurred.
Other	String handle created.

DrgAllocDraginfo

This function allocates a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

PDRAGINFO DrgAllocDraginfo  (ULONG cDitem)
```

Parameters

cDitem (ULONG) – input
Number of objects being dragged.

Returns

Draginfo (PDRAGINFO) – returns
Pointer to the DRAGINFO structure.

NULL	Error occurred.
Other	The DRAGINFO structure.

DrgAllocDragtransfer

This function allocates a specified number of DRAGTRANSFER structures from a single segment.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGTRANSFER DrgAllocDragtransfer (ULONG cdxfer)
```

Parameters

cdxfer (ULONG) – input
Number of DRAGTRANSFER structures to be allocated.

Returns

pDragtransfer (PDRAGTRANSFER) – returns
Pointer to an array of DRAGTRANSFER structures.

NULL	Error occurred.
Other	The array of DRAGTRANSFER structures.

DrgCancelLazyDrag

This function is called to cancel the current drag operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgCancelLazyDrag ()
```

Parameters

None.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Lazy drag is successfully canceled.
FALSE	An error occurred.

DrgDeleteDraginfoStrHandles

This function deletes each unique string handle in a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgDeleteDraginfoStrHandles  (PDRAGINFO  pDraginfo)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure that contains string handles to delete.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgDeleteStrHandle

This function deletes a string handle.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgDeleteStrHandle (HSTR Hstr)
```

Parameters

Hstr (HSTR) – input
The string handle to delete.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgDrag

This function performs a drag operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
HWND DrgDrag (HWND hwndSource, PDRAGINFO pDraginfo,
              PDRAGIMAGE pdimg, ULONG cdimg, LONG vkTerminate,
              PVOID pReserved)
```

Parameters

hwndSource (HWND) – input
Handle of the source window calling this function.

pDraginfo (PDRAGINFO) – in/out
Pointer to the DRAGINFO structure.

pdimg (PDRAGIMAGE) – input
Pointer to an array of DRAGIMAGE structures.

cdimg (ULONG) – input
Number of DRAGIMAGE structures in the *pdimg* array. Must be > 0.

vkTerminate (LONG) – input
Pointing device button that ends the drag operation.

VK_BUTTON1	Release of button 1 ends the drag.
VK_BUTTON2	Release of button 2 ends the drag.
VK_BUTTON3	Release of button 3 ends the drag.
VK_ENDDRAG	Release of the system-defined direct manipulation button ends the drag. This is the recommended value if the DrgDrag function call is invoked in response to a WM_BEGINDRAG message.

pReserved (PVOID) – input
Reserved value, must be NULL.

Returns

hwndDest (HWND) – returns
Handle of window on which the dragged objects were dropped.

DrgDragFiles

This function begins a direct manipulation operation for one or more files.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

BOOL DrgDragFiles  (HWND Hwnd, PAPSZ pFiles, PAPSZ pTypes,
                    PAPSZ pTargets, ULONG cFiles, HPOINTER hptrDrag,
                    ULONG vkTerm, BOOL fSourceRender, ULONG ulReserved)
```

Parameters

Hwnd (HWND) – input
Handle of calling window.

pFiles (PAPSZ) – input
The names of the files to be dragged.

pTypes (PAPSZ) – input
The file types of the files to be dragged.

pTargets (PAPSZ) – input
Target file names.

cFiles (ULONG) – input
Number of files to be dragged.

hptrDrag (HPOINTER) – input
Icon to display during the drag.

vkTerm (ULONG) – input
Button that ends the drag.

VK_BUTTON1	Release of button 1 ends the drag.
VK_BUTTON2	Release of button 2 ends the drag.
VK_BUTTON3	Release of button 3 ends the drag.
VK_ENDDRAG	Release of the system-defined direct manipulation button ends the drag. This is the recommended value if the DrgDrag function call is invoked in response to a WM_BEGINDRAG message.

fSourceRender (BOOL) – input
Flag indicating whether the source must perform the move or copy.

TRUE	The caller will receive a DM_RENDERFILE message for each file.
FALSE	All file manipulation is performed by DrgDragFiles.

ulReserved (ULONG) – input
Reserved.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	The drag operation was initiated successfully.
FALSE	An error occurred.

DrgFreeDraginfo

This function frees a DRAGINFO structure allocated by DrgAllocDraginfo.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgFreeDraginfo  (PDRAGINFO  pDraginfo)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgFreeDragtransfer

This function frees the storage associated with a DRAGTRANSFER structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgFreeDragtransfer  (PDRAGTRANSFER  pdxfer)
```

Parameters

pdxfer (PDRAGTRANSFER) – input
Pointer to the DRAGTRANSFER structures to be freed.

Returns

rc (BOOL) – returns
Return code.

TRUE	The structure was freed successfully.
FALSE	The deallocation failed.

DrgGetPS

This function gets a presentation space that is used to provide target feedback to the user during a drag operation.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

HPS DrgGetPS  (HWND  Hwnd)
```

Parameters

Hwnd (HWND) – input
Handle of the window for which presentation space is required.

Returns

Hps (HPS) – returns
Presentation-space handle used for drawing in the window.

NULLHANDLE	Error occurred.

DrgLazyDrag

This function is called when a direct-manipulation button is pressed wile the lazy drag augmentation key is held to initiate a pickup and drop (lazy drag) operation.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

BOOL DrgLazyDrag  (HWND  hwndSource, PDRAGINFO  pDraginfo,
                   PDRAGIMAGE  pdimg, ULONG  cdimg, PVOID  Reserved)
```

Parameters

hwndSource (HWND) – input
Handle of the source window that is calling this function.

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure which contains information about the objects being dragged.

pdimg (PDRAGIMAGE) – input
Pointer to an array of DRAGIMAGE structures.

cdimg (ULONG) – input
Number of DRAGIMAGE structures in the *pdimg* array.

Reserved (PVOID) – input
Reserved value, must be 0.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	A lazy drag operation was successfully started.
FALSE	An error occurred while initiating a lazy drag operation.

DrgLazyDrop

This function is called to invoke a lazy drop operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>

BOOL DrgLazyDrop (HWND hwndTarget, ULONG ulsOperation,
                  PPOINTL pptlDrop)
```

Parameters

hwndTarget (HWND) – input
Handle of the target window receiving the drop.

ulsOperation (ULONG) – input
Drop operation code.

DO_DEFAULT	Default operation.
DO_COPY	Operation is a copy.
DO_MOVE	Operation is a move.
DO_LINK	Operation is a link.

pptlDrop (PPOINTL) – input
Pointer to the drop location in desktop coordinates.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Objects are successfully dropped.
FALSE	An error occurred.

DrgPostTransferMsg

This function posts a message to the other application involved in the direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

BOOL DrgPostTransferMsg  (HWND hwndTo, ULONG ulMsgid,
                          PDRAGTRANSFER pdxfer, ULONG fs,
                          ULONG ulReserved, BOOL fRetry)
```

Parameters

hwndTo (HWND) – input
Window handle to which the message is to be posted.

Target	*hwndItem* in the DRAGITEM structure.
Source	*hwndClient* in the DRAGTRANSFER structure.

ulMsgid (ULONG) – input
Identifier of the message to be posted.

pdxfer (PDRAGTRANSFER) – input
Pointer to the DRAGTRANSFER structure.

fs (ULONG) – input
Flags to be passed in the *param2* parameter of the message identified by *ulMsgid*.

ulReserved (ULONG) – input
Reserved value, must be 0.

fRetry (BOOL) – input
Retry indicator.

TRUE
: If the destination queue is full, the message posting is retried at 1-second intervals until the message is posted successfully.

  In this case, DrgPostTransferMsg dispatches any messages in the queue by calling WinPeekMsg and WinDispatchMsg in a loop. The application can receive messages sent by other applications while it is trying to post drag transfer messages.

FALSE
: The call returns FALSE without retrying.

Returns

rc (BOOL) – returns
Success indicator.

TRUE
: Successful completion.

FALSE
: Error occurred.

DrgPushDraginfo

This function gives a process access to a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

BOOL DrgPushDraginfo  (PDRAGINFO  pDraginfo, HWND  hwndDest)
```

Parameters
pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure.

hwndDest (HWND) – input
Handle of the window whose process is to be given access to a DRAGINFO structure.

Returns
rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgQueryDraginfoPtrFromDragitem
This function is called to obtain a pointer to the DRAGINFO structure associated with a given DRAGITEM structure.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

PDRAGINFO DrgQueryDraginfoPtrFromDragitem  (PDRAGITEM  pDragitem)
```

Parameters
pDragitem (PDRAGITEM) – input
Pointer to a DRAGITEM structure whose corresponding DRAGINFO is to be returned.

Returns
pDraginfo (PDRAGINFO) – returns
Pointer to the DRAGINFO structure for the specified *pDragitem*.

DrgQueryDraginfoPtrFromHwnd

This function determines whether a particular window has allocated a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGINFO DrgQueryDraginfoPtrFromHwnd (HWND hwndSource)
```

Parameters

hwndSource (HWND) – input
  Handle of the window whose associated DRAGINFO pointer is to be returned.

Returns

pDraginfo (PDRAGINFO) – returns
  Pointer to the DRAGINFO structure allocated by the window specified by *hwndSource*.

DrgQueryDragitem

This function returns a DRAGITEM structure used in the direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgQueryDragitem (PDRAGINFO pDraginfo, ULONG cbBuffer,
                       PDRAGITEM pDragitem, ULONG iItem)
```

Parameters

pDraginfo (PDRAGINFO) – input
  Pointer to the DRAGINFO structure from which the DRAGITEM structure is obtained.

cbBuffer (ULONG) – input
  Maximum number of bytes to copy to the buffer.

pDragitem (PDRAGITEM) – output
Pointer to the buffer into which the DRAGITEM structure is copied.

iItem (ULONG) – input
Zero-based index of the DRAGITEM to be returned.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgQueryDragitemCount

This function returns the number of objects being dragged during the current direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

ULONG DrgQueryDragitemCount  (PDRAGINFO pDraginfo)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure for which number of dragged objects is requested.

Returns

cDitem (ULONG) – returns
Number of objects being dragged.

DrgQueryDragitemPtr

This function returns a pointer to the DRAGITEM structure used in the direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGITEM DrgQueryDragitemPtr (PDRAGINFO pDraginfo, ULONG ulIndex)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure from which the DRAGITEM structure is obtained.

ulIndex (ULONG) – input
Zero-based index of the DRAGITEM structure for which the pointer is to be returned.

Returns

Dragitem (PDRAGITEM) – returns
Pointer to the DRAGITEM structure.

DrgQueryDragStatus

This function determines the status of the current drag operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryDragStatus ()
```

Parameters

None.

Returns

rc (ULONG) – returns
  Flag indicating the current drag status.

0	A drag operation is not currently in progress.
DGS_DRAGINPROGRESS	A standard drag operation is in progress.
DGS_LAZYDRAGINPROGRESS	A lazy drag operation is in progress.

DrgQueryNativeRMF

This function obtains the ordered pair that represents the native rendering mechanism and format of the dragged object.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

BOOL DrgQueryNativeRMF  (PDRAGITEM pDragitem, ULONG cbBuffer,
                         PCHAR ppBuffer)
```

Parameters

pDragitem (PDRAGITEM) – input
  Pointer to the DRAGITEM structure.

cbBuffer (ULONG) – input
  Maximum number of bytes to copy to the buffer.

ppBuffer (PCHAR) – output
  Pointer to the buffer in which the null-terminated string is to be returned.

Returns

rc (BOOL) – returns
  Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgQueryNativeRMFLen

This function obtains the length of the string representing the native rendering mechanism and format of the dragged object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryNativeRMFLen  (PDRAGITEM pDragitem)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose native rendering mechanism and format string length are to be obtained.

Returns

ulLength (ULONG) – returns
String length of the ordered pair.

0	Error occurred.
Other	String length of the ordered pair, excluding the null-terminating byte.

DrgQueryStrName

This function gets the contents of a string associated with a string handle.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryStrName  (HSTR Hstr, ULONG cbBuflen, PSZ pBuffer)
```

Parameters

Hstr (HSTR) – input
The handle must have been created with DrgAddStrHandle.

cbBuflen (ULONG) – input
Maximum number of bytes to copy into *pBuffer*.

pBuffer (PSZ) – output
Buffer where the null-terminated string is returned.

Returns

ulLength (ULONG) – returns
Number of bytes written to *pBuffer*.

DrgQueryStrNameLen

This function gets the length of a string associated with a string handle.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryStrNameLen  (HSTR Hstr)
```

Parameters

Hstr (HSTR) – input
String handle.

Returns

cLength (ULONG) – returns
Length of the string associated with *Hstr*.

0	The string handle is NULLHANDLE or is not valid.
Other	The length of the string associated with the string handle, excluding the null terminating byte.

DrgQueryTrueType

This function obtains the true type of a dragged object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgQueryTrueType (PDRAGITEM pDragitem, ULONG cbBuflen,
                       PSZ pBuffer)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose type is to be obtained.

cbBuflen (ULONG) – input
Maximum number of bytes to copy to *pBuffer*. Must be > 0.

pBuffer (PSZ) – output
Buffer in which the null-terminated string is to be returned.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgQueryTrueTypeLen

This function obtains the length of the string that represents the true type of a dragged object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryTrueTypeLen (PDRAGITEM pDragitem)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose type length is to be obtained.

Returns

ulLength (ULONG) – returns
String length of the first element of the character string associated with *hstrType*.

0	Error occurred.
Other	The length of the first element of the character string associated with *hstrType*, excluding the null-terminating byte.

DrgReallocDragInfo

This function releases the current DRAGINFO structure and reallocates a new one.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

PDRAGINFO DrgReallocDragInfo  (PDRAGINFO pdinfoOld, ULONG cditem)
```

Parameters

pdinfoOld (PDRAGINFO) – input
Pointer to the current DRAGINFO structure.

cditem (ULONG) – input
Number of DRAGITEM structures to be allocated.

Returns

pdinfoCurrent (PDRAGINFO) – returns
Pointer to a newly allocated DRAGINFO structure.

DrgReleasePS

This function releases a presentation space obtained by using the DrgGetPS function.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgReleasePS (HPS Hps)
```

Parameters

Hps (HPS) – input
Handle of the presentation space to release.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgSendTransferMsg

This function sends a message to the other application involved in the direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
MRESULT DrgSendTransferMsg (HWND hwndTo, ULONG ulMsgid,
                            MPARAM mpParam1, MPARAM mpParam2)
```

Parameters

hwndTo (HWND) – input
Window handle to which the message is to be sent.

Target	*hwndItem* in the DRAGITEM structure.
Source	*hwndClient* in the DRAGTRANSFER structure.

ulMsgid (ULONG) – input
Identifier of the message to be sent.

mpParam1 (MPARAM) – input
First message parameter.

mpParam2 (MPARAM) – input
Second message parameter.

Returns

mresReply (MRESULT) – returns
Message-return data.

DrgSetDragImage

This function sets the image that is being dragged.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>

BOOL DrgSetDragImage  (PDRAGINFO pDraginfo, PDRAGIMAGE pdimg,
                       ULONG cdimg, PVOID pReserved)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure. representing the drag operation for which the pointer is to be set.

pdimg (PDRAGIMAGE) – input
Pointer to an array of DRAGIMAGE structures.

cdimg (ULONG) – input

Number of DRAGIMAGE structures in the *pdimg* array.

pReserved (PVOID) – input
Reserved value, must be NULL.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgSetDragitem

This function sets the values in a DRAGITEM structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgSetDragitem  (PDRAGINFO pDraginfo, PDRAGITEM pDragitem,
                      ULONG cbBuffer, ULONG iItem)
```

Parameters

pDraginfo (PDRAGINFO) – input
    Pointer to the DRAGINFO structure in which to place the DRAGITEM.

pDragitem (PDRAGITEM) – input
    Pointer to the DRAGITEM structure to place in DRAGINFO.

cbBuffer (ULONG) – input
    Size of the DRAGITEM addressed by *pDragitem*.

iItem (ULONG) – input
    Zero-based index of the DRAGITEM to be set.

Returns

rc (BOOL) – returns
    Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgSetDragPointer

This function sets the pointer to be used while over the current target.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgSetDragPointer  (PDRAGINFO pDraginfo, HPOINTER hptrHandle)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure. to be used for this drag.

hptrHandle (HPOINTER) – input
Handle to the pointer to use.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyNativeRMF

This function determines if the native rendering mechanism and format of an object match any supplied by the application.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgVerifyNativeRMF  (PDRAGITEM  pDragitem, PSZ  pRMF)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure. whose native rendering mechanism and format are to be verified.

pRMF (PSZ) – input
A String specifying the rendering mechanism and format.

Returns

rc (BOOL) – returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyRMF

This function determines if a given rendering mechanism and format are supported for a dragged object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgVerifyRMF (PDRAGITEM pDragitem, PSZ pMech, PSZ pFormat)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose native rendering mechanism and format are to be validated.

pMech (PSZ) – input
String specifying the rendering mechanism to search for.

pFormat (PSZ) – input
String specifying the rendering format to search for.

Returns

rc (BOOL) – returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyTrueType

This function determines if the true type of a dragged object matches an application-supplied type string.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgVerifyTrueType (PDRAGITEM pDragitem, PSZ pType)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose true type is to be verified.

pType (PSZ) – input
String specifying a type.

Returns

rc (BOOL) – returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyType

This function verifies whether a given type is present in the list of types defined for a drag object.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

BOOL DrgVerifyType  (PDRAGITEM  pDragitem, PSZ  pType)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose *hstrType* is to be verified.

pType (PSZ) – input
String specifying the types to search for.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyTypeSet

This function returns the intersection of the contents of the string associated with the type-string handle for an object and an application-specified type string.

Syntax

```
#define INCL_WINSTDDRAG

#include <os2.h>

BOOL DrgVerifyTypeSet (PDRAGITEM pDragitem, PSZ pType, ULONG cbBuflen,
                       PSZ pBuffer)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose *hstrType* is to be verified.

pType (PSZ) – input
String specifying the types to search for.

cbBuflen (ULONG) – input
Size of the return buffer.

pBuffer (PSZ) – output
Buffer where the intersection string is returned.

Returns

rc (BOOL) – returns
Match indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Related Window Messages

This section covers the window messages that are related to direct manipulation.

DM_DISCARDOBJECT

This message is sent to a source that supports the "DRM_DISCARD" rendering method.

Parameters

param1

pDragInfo (PDRAGINFO)
Pointer to the DRAGINFO structure representing the items to be discarded.

mpparam2

ulReserved (MPARAM)
Reserved value, should be NULL.

Returns

ulAction (ULONG)
Flag.

DRR_SOURCE	The source window procedure accepts responsibility for the operation.
DRR_TARGET	The target window procedure is to accept responsibility for the operation. The OS/2 shell supports the discarding of dragitems that can be rendered by the DRM_OS2FILE method.
DRR_ABORT	Abort the entire DM_DROP action.

DM_DRAGERROR

This message is sent to the caller of DrgDragFiles or DrgAcceptDroppedFiles when an error occurs during a move or copy operation for a file.

Parameters

param1

usError (USHORT)
Error code.

Returned from DosCopy, DosMove, or DosDelete.

usOperation (USHORT)
Flag.

Flag indicating the operation that failed.

DFF_MOVE	DosMove failed.
DFF_COPY	DosCopy failed.
DFF_DELETE	DosDelete failed.

param2

hstr (HSTR)
HSTR of file contributing to the error.

Returns

hstrAction (HSTR)
Action indicator.

DME_IGNORECONTINUE	Do not retry the operation, but continue with the rest of the files.
DME_IGNOREABORT	Do not retry the operation, and do not try any other files.
DME_RETRY	Retry the operation.
DME_REPLACE	Replace the file at the destination. Used if FALSE is not specified.
Other	HSTR of new file name to use for retry.

DM_DRAGFILECOMPLETE

This message is sent when a direct manipulation operation on a file or files is complete.

Parameters

param1

hstr (HSTR)
File handle.

param2

usOperation (USHORT)
Flags.

DF_MOVE	The operation was a move. If this flag is not set, the operation was a copy.
DF_SOURCE	The receiving window was the source of the drag. If this flag is not set, the receiver was the target of the drop.

DF_SUCCESSFUL	The drag operation was successful for the file. If this flag is not set, the operation failed.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

DM_DRAGLEAVE

This message is sent to a window that is being dragged over when one of these conditions occur:

- The object is dragged outside the boundaries of the window.
- The drag operation is terminated while the object is over the window.

Parameters

param1

pDraginfo (PDRAGINFO)
Pointer to the DRAGINFO structure for the drag operation.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

DM_DRAGOVER

This message allows the window under the mouse pointer to determine if the object or objects currently being dragged can be dropped.

param2 is the pointing device pointer location.

Parameters

param1

pDraginfo (PDRAGINFO)
Pointer to the DRAGINFO structure representing the object being dragged.

param2

sxDrop (SHORT)
X-coordinate of the pointing device pointer in desktop coordinates.

syDrop (SHORT)
Y-coordinate of the pointing device pointer in desktop coordinates.

Returns

ReturnCode

usDrop (USHORT)
Drop indicator.

DOR_DROP	Object can be dropped. When this reply is given, *usDefaultOp* must be set to indicate which operation is performed if the user should drop at this location.
DOR_NODROP	Object cannot be dropped at this time. The target can accept the object in the specified type and format using the specified operation, but the current state of the target will not allow it to be dropped on. The target may change state in the future so that the same object may be acceptable.
DOR_NODROPOP	Object cannot be dropped at this time. The target can accept the object in the specified type and format, but the current operation is not acceptable. A change in the drag operation may change the acceptability of the object.
DOR_NEVERDROP	Object cannot be dropped. The target cannot accept the object now and will not change state so that the object will be acceptable in the future. If this response is returned, no more DM_DRAGOVER messages will be sent to the target until the pointer is moved out of and back into the target window.

usDefaultOp (USHORT)
Target-defined default operation.

DO_COPY	Operation is a copy.
DO_LINK	Operation is a link.
DO_MOVE	Operation is a move.
Other	Operation is defined by the application. This value should be greater than or equal to (>=) DO_UNKNOWN.

DM_DRAGOVERNOTIFY

This message is sent to the source of a drag operation immediately after a DM_DRAGOVER message is sent to a target window.

param2 is the target's reply to the DM_DRAGOVER message.

Parameters

param1

pDraginfo (PDRAGINFO)
Pointer to the DRAGINFO structure that represents the object being dragged.

param2
Target's reply.

usDrop (USHORT)
Drop indicator.

usDefaultOp (USHORT)
Default operation.

Target-defined default operation.

Returns

ulReserved (ULONG)
Reserved value.

DM_DROP

This message is sent to the target when the dragged object is dropped.

Parameters

param1

pDraginfo (PDRAGINFO)
Pointer to the DRAGINFO structure.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

DM_DROPHELP

This message requests help for the current drag operation.

Parameters

param1

pDraginfo (PDRAGINFO)
Pointer to the DRAGINFO structure used in the drag operation.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

DM_DROPNOTIFY

This message provides the source window with the target window handle and a pointer to the DRAGINFO structure allocated by the source window.

Parameters

param1

pDraginfo (PDRAGINFO)
Pointer to the DRAGINFO structure allocated by the source window receiving the message.

param2

hwndTarget (HWND)
Handle of the target window that the drag set was dropped on.

Note: If *hwndTarget* is equal to zero, the drag is canceled, and the drag set is not dropped. DrgCancelLazyDrag posts a DM_DROPNOTIFY message with an *hwndTarget* value of zero to the source window.

Returns

returns

ulReserved (ULONG)
Reserved value, must be 0.

DM_EMPHASIZETARGET

This message is sent to the caller of DrgAcceptDroppedFiles to inform it to either apply or remove target emphasis from itself.

Parameters

param1

sx (SHORT)
X-coordinate.

X-coordinate of the pointing device pointer in window coordinates.

sy (SHORT)
Y-coordinate.

Y-coordinate of the pointing device pointer in window coordinates.

usparam2

usEmphasis (USHORT)
Flags.

TRUE	Apply emphasis.
FALSE	Remove emphasis.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

DM_ENDCONVERSATION

The target uses this message to notify a source that a drag operation is complete.

Parameters

param1

ulItemID (ULONG)
Item ID.

The *ulItemID* from the DRAGITEM that was contained within the DRAGINFO structure when the object was dropped.

param2

ulFlags (ULONG)
Flags.

The flags are set as follows:

DMFL_TARGETSUCCESSFUL	The target successfully completed its portion of the rendering operation.
DMFL_TARGETFAIL	The target failed to complete its portion of the rendering operation.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

DM_FILERENDERED

This message is sent to the window handling the drag conversation for the caller of DrgDragFiles.

Parameters

param1

rndf (PRENDERFILE)
Pointer to a RENDERFILE structure.

param2

usOperation (USHORT)
Flags.

TRUE	Operation succeeded
FALSE	Operation failed.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

DM_PRINTOBJECT

This message is sent to a source that supports the "DRM_PRINT" rendering method when objects are dropped on a printer object.

Parameters

param1

pDragInfo (PDRAGINFO)
Pointer to the DRAGINFO structure representing the objects to be printed.

param2

pPrintDest (PPRINTDEST)
Pointer to the PRINTDEST structure representing printer object to print to.

The structure contains all the parameters required to call the functions DevPostDeviceModes and DevOpenDC.

Returns

ulAction (ULONG)
Flag.

DRR_SOURCE	The source window procedure/object procedure will take responsibility for the print operation.
DRR_TARGET	The target printer object will take responsibility for the print operation (this will only work on objects which are of the pre-registered rendering method; "DRM_OS2FILE."
DRR_ABORT	Abort the entire DM_DROP action (do not send any more DM_PRINTOBJECT messages to any selected source object involved in this DM_DROP.

DM_RENDER

This message is used to request a source to provide a rendering of an object in a specified rendering mechanism and format.

Parameters

param1

pDxfer (PDRAGTRANSFER)
Pointer to the DRAGTRANSFER structure.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DM_RENDERCOMPLETE

This message is posted by a source to a target window. It informs the target that the source has completed a requested rendering operation.

Parameters

param1

pDxfer (PDRAGTRANSFER)
Pointer to the DRAGTRANSFER structure.

param2

usFS (USHORT)
Flag field.

Flag field indicating successful completion.

DMFL_RENDERFAIL	The source is unable to perform the rendering operation. The target may be allowed to retry. If the target is allowed to retry and chooses not to, it must send a DM_ENDCONVERSATION message to the source.

DMFL_RENDEROK	The source has completed the rendering operation. When the target completes its part of the rendering operation, it must post a DM_RENDERCOMPLETE message to the source.
DMFL_RENDERRETRY	The source has completed the rendering operation and will allow the target to retry its part of the operation if it fails. This flag can be set in conjunction with either the DMFL_RENDERFAIL or DMFL_RENDEROK flags.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

DM_RENDERFILE

This message is sent to the caller of DrgDragFiles to tell it to render a file.

Parameters

param1

rndf (PRENDERFILE)
Pointer to a RENDERFILE structure.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Render handling.

TRUE	The receiver handled the rendering.
FALSE	DrgDragFiles should render this file.

DM_RENDERPREPARE

This message tells a source to prepare for the rendering of an object.

Parameters

param1

pDxfer (PDRAGTRANSFER)
Pointer to a DRAGTRANSFER structure.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

rc (BOOL)
Success indicator.

TRUE — The message was processed by the recipient and it is ready to perform the rendering operation. The target of the drop sends a DM_RENDER message to request the rendering with a specific rendering mechanism and format.

FALSE — The message either was not processed by the recipient, or it is unprepared to perform the rendering. The *hwndItem* field in DRAGITEM may not be properly initialized, and therefore the target should not send a DM_ENDCONVERSATION message.

Related Data Structures

This section covers the data structures that are related to direct manipulation.

DRAGIMAGE

Dragged-object-image structure which describes the images that are to be drawn under the direct-manipulation pointer for the duration of a drag operation.

Syntax

```
typedef struct _DRAGIMAGE {
USHORT       cb;
USHORT       cptl;
LHANDLE      hImage;
SIZEL        sizlStretch;
ULONG        fl;
SHORT        cxOffset;
SHORT        cyOffset;
 } DRAGIMAGE;

typedef DRAGIMAGE *PDRAGIMAGE;
```

Fields

cb (USHORT)
Size, in bytes, of the DRAGIMAGE structure.

cptl (USHORT)
The number of points in the point array if *fl* is specified as DRG_POLYGON.

hImage (LHANDLE)
Handle representing the image to display.

The type is determined by *fl*.

sizlStretch (SIZEL)
Dimensions for stretching when *fl* is specified as DRG_STRETCH.

fl (ULONG)
Flags.

DRG_ICON	*hImage* is an HPOINTER.
DRG_BITMAP	*hImage* is an HBITMAP.
DRG_POLYGON	*hImage* is a pointer to an array of points that will be connected with GpiPolyLine to form a polygon. The first point of the array should be (0,0), and the other points should be placed relative to this position.
DRG_STRETCH	If DRG_ICON or DRG_BITMAP is specified, the image is expanded or compressed to the dimensions specified by *sizlStretch*.

DRG_TRANSPARENT	If DRG_ICON is specified, an outline of the icon is generated and displayed instead of the original icon.
DRG_CLOSED	If DRG_POLYGON is specified, a closed polygon is formed by moving the current position to the last point in the array before calling GpiPolyLine.

cxOffset (SHORT)
X-offset from the pointer hot spot to the origin of the image.

cyOffset (SHORT)
Y-offset from the pointer hot spot to the origin of the image.

DRAGINFO

Drag-information structure.

Syntax

```
typedef struct _DRAGINFO {
ULONG        cbDraginfo;
USHORT       cbDragitem;
USHORT       usOperation;
HWND         hwndSource;
SHORT        xDrop;
SHORT        yDrop;
USHORT       cditem;
USHORT       usReserved;
 } DRAGINFO;

typedef DRAGINFO *PDRAGINFO;
```

Fields

cbDraginfo (ULONG)
Structure size, in bytes.

The size includes the array of DRAGITEM structures.

cbDragitem (USHORT)
Size, in bytes, of each DRAGITEM structure.

usOperation (USHORT)
Modified drag operations.

An application can define its own modified drag operations for use when simulating a drop. These operations must have a value greater than DO_UNKNOWN. Possible values are described in the following list:

DO_DEFAULT	Execute the default drag operation. No modifier keys are pressed.
DO_COPY	Execute a copy operation. The Ctrl key is pressed.
DO_LINK	Execute a link operation. The Ctrl+Shift keys are pressed.

DO_MOVE Execute a move operation. The Shift key is pressed.
DO_UNKNOWN An undefined combination of modifier keys is pressed.

hwndSource (HWND)
Window handle of the source of the drag operation.

xDrop (SHORT)
X-coordinate of drop point expressed in desktop coordinates.

yDrop (SHORT)
Y-coordinate of drop point expressed in desktop coordinates.

cditem (USHORT)
Count of DRAGITEM structures.

usReserved (USHORT)
Reserved.

DRAGITEM

Drag-object structure.

Syntax

```
typedef struct _DRAGITEM {
HWND          hwndItem;
ULONG         ulItemID;
HSTR          hstrType;
HSTR          hstrRMF;
HSTR          hstrContainerName;
HSTR          hstrSourceName;
HSTR          hstrTargetName;
SHORT         cxOffset;
SHORT         cyOffset;
USHORT        fsControl;
USHORT        fsSupportedOps;
  } DRAGITEM;

typedef DRAGITEM *PDRAGITEM;
```

Fields

hwndItem (HWND)
Window handle of the source of the drag operation.

ulItemID (ULONG)
Information used by the source to identify the object being dragged.

hstrType (HSTR)
String handle of the object type.

The string handle must be created using the DrgAddStrHandle function. The string is of the form:

```
type[,type...]
```

The first type in the list must be the true type of the object. The following types are used by the OS/2* shell:

DRT_ASM	Assembler code
DRT_BASIC	BASIC code
DRT_BINDATA	Binary data
DRT_BITMAP	Bit map
DRT_C	C code
DRT_COBOL	COBOL code
DRT_DLL	Dynamic link library
DRT_DOSCMD	DOS command file
DRT_EXE	Executable file
DRT_FONT	Font
DRT_FORTRAN	FORTRAN code
DRT_ICON	Icon
DRT_LIB	Library
DRT_METAFILE	Metafile
DRT_OS2CMD	OS/2 command file
DRT_PASCAL	Pascal code
DRT_RESOURCE	Resource file
DRT_TEXT	Text
DRT_UNKNOWN	Unknown type.

hstrRMF (HSTR)
String handle of the rendering mechanism and format.

The string handle must be created using the DrgAddStrHandle function. The string is of the form:

```
mechfmt[,mechfmt...]
```

where `mechfmt` can be in either of the following formats:

- <mechanism(1),format(1)>
- (mechanism(1)[, mechanism(n)...]) x (format(1)[,format(n)...])

The first mechanism/format pair must be the native rendering mechanism and format of the object.

Valid mechanisms are:

"DRM_DDE"	Dynamic data exchange
"DRM_OBJECT"	Item being dragged is a workplace object.

"DRM_OS2FILE"	OS/2 file
"DRM_PRINT"	Object can be printed using direct manipulation.

Valid formats are:

"DRF_BITMAP"	OS/2 bit map
"DRF_DIB"	DIB
"DRF_DIF"	DIF
"DRF_DSPBITMAP"	Stream of bit-map bits
"DRF_METAFILE"	Metafile
"DRF_OEMTEXT"	OEM text
"DRF_OWNERDISPLAY"	Bit stream
"DRF_PTRPICT"	Printer picture
"DRF_RTF"	Rich text
"DRF_SYLK"	SYLK
"DRF_TEXT"	Null-terminated string
"DRF_TIFF"	TIFF
"DRF_UNKNOWN"	Unknown format.

hstrContainerName (HSTR)
String handle of the name of the container holding the source object.

The string handle must be created using the DrgAddStrHandle function.

hstrSourceName (HSTR)
String handle of the name of the source object.

The string handle must be created using the DrgAddStrHandle function.

hstrTargetName (HSTR)
String handle of the suggested name of the object at the target.

It is the responsibility of the source of the drag operation to create this string handle before calling DrgDrag.

cxOffset (SHORT)
X-offset from the pointer hot spot to the origin of the image that represents this object.

This value is copied from *cxOffset* in the DRAGIMAGE structure by DrgDrag.

cyOffset (SHORT)
Y-offset from the pointer hot spot to the origin of the image that represents this object.

This value is copied from *cyOffset* in the DRAGIMAGE structure by DrgDrag.

fsControl (USHORT)
Source-object control flags.

DC_OPEN	Object is open
DC_REF	Reference to another object
DC_GROUP	Group of objects
DC_CONTAINER	Container of other objects
DC_PREPARE	Source requires a DM_RENDERPREPARE message before it establishes a data transfer conversation

DC_REMOVEABLEMEDIA	Object is on removable media, or object cannot be recovered after a move operation.

fsSupportedOps (USHORT)
Direct manipulation operations supported by the source object.

DO_COPYABLE	Source supports DO_COPY
DO_LINKABLE	Source supports DO_LINK
DO_MOVEABLE	Source supports DO_MOVE.

DRAGTRANSFER

Drag-conversation structure.

Syntax

```
typedef struct _DRAGTRANSFER {
ULONG           cb;
HWND            hwndClient;
PDRAGITEM       pditem;
HSTR            hstrSelectedRMF;
HSTR            hstrRenderToName;
ULONG           ulTargetInfo;
USHORT          usOperation;
USHORT          fsReply;
 } DRAGTRANSFER;

typedef DRAGTRANSFER *PDRAGTRANSFER;
```

Fields

cb (ULONG)
Size, in bytes, of the structure.

hwndClient (HWND)
Handle of the client window.

This can be the target window or a window that represents an object in a container that was dropped on.

pditem (PDRAGITEM)
Pointer to the DRAGITEM structure that is to be rendered.

This structure must exist within the DRAGINFO structure that was passed in the DM_DROP message.

hstrSelectedRMF (HSTR)
String handle for the selected rendering mechanism and format for the transfer operation.

This handle must be created using DrgAddStrHandle. The target is responsible for deleting this handle when the conversation is complete. The string is in the format: <MECHANISM,FORMAT>.

hstrRenderToName (HSTR)
String handle representing the name where the source places, and the target finds, the data that is rendered.

The target is responsible for deleting this string handle when the conversation terminates. The contents of this field vary according to the rendering mechanism. See *hstrRMF* field in DRAGITEM.

OS/2 File	The string handle represents the fully qualified name of the file where the rendering will be placed.
DDE	This field is not used.
Print	This field is not used.

ulTargetInfo (ULONG)
Reserved.

Reserved for use by the target. The target can use this field for information about the object and rendering operation.

usOperation (USHORT)
The operation.

Values are:

DO_COPY	Execute a copy operation.
DO_LINK	Execute a link operation.
DO_MOVE	Execute a move operation.
OTHER	Execute an application-defined operation.

fsReply (USHORT)
Reply flags.

Replay flags for the message. These flags can be set as follows:

DMFL_NATIVERENDER	The source does not support rendering for this object. A source should not set this flag unless it provides sufficient information at the time of the drop for the target to perform the rendering operation. The target must send DM_ENDCONVERSATION to the source after carrying out the rendering operation, or when it elects not to do a native rendering.
DMFL_RENDERRETRY	The source supports rendering for the object, but does not support the selected rendering mechanism and format. The target can try another mechanism and format by sending another DM_RENDER message. If the target does not retry, it must send a DM_RENDERCOMPLETE message to the source. This flag is set in conjunction with the DMFL_NATIVERENDER flag.

Summary

Following are tables that describe the OS/2 functions used by the source, functions used by the target, window messages, notification code, and data structures used in direct manipulation:

Table 12-3. Direct Manipulation Functions Used by the Source

Function Name	Description
DrgAddStrHandle	Creates a handle for an input string.
DrgAllocDraginfo	Allocates a DRAGINFO data structure in shared memory.
DrgAllocDragtransfer	Allocates a specified number of DRAGTRANSFER data structures from a single segment.
DrgDrag	Handles movement of the source-specified pointer around the screen. Provides visible feedback to the user.
DrgFreeDraginfo	Deallocates the memory associated with a DRAGINFO data structure.
DrgLazyDrag	Called when alt + mouse button 2 is pressed to initiate a pickup and drop (lazy drag) operation.
DrgReallocDraginfo	Releases the current DRAGINFO data structure and reallocates a new one.
DrgSetDragitem	Initializes each object element in a DRAGINFO data structure.

Table 12-4 (Page 1 of 3). Direct Manipulation Functions Used by the Target

Function Name	Description
DrgAcceptDroppedFiles	Handles the file direct manipulation protocol for a given window.
DrgAccessDraginfo	Provides access to the shared segment containing the DRAGINFO data structure.
DrgCancelLazyDrag	Cancels the current drag operation.
DrgDeleteDraginfoStrHandles	Does a DrgDeleteStrHandle for all string handles in a DRAGINFO data structure.
DrgDeleteStrHandle	Disassociates a string from the handle that was assigned to it by DrgAddStrHandle.
DrgDragFiles	Begins a direct manipulation operation for one or more files.
DrgFreeDraginfo	Releases the memory associated with a DRAGINFO data structure. This function should be called when the target no longer needs the DRAGINFO structure, or has previously called DrgAccessDraginfo, or a drop has occurred.
DrgFreeDragtransfer	Frees the storage associated with a DRAGTRANSFER data structure.

Table 12-4 (Page 2 of 3). Direct Manipulation Functions Used by the Target	
Function Name	**Description**
DrgGetPS	Returns a handle to a cached presentation space that the target can use to provide target emphasis.
DrgLazyDrop	Invokes a drop during a Pickup and Drop operation.
DrgPostTransferMsg	Posts a message to the other application involved in the direct manipulation.
DrgPushDraginfo	Gives a process access to a DRAGINFO data structure.
DrgQueryDraginfoPtrFromDragitem	Obtains a pointer to the DRAGINFO data structure associated with a given DRAGITEM data structure.
DrgQueryDraginfoPtrFromHwnd	Determines whether a particular window has allocated a DRAGINFO data structure.
DrgQueryDragitem	Copies a given object in a DRAGINFO data structure.
DrgQueryDragitemCount	Returns the number of objects involved in a drag operation.
DrgQueryDragitemPtr	Returns a pointer to a given DRAGITEM data structure.
DrgQueryDragStatus	This function determines the status of the current drag operation.
DrgQueryNativeRMF	Returns the ordered pair representing the native rendering mechanism and format for an object.
DrgQueryNativeRMFLen	Returns the length of the string representing the native rendering mechanism and format of an object, excluding the null terminating byte.
DrgQueryStrName	Returns the contents of a string associated with a given string handle that was created by DrgAddStrHandle.
DrgQueryStrNameLen	Returns the length of the string associated with a given string handle that was created by DrgAddStrHandle.
DrgQueryTrueType	Returns the string representing the true type of an object being dragged.
DrgQueryTrueTypeLen	Returns the length of the string representing the true type of an object being dragged, excluding the null terminating byte.
DrgReleasePS	Releases the cache presentation space obtained using DrgGetPS.
DrgSendTransferMsg	Sends a message to the other application involved in the direct manipulation.
DrgSetDragImage	Enables a target to provide a customized image to be dragged.
DrgSetDragPointer	Enables a target to provide a customized image while it is the target of a drop.
DrgVerifyNativeRMF	Verifies that the native rendering mechanism and format for an object being dragged is one of a set of application-supplied rendering mechanisms and formats.

Table 12-4 (Page 3 of 3). Direct Manipulation Functions Used by the Target	
Function Name	**Description**
DrgVerifyRMF	Verifies that an application-specified rendering mechanism and format is valid for an object being dragged.
DrgVerifyTrueType	Verifies that an application-specified type is the true type of the object being dragged.
DrgVerifyType	Verifies that an application-specified type is valid for an object being dragged.
DrgVerifyTypeSet	Returns the intersection between the contents of the string represented by the type string handle and an application-supplied type string.

Table 12-5 (Page 1 of 2). Direct Manipulation Window Messages	
Message Name	**Description**
DM_DISCARDOBJECT	Sent to a source that supports the "DRM_DISCARD" rendering method.
DM_DRAGERROR	Sent to the caller of DrgDragFiles or DrgAcceptDroppedFiles when an error occurs during a Move or Copy operation.
DM_DRAGFILECOMPLETE	Sent when a direct manipulation operation on a file is complete.
DM_DRAGLEAVE	Sent to a window that is being dragged over when one of the following occurs: • The object is dragged outside the boundaries of the window. • The drag operation is terminated while the object is over the window.
DM_DRAGOVER	Lets the window under the pointer determine whether the object currently being dragged can be dropped.
DM_DRAGOVERNOTIFY	Sent to the source of a drag immediately after a DM_DRAGOVER message is sent to a target window.
DM_DROP	Sent to the target when the dragged object is dropped.
DM_DROPHELP	Requests help for the current drag operation.
DM_DROPNOTIFY	Notifies the source window of a drop operation.
DM_EMPHASIZETARGET	Sent to the caller of DrgAcceptDroppedFiles to tell it to either apply or remove target emphasis from itself.
DM_ENDCONVERSATION	The target used this message to notify a source that a drag operation is complete.
DM_FILERENDERED	Sent to the window handling the drag conversation for the caller of DrgDragFiles.
DM_PRINTOBJECT	Sent to a source to request it to print the current view of an object.

Table 12-5 (Page 2 of 2). Direct Manipulation Window Messages

Message Name	Description
DM_RENDER	Used to request a source to provide a rendering of an object in a specified rendering mechanism and format.
DM_RENDERCOMPLETE	Posted by a source to a target window.
DM_RENDERFILE	Sent to the caller of DrgDragFiles to tell it to render a file.
DM_RENDERPREPARE	Tells a source to prepare for the rendering of an object.
WM_PICKUP	Adds objects to the pickup set during a Pickup and Drop operation.

Table 12-6. Direct Manipulation Notification Code

Code Name	Description
CN_PICKUP	Determines if mouse position is over target object, white space, or desktop. The container control sends a WM_CONTROL message with the CN_PICKUP notification code to its owner when a Pickup and Drop operation is initiated over a container (WM_PICKUP message is received).

Table 12-7. Direct Manipulation Data Structures

Data Structure Name	Description
DRAGIMAGE	Dragged-image structure.
DRAGINFO	Drag-information data structure.
DRAGITEM	Drag-object data structure.
DRAGTRANSFER	Drag-conversation data structure.

Chapter 13. Hooks

A *hook* is a point in a system-defined function where an application can supply additional code that the system processes as though it were part of the function. This chapter describes how to use hooks in PM applications.

About Hooks

Many operating system functions provide points where an application can *hook in* its own code to enhance or override the default processing of the function. Most hooks enable an application to monitor some aspect of the message stream. For example, the input hook enables an application to monitor all messages posted to a particular message queue.

A hook function can be associated with the system-message queue, so that it monitors messages for all applications. These system-queue hook functions can be called in the context of any application. However, they must be defined in separate dynamic link library (DLL) modules, because it is not possible to call application-module procedures from other applications.

A hook function can also be associated with the message queue of an individual thread, so that it monitors messages for that thread only. These message-queue hook functions are called only in the context of the thread. Therefore, these hook functions are typically defined locally.

OS/2 operating system contains many types of hooks, and the system maintains a separate *hook list* for each type of hook supported.

Hook Lists

A *hook list* contains the addresses of the functions that the system calls while processing a hook. An application can take advantage of a particular type of hook by defining a hook function and using WinSetHook to enter the address of the function in the corresponding hook list. To specify the hook type in WinSetHook, the application uses one of the constants listed in Table 13-1.

Table 13-1 (Page 1 of 2). Hook Constants

Constant Name	Description
HK_CODEPAGECHANGED	Enables applications to determine when the code page changes.
HK_FINDWORD	Enables applications to control where WinDrawText places line breaks.
HK_HELP	Monitors the WM_HELP message.
HK_INPUT	Monitors messages in the specified message queue.
HK_JOURNALPLAYBACK	Enables applications to insert messages into the system message queue.

Table 13-1 (Page 2 of 2). Hook Constants

Constant Name	Description
HK_JOURNALRECORD	Allows applications to record mouse and keyboard input messages.
HK_MSGFILTER	Monitors input events during system modal loops.
HK_SENDMSG	Monitors messages sent by using WinSendMsg.

While executing a function that contains a hook, the system checks for any function addresses in the hook list that correspond to the type of hook. If an address is found, the system tries to locate and execute the function.

Hook Chains

In the hook lists associated with most message-monitoring hooks, the function addresses are linked to form chains. The system passes a message to each hook function in the list, one after the other. Each function can modify the message or stop its progress through the chain, thereby preventing it from reaching the next hook or the destination window. The system calls chained hook functions in last-installed, first-called order.

Hook Types

Each type of hook passes a characteristic set of arguments to the functions referenced in the corresponding hook list. For an application to use a particular hook, it must define a function that processes those arguments and enter the address of the function in the hook list using WinSetHook. This section describes the types of hooks available in OS/2 operating system and the requirements of the functions that process each hook type.

Input Hook

The *input hook* enables an application to monitor the system-message queue or an application-message queue. The system calls an input-hook function whenever WinGetMsg or WinPeekMsg is about to return a message. Typically, an application uses the input hook to monitor mouse and keyboard input and other messages posted to a queue. Figure 13-1 shows the syntax for an input-hook function.

```
BOOL EXPENTRY InputHook(HAB hab, PQMSG pQmsg, ULONG fs)
```

Figure 13-1. Syntax for an Input-Hook Function

The *pQmsg* parameter is a pointer to a QMSG data structure that contains information about the message.

The *fs* parameter of InputHook can contain the following flags from WinPeekMsg, indicating whether or not the message is removed from the queue:

```
PM_NOREMOVE
PM_REMOVE
```

If an input-hook function returns TRUE, the system does not pass the message to the rest of the hook chain or to the application. If the function returns FALSE, the system passes the message to the next hook in the chain or to the application if no other hooks exist.

An input-hook function can modify a message by changing the contents of the QMSG data structure, then returning FALSE to pass the modified message to the rest of the chain. The following problems can occur when a hook modifies a message:

- If the caller uses WinPeekMsg or WinGetMsg with a message filter range (msgFilterFirst through msgFilterLast), the message is checked before the hook functions are called, not after. If the input-hook function modifies the *msg* field of the QMSG data structure, the caller can receive messages that are not in the range of the message filter of the caller.
- If the input-hook function changes a WM_CHAR message from one character into another—for example, if the function modifies all Tab messages into F6 messages—an application that depends on the key state is unable to interpret the result. (When the Tab key is translated into the F6 key, the application receives the F6 keystroke and enters a process loop, waiting for the F6 key to be released; the application calls WinGetKeyState with the HWND_DESKTOP and VK_F6 arguments).

Send-Message Hook

The *send-message hook* enables an application to monitor messages that the system does not post to a queue. The system calls a send-message hook function while processing WinSendMsg, before delivering the message to the recipient window. By installing an input-hook function and a send-message hook function, an application can monitor all window messages effectively. Figure 13-2 shows the syntax for a send-message hook function.

```
VOID EXPENTRY SendMsgHook(HAB hab, PSMHSTRUCT psmh,
                          BOOL fInterTask)
```

Figure 13-2. Syntax for a Send-Message Hook Function

The *psmh* parameter is a pointer to an SMHSTRUCT data structure that contains information about the message.

The *fInterTask* parameter is TRUE if the message is sent between two threads, or FALSE if the message is sent within a thread.

A send-message hook function does not return a value, and the next function in the chain is always called. The function can modify values in the SMHSTRUCT data structure before returning.

Message-Filter Hook

The *message-filter hook* allows an application to provide input filtering (such as monitoring hot keys) during system-modal loops. The system calls a message-filter hook function while tracking the window size and movement, displaying a modal dialog window or message box, tracking a scroll bar, and during window-enumeration operations. Figure 13-3 shows the syntax for a message-filter hook function.

```
BOOL EXPENTRY MsgFilterHook(HAB hab, PQMSG pQmsg, ULONG msgf)
```

Figure 13-3. Syntax for a Message-Filter Hook Function

The *msgf* parameter can have one of the three values shown in Table 13-2.

Table 13-2. Hook Parameter Values (Message-Filter)

Parameter Value	Description
MSGF_DIALOGBOX	Message originated while processing a modal dialog window or a message box.
MSGF_MESSAGEBOX	Message originated while processing a message box.
MSGF_TRACK	Message originated while tracking a control (such as a scroll bar).

The *pQmsg* parameter of MsgFilterHook is a pointer to a QMSG data structure containing information about the message.

If a message-filter hook function returns TRUE, the system does not pass the message to the rest of the hook chain or to the application. If the function returns FALSE, the system passes the message to the next hook function in the chain or to the application if no other functions exist.

This hook enables applications to perform message filtering during modal loops that is equivalent to the typical filtering for the main message loop. For example, applications often examine a new message in the main event loop between the time they retrieve the message from the queue and the time they dispatch it, performing special processing as appropriate. An application usually cannot do this sort of filtering during a modal loop, because the system executes the loop created by WinGetMsg and WinDispatchMsg. If an application installs a message-filter hook function, the system calls the function between WinGetMsg and WinDispatchMsg in the modal processing loop.

An application can also call the message-filter hook function directly by calling WinCallMsgFilter. With this function, the application can use the same code as the main message loop to filter messages during modal loops. To do so, the application encapsulates the filtering operations in a message-filter hook function and calls WinCallMsgFilter between WinGetMsg and WinDispatchMsg calls, as shown in the following code fragment illustrated in Figure 13-4 on page 13-5.

```
while (WinGetMsg(hab, (PQMSG) &qmsg, (HWND) NULL, 0, 0))
{
   if (!WinCallMsgFilter(hab, (PQMSG) &qmsg, 0))
      WinDispatchMsg(hab, (PQMSG) &qmsg);
}
```

Figure 13-4. Sample Code Calling WinCallMsgFilter Directly

The last argument of WinCallMsgFilter is passed to the hook function; the application can enter any value. By defining a constant such as MSGF_MAINLOOP, the hook function can use that value to determine from where the function was called.

Journal-Record Hook

The *journal-record hook* allows an application to monitor the system-message queue and to record input events. Typically, an application uses this hook to record a sequence of mouse and keyboard events that it can play back later by using the journal-playback hook. A journal-record hook function can be associated only with the system-message queue. Figure 13-5 shows the syntax for a journal-record hook function.

```
VOID EXPENTRY JournalRecordHook(HAB hab, PQMSG pQmsg)
```

Figure 13-5. Syntax for a Journal-Record Hook Function

The *pQmsg* parameter is a pointer to a QMSG data structure containing information about the message. The system calls the journal-record hook function after processing the raw input enough to create valid WM_CHAR or mouse messages and after setting the *window-handle* field of the QMSG data structure.

A journal-record hook function does not return a value, and the system always calls the next function in the chain. Typically, a journal-record hook function saves the input events to a disk file to be played back later. The *hwnd* field of the QMSG data structure is not important and is ignored when the message is played back.

The following messages are passed to the journal-record hook:

```
WM_CHAR
WM_BUTTON1DOWN
WM_BUTTON1UP
WM_BUTTON2DOWN
WM_BUTTON2UP
WM_BUTTON3DOWN
WM_BUTTON3UP
WM_MOUSEMOVE.
```

The positions stored in the mouse messages are in screen coordinates. The system does not combine mouse clicks into double clicks before calling the hook, because there is no guarantee that both clicks will be in the same window when they are played back.

The system passes a WM_JOURNALNOTIFY message to the journal-record hook function whenever an application calls WinGetPhysKeyState or WinQueryQueueStatus. This message is necessary because the system-message queue is only one message deep while a playback hook is active. For example, the user might press the A, B, and C keys while in record mode. While the application is processing the *A* character message, the B key might be down; WinGetPhysKeyState returns this information. However, during playback mode, the system knows only that it currently is processing the A key.

Journal-Playback Hook

The *journal-playback hook* enables an application to insert messages into the system-message queue. Typically, an application uses this hook to play back a series of mouse and keyboard events that were recorded earlier using the journal-record hook. A journal-playback hook function can be associated only with the system-message queue.

Regular mouse and keyboard input is disabled as long as a journal-playback hook is installed. It is important to notice that, because mouse and keyboard input are disabled, this hook can easily hang the system. Figure 13-6 shows the syntax for a journal-playback hook function.

```
ULONG EXPENTRY JournalPlaybackHook(HAB hab, BOOL fSkip,
                                   PQMSG pQmsg)
```

Figure 13-6. Syntax for a Journal-Playback Hook Function

The *pQmsg* parameter is a pointer to a QMSG data structure that the journal-playback hook function fills in with the message to be played back. If the *fSkip* parameter is FALSE, the function fills in the QMSG data structure with the current recorded message. The function returns the same message each time it is called, until *fSkip* is TRUE. The same message is returned many times if an application is examining the queue but not removing the message. If *fSkip* is TRUE, the function advances to the next message without filling in the QMSG data structure, because the *pQmsg* parameter is NULL when *fSkip* is TRUE.

The journal-playback hook returns a ULONG time-out value that tells the system how many milliseconds to wait before processing the current message from the playback hook. This enables the hook to control the timing of the events it plays back.

The *time* field of the QMSG data structure is filled in with the current time before the playback hook is called. The hook should use the time stored in this field, instead of the system clock, to set up delays between events.

Help Hook

The *help hook* allows an application to include online help. The system calls a help-hook function during the default processing of the WM_HELP message. Help processing is done in two stages: creating the WM_HELP message and calling the help hook. The WM_HELP message can come from the following sources:

- WM_CHAR message, after translation by an ACCEL data structure with the AF_HELP style. The default system accelerator table translates the F1 key into a help message. The WM_HELP message is posted to the current focus window, which can be a menu, a button, a frame, or your client window.
- Menu-bar selection, when the MIS_HELP style is specified for the menu-bar item. The WM_HELP message is posted to the current focus window.
- Dialog-window push button, when the BS_HELP style is specified for the push button. The WM_HELP message is posted to the owner window of the button, which normally is the dialog window.
- Message box, when the MB_HELP style is specified for the message box. The WM_HELP message is posted to the message box.

The WM_HELP message is posted to the current focus window. The default processing in WinDefWindowProc is to pass the message up to the parent window. If the message reaches the client window, it can be processed there. If the message reaches a frame window, the default frame-window procedure calls the help hook. The help hook is also called if a WM_HELP message is generated while the application is in menu mode, that is, while a selection is being made from a menu. Figure 13-7 shows the syntax for a help-hook function.

```
BOOL EXPENTRY HelpHook(HAB hab, ULONG usMode, ULONG idTopic,
                       ULONG idSubTopic, PRECTL prcPosition)
```

Figure 13-7. Syntax for a Help-Hook Function

If a help-hook function returns TRUE, the system does not call the next help-hook function in the chain. If the function returns FALSE, the system calls the next help-hook function in the chain. The arguments passed to the function provide contextual information, such as the screen coördinates of the focus window and whether the message originated in a message box or a menu.

The WM_HELP message often goes to a frame window instead of to the client window. The frame window processes a WM_HELP message as follows:

- If the window with the focus is the FID_CLIENT window, the frame window passes the WM_HELP message to the FID_CLIENT window.
- If the parent of the window with the focus is the FID_CLIENT frame-control window, the frame window calls the help hook, specifying in Figure 13-8.

```
Mode     = HLPM_FRAME
Topic    = frame-window identifier
Subtopic = focus-window identifier
Position = screen coordinates of focus window
```

Figure 13-8. Fields to Specify when Focus Is FID_CLIENT

- If the parent of the focus window is not an FID_CLIENT window (it could be the frame window or a second-level dialog window), the frame window calls the help hook, specifying in Figure 13-9.

```
Mode     = HLPM_WINDOW
Topic    = identifier of parent of focus window
Subtopic = focus-window identifier
Position = screen coordinates of focus window
```

Figure 13-9. Fields to Specify when Focus Is Not FID_CLIENT

An application receives the WM_HELP message in its dialog-window procedure. The application can ignore the message, in which case the frame-window action occurs as described, or the application can handle the WM_HELP message directly.

Menu windows receive a WM_HELP message when the user presses the Help accelerator key (F1 by default) while a menu is displayed. Menu windows process WM_HELP messages by calling the help hook, specifying in Figure 13-10.

```
Mode     = HLPM_MENU
Topic    = identifier of pull-down menu
Subtopic = identifier of selected item in pull-down menu
Position = screen coordinates of selected item
```

Figure 13-10. Fields to Specify when Processing WM_HELP

A help-hook function should respond by displaying information about the selected menu item.

WinDefWindowProc processes WM_HELP messages by passing the message to the parent window. Typically, the message moves up the parent chain until it arrives at a frame window.

Find-Word Hook

The *find-word hook* allows an application to control where WinDrawText breaks a character string that is too wide for the drawing rectangle. If the DT_WORDBREAK flag is set, the system calls this hook from within WinDrawText. Typically, this hook is used to avoid awkward line breaks in applications that use double-byte character sets. Figure 13-11 on page 13-9 shows the syntax for a find-word hook function.

```
ULONG EXPENTRY FindWordHook(USHORT usCodePage,
                            PSZ pszText, ULONG cb,
                            ULONG ich,
                            PULONG pichStart,
                            PULONG pichEnd,
                            PULONG pichNext)
```

Figure 13-11. Syntax for a Find-Hook Function

The *usCodePage* parameter contains the code page identifier of the string to be formatted; the *pszText* parameter contains a pointer to the actual string.

The *cb* parameter contains a value specifying the number of bytes in the string. This value is 0 if the string is null-terminated.

The *ich* parameter contains the index of the character in the string that intersects the right edge of the drawing rectangle.

A find-word hook function uses these four parameters to determine the word that contains the intersecting character. It then fills the remaining three parameters, *pichStart*, *pichEnd*, and *pichNext*, with the indexes of the starting character of the word, ending character of the word, and starting character of the next word in the string.

If the find-word hook function returns TRUE, WinDrawText draws the string only up to, but not including, the specified word. If the function returns FALSE, WinDrawText formats the string in the default manner.

Codepage-Changed Hook

The *codepage-changed hook* notifies an application when the code page associated with the specified message queue has been changed. The system calls a codepage-changed hook function after setting the new code page. Typically, the codepage-changed hook is used in applications that support multiple languages. Figure 13-12 shows the syntax for a codepage-changed hook function.

```
VOID EXPENTRY CodePageChangedHook(HMQ hmq, USHORT usOldCodepage,
                                  USHORT usNewCodepage)
```

Figure 13-12. Syntax for a Codepage-Changed Hook Function

The *hmq* parameter receives the handle of the message queue that is changing its codepage. The usOldCodepage is the codepage identifier of the previous code page; usNewCodepage is the identifier of the new code page.

A codepage-changed hook function does not return a value, and the system always calls the next function in the chain.

Using Hooks

This section explains how to perform the following tasks:

- Install hook functions
- Release hook functions and free memory
- Record and play back input events.

Note: Much of the sample codes in this section are part of a complete program which is illustrated in "Sample Code for Hooks" on page 13-14.

Installing Hook Functions

You can install hook functions by calling WinSetHook, specifying the type of hook that calls the function—whether the function is to be associated with the system-message queue or with the queue of a particular thread—and a pointer to a function entry point. The sample code illustrated in Figure 13-13 shows how to install a hook function into the message queue of a thread.

```
BOOL EXPENTRY MyInputHook(HAB, PQMSG, USHORT);
HAB  hab;
HMQ  hmq;

WinSetHook(hab,                    /* Anchor block handle          */
           hmq,                    /* Thread message queue         */
           HK_INPUT,               /* Called by the input hook     */
           (PFN) MyInputHook,      /* Address of input-hook function */
           (HMODULE)NULL);         /* Function is in appl. module  */
```

Figure 13-13. Sample Code Installing a Hook into a Thread Message Queue

Place hook functions associated with the system-message queue in a dynamic link library (DLL) separate from the application that installs the hook function. The installing application needs the handle of the DLL module before it can install the hook function. DosLoadModule, given the name of the DLL, returns the handle of the DLL module. Once you have the handle, you can call DosQueryProcAddr to obtain the address of the hook function. Finally, use WinSetHook to install the hook-function address in the appropriate hook list. WinSetHook passes the module handle, a pointer to the hook-function entry point, and NULL for the message-queue argument, indicating that the hook function should be associated with the system queue. The sample code illustrated in Figure 13-14 on page 13-11 shows functions, called from the application's main routine, that initialize a DLL and install the hook function.

```
HAB      habDLL;
HMODULE hMod;
PFN      pfnInput;

/**********************************************************************/
/*  InitDLL: This function sets up the DLL and sets all variables.     */
/**********************************************************************/
void EXPENTRY InitDLL(HAB hab)
{
    habDLL = hab;

/**********************************************************************/
/*  Load the dll - actually, just get our module handle.              */
/**********************************************************************/
    DosLoadModule(NULL, 0, "HOOKDLL", &hMod);

/**********************************************************************/
/*  Find the address of the input hook procedure.                     */
/**********************************************************************/
    DosQueryProcAddr(hMod, 0, "InputProc", &pfnInput);
}

/**********************************************************************/
/*  StartInputHook: This function starts the hook filtering.          */
/**********************************************************************/
void EXPENTRY StartInputHook(void)
{

/**********************************************************************/
/*  Set a hook to our input filter routine.                           */
/**********************************************************************/
    WinSetHook(habDLL, NULLHANDLE, HK_INPUT, pfnInput, hMod);
}
```

Figure 13-14. Sample Code Installing a Hook in a DLL

Releasing Hook Functions

You can release a hook function and remove its address from the hook list by calling WinReleaseHook with the same arguments that you used when installing the hook function, as shown in the sample code illustrated in Figure 13-15 on page 13-12.

```
BOOL EXPENTRY MyInputHook(HAB, PQMSG, USHORT);
HAB  hab;
HMQ  hmq;

WinReleaseHook(hab,                    /* Anchor block handle          */
               hmq,                    /* Thread message queue         */
               HK_INPUT,               /* Called by the input hook     */
               (PFN) MyInputHook,      /* Address of input-hook function */
               (HMODULE)NULL);         /* Function is in appl. module  */
```

Figure 13-15. Sample Code Releasing a Hook from a Thread Message Queue

Release all hook functions before the application terminates, even though the system automatically releases them if the application does not. You also need to free the memory associated with the hook.

Freeing Memory

How memory for the hook is freed depends on the type of hook chain an event is linked to:

- Queue (current) hook chain
- System hook chain.

A queue hook chain is a private hook chain. It applies only to the current calling thread that created the queue with which the hook chain is associated. It may or may not reside in a DLL. If it is not associated with a DLL, its memory can be freed by WinReleaseHook, as shown in the sample code illustrated in Figure 13-15.

A system hook chain must reside in a DLL; therefore, it affects the entire system. WinSetHook allocates memory and associates it with a DLL. This memory is not freed until the DLL module is freed. WinReleaseHook cannot free the DLL's memory, because another process cannot free the DLL and its associated memory. However, this memory can be freed by launching a thread that does the following:

- Loads the DLL and sets the hook
- When the playback sequence is complete, releases the hook and frees the DLL, thus relinquishing its memory.

As long as any DLL associated with the hook is alive, WinReleaseHook cannot free the memory.

The implication here is straightforward:

- If a queue hook is being installed and it is not associated with a DLL, WinReleaseHook can free its memory.
- If a system hook is being installed, its memory cannot be freed until the DLL is freed. WinReleaseHook has to do a DosFreeModule, but it cannot do this for another process. The application must use DosFreeModule to relinquish hook-allocated memory associated with a DLL.

The sample code illustrated in Figure 13-16 shows a function, called from an application's main routine, that releases the hook and frees the memory of the hook installed in Figure 13-14 on page 13-11.

```
/**********************************************************************/
/*  StopInputHook: This function stops the hook filtering.            */
/**********************************************************************/
void EXPENTRY StopInputHook(void)
{

/**********************************************************************/
/*  Drop a hook to our input filter routine.                          */
/**********************************************************************/
    WinReleaseHook(habDLL, NULLHANDLE, HK_INPUT, pfnInput, hMod);

/**********************************************************************/
/*  Decrement the DLL usage count.                                    */
/**********************************************************************/
    DosFreeModule(hMod);
}
```

Figure 13-16. Sample Code Releasing a Hook from a DLL

Recording and Playing Back Input Events

To record and play back input events, use the journal-record hook to create a local queue to store the recorded events, then use the journal-playback hook to create a second thread to read from the queue. Do not attempt to spend any significant cycles within JournalRecordHook. Because the recorded events include semaphores, Win calls, and I/O functions, it can cause system deadlocks. The pseudocode illustrated in Figure 13-17 describes how to play back recorded functions.

```
Store the passed time as the current time

If the system requests a new message to be prepared (skip is TRUE)
   If all messages have been played back, release the Playback Hook
      (After release, your playback hook function will still
       be called a few times more.  So leave a null mouse move
       message as the next message to be copied.)
```

Figure 13-17 (Part 1 of 2). Pseudocode Describing how to Play Back Recorded Functions

```
    Otherwise:
       Save the last message time
       Copy the new message to the passed qmsg buffer
       Calculate the time until the next message
       (You should know, from the recorded times, the
        delta time which actually occurred between each
        message.  During playback you will need to calculate
        the amount of time remaining between the time passed
        to you in the qmsg buffer, i.e., "current time", and
        the time at which the next message is due to be
        kicked off.)
            .
 Otherwise (skip is FALSE, so the system wants a peek
            at the current message):
    Copy the existing (current) message to the passed qmsg buffer

 Recalculate and return the REMAINING delay for the current message
```

Figure 13-17 (Part 2 of 2). Pseudocode Describing how to Play Back Recorded Functions

An alternative method for installing a system-queue hook function is to provide an installation function in the DLL along with the hook function. With this method, the installing application does not need the handle of the DLL module. By linking with the DLL, the application gains access to the installation function, which can supply the DLL module handle and other details in the call to WinSetHook. The DLL can also contain a function that releases the system-queue hook function. The application can call this hook-releasing function when it terminates.

Sample Code for Hooks

This section illustrates a complete hook sample program. Several parts of this program are explained in "Using Hooks" on page 13-10.

Hooks Application Sample Code

The hook application includes the following files:

- Hookdemo.C
- Hookdll.C
- Hookdemo.RC
- Hookdemo.H
- Hookdemo.DEF
- Hookdemo.LNK
- Hookdll.DEF
- Hookdll.LNK
- Hookdemo.MAK

Figure 13-18 on page 13-15 illustrates the hook application sample code.

```
============
HOOKDEMO.C
============

#define  INCL_WIN
#define  INCL_GPI

#include <os2.h>
#include "hookdemo.h"

#pragma  linkage (main,optlink)
INT      main(VOID);

/*************************************************************************/
/*  Main() - program entry point.                                        */
/*************************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);

HAB     hab;
HWND    hFrameWnd;
PFNWP   SysWndProc;

INT main (VOID)
{
    HMQ         hmq;
    FRAMECDATA  fcd;
    QMSG        qmsg;

    if (!(hab = WinInitialize(0)))
      return FALSE;

/*************************************************************************/
/*  Initialize our DLL, which holds the system hook routines.            */
/*************************************************************************/
    InitDLL(hab);

    if (!(hmq = WinCreateMsgQueue(hab, 0)))
      return FALSE;
```

Figure 13-18 (Part 1 of 9). Sample Code for a Hook Application

```
/***********************************************************************/
/*  Setup the frame control data for the frame window.                 */
/***********************************************************************/
    fcd.cb            = sizeof(FRAMECDATA);
    fcd.flCreateFlags = FCF_TITLEBAR      |
                        FCF_SYSMENU       |
                        FCF_MENU          |
                        FCF_SIZEBORDER    |
                        FCF_SHELLPOSITION |
                        FCF_MINMAX        |
                        FCF_TASKLIST;
    fcd.hmodResources = NULLHANDLE;
    fcd.idResources   = HOOKDEMO;

/***********************************************************************/
/*  Create the frame - it will hold the container control.             */
/***********************************************************************/
    hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                                WC_FRAME,
                                "HookDemo",
                                0, 0, 0, 0, 0,
                                NULLHANDLE,
                                HWND_TOP,
                                0,
                                &fcd,
                                NULL);

/***********************************************************************/
/*  Verify that the frame was created; otherwise, stop.                */
/***********************************************************************/
    if (!hFrameWnd)
      return FALSE;

/***********************************************************************/
/*  Set an icon for the frame window.                                  */
/***********************************************************************/
    WinSendMsg(hFrameWnd,
               WM_SETICON,
               (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                          SPTR_FOLDER,
                                          FALSE),
                NULL);
```

Figure 13-18 (Part 2 of 9). Sample Code for a Hook Application

```
/**********************************************************************/
/*  We must intercept the frame window's messages.                    */
/*  We save the return value (the current WndProc),                   */
/*  so we can pass it all the other messages the frame gets.          */
/**********************************************************************/
    SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

    WinShowWindow(hFrameWnd,TRUE);

/**********************************************************************/
/*  Standard PM message loop - get it, dispatch it.                   */
/**********************************************************************/
    while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
    {
       WinDispatchMsg(hab, &qmsg);
    }

/**********************************************************************/
/*  Clean up on the way out.                                          */
/**********************************************************************/
    WinDestroyWindow(hFrameWnd);
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);

    return TRUE;
}

/**********************************************************************/
/*  LocalWndProc() - window procedure for the frame window.           */
/*  Called by PM whenever a message is sent to the frame.             */
/**********************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
    char                szBuffer[80];
    POINTL              pt;
    int                 x;

    switch(msg)
    {
```

Figure 13-18 (Part 3 of 9). Sample Code for a Hook Application

```
/*********************************************************************/
/*  Send the message to the usual WC_FRAME WndProc.                  */
/*********************************************************************/
        case WM_COMMAND:
            switch (SHORT1FROMMP(mp1))
            {

/*********************************************************************/
/*  Start the hook routine - it stops all WM_COMMAND messages.       */
/*  (which means all these other messages will be ignored).          */
/*********************************************************************/
                case IDM_START:
                    StartInputHook();
                    break;

                case IDM_STOP:
                    StopInputHook();
                    break;

                case IDM_EXIT:
                    WinPostMsg(hwnd, WM_CLOSE, 0, 0);
                    break;

                default:
                    return (*SysWndProc)(hwnd, msg, mp1, mp2);
            }
            break;

/*********************************************************************/
/*  Send the message to the usual WC_FRAME WndProc.                  */
/*********************************************************************/
        default:
            return (*SysWndProc)(hwnd, msg, mp1, mp2);
            break;
    }

    return FALSE;
}
```

Figure 13-18 (Part 4 of 9). Sample Code for a Hook Application

```
============
HOOKDLL.C
============
#define  INCL_WIN
#define  INCL_DOS
#include <os2.h>

/**********************************************************************/
/*  Global variables.                                                 */
/**********************************************************************/
HAB     habDLL;
HMODULE hMod;
PFN     pfnInput;

/**********************************************************************/
/*  InitDLL: This function sets up the DLL and sets all variables     */
/**********************************************************************/
void EXPENTRY InitDLL(HAB hab)
{
    habDLL = hab;

/**********************************************************************/
/*  Load the DLL - actually, just get our module handle.              */
/**********************************************************************/
    DosLoadModule(NULL, 0, "HOOKDLL", &hMod);

/**********************************************************************/
/*  Find the address of the input hook procedure.                     */
/**********************************************************************/
    DosQueryProcAddr(hMod, 0, "InputProc", &pfnInput);
}

/**********************************************************************/
/*  StartInputHook: This function starts the hook filtering.          */
/**********************************************************************/
void EXPENTRY StartInputHook(void)
{
```

Figure 13-18 (Part 5 of 9). Sample Code for a Hook Application

```
/************************************************************************/
/*  Set a hook to our input filter routine.                             */
/************************************************************************/
  WinSetHook(habDLL, NULLHANDLE, HK_INPUT, pfnInput, hMod);
}

/************************************************************************/
/*  StopInputHook: This function stops the hook filtering.              */
/************************************************************************/
void EXPENTRY StopInputHook(void)
{

/************************************************************************/
/*  Drop a hook to our input filter routine.                            */
/************************************************************************/
    WinReleaseHook(habDLL, NULLHANDLE, HK_INPUT, pfnInput, hMod);

/************************************************************************/
/*  Decrement the DLL usage count.                                      */
/************************************************************************/
    DosFreeModule(hMod);
}

/************************************************************************/
/*  InputProc: This is the input filter routine.                        */
/*  While the hook is active, all messages come here                    */
/*  before being dispatched.                                            */
/************************************************************************/
BOOL EXPENTRY InputProc(HAB hab, PQMSG pqMsg, ULONG fs)
{

/************************************************************************/
/*  Check for WM_COMMAND messages.                                      */
/************************************************************************/
    if (pqMsg->msg == WM_COMMAND)
    {

/************************************************************************/
/*  Ignore all WM_COMMAND messages (stops menu processing).             */
/************************************************************************/
      return TRUE;
    }
```

Figure 13-18 (Part 6 of 9). Sample Code for a Hook Application

```
/**********************************************************************/
/*  Pass the message on to the next hook in line.                     */
/**********************************************************************/
   return FALSE;
}

============
HOOKDEMO.RC
============
#include <os2.h>
#include "hookdemo.h"

MENU    HOOKDEMO
BEGIN
  SUBMENU       "Command",  IDM_CMD
  BEGIN
    MENUITEM    "Start",    IDM_START
    MENUITEM    "Stop",     IDM_STOP
    MENUITEM    "Exit",     IDM_EXIT
  END
END

============
HOOKDEMO.H
============
#define HOOKDEMO        256
#define IDM_CMD         400
#define IDM_START       401
#define IDM_STOP        402
#define IDM_EXIT        403
```

Figure 13-18 (Part 7 of 9). Sample Code for a Hook Application

```
============
HOOKDEMO.DEF
============
NAME HOOKDEMO WINDOWAPI

DESCRIPTION 'PM Hooks Sample'

CODE  MOVEABLE
DATA  MOVEABLE MULTIPLE

STACKSIZE   24576
HEAPSIZE    10240

PROTMODE

============
HOOKDEMO.LNK
============
hookdemo.obj /NOI
hookdemo.exe
hookdemo.map
hookdll.lib
hookdemo.def

============
HOOKDLL.DEF
============
LIBRARY HOOKDLL

DESCRIPTION 'PM Hooks Sample'

CODE    LOADONCALL
DATA    LOADONCALL

PROTMODE

EXPORTS
  InitDLL
  StartInputHook
  StopInputHook
  InputProc
```

Figure 13-18 (Part 8 of 9). Sample Code for a Hook Application

```
============
HOOKDLL.LNK
============
hookdll.obj /NOI
hookdll.dll
hookdll.map
hookdll.def

============
HOOKDEMO.MAK
============
CC      = icc /c /Ge /Gd- /Se /Re /ss /Gm+
LINK    = link386
HEADERS = hookdemo.h

#-------------------------------------------------------------------
#  A list of all of the object files.
#-------------------------------------------------------------------
ALL_OBJ1 = hookdemo.obj

ALL_OBJ2 = hookdll.obj

all: hookdemo.exe hookdll.dll

hookdemo.res: hookdemo.rc hookdemo.h

hookdemo.obj: hookdemo.c $(HEADERS)
              icc /C /Ss /W3 hookdemo.c

hookdll.obj:  hookdll.c
              icc /C+ /Ge- /Gm+ hookdll.c

hookdll.dll:  $(ALL_OBJ2) hookdll.def hookdll.lnk
              $(LINK) @hookdll.lnk
              implib hookdll.lib hookdll.def

hookdemo.exe: $(ALL_OBJ1) hookdemo.def hookdemo.lnk hookdemo.res hook dll.lib
              $(LINK) @hookdemo.lnk
              rc -p -x hookdemo.res hookdemo.exe
```

Figure 13-18 (Part 9 of 9). Sample Code for a Hook Application

Related Functions

This section covers the functions that are related to hooks.

MsgFilterHook

This hook filters messages from inside a mode loop.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */

#include <os2.h>

BOOL MsgFilterHook  (HAB hab, PQMSG pQmsg, ULONG msgf)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pQmsg (PQMSG) – input
A queue message data structure.

msgf (ULONG) – input
Context in which the hook has been called.

MSGF_DIALOGBOX	Dialog-box mode loop.
MSGF_TRACK	Window-movement and size tracking. When this hook is used the TRACKINFO structure specified the *ptiTrackinfo* parameter of the WinTrackRect function is updated to give the current state before the hook is called. Only the *rclTrack* and the *fs* parameters are updated.
MSGF_DRAG	Direct manipulation mode loop.
MSGF_DDEPOSTMSG	DDE post message mode loop.

Returns

rc (BOOL) – returns
Processed indicator.

TRUE	The message is not passed on to the next hook in the chain or to the application
FALSE	The message is passed on to the next hook in the chain or to the application.

RegisterUserHook

This hook is called whenever a user message or data type is registered.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

BOOL RegisterUserHook (HAB hab, SHORT idContext, USHORT msg,
                       SHORT type1, SHORT dir1, SHORT type2, SHORT dir2,
                       SHORT typer, SHORT uShort, PSHORT arRMP,
                       PBOOL fRegistered)
```

Parameters

hab (HAB) – input
The application anchor block.

idContext (SHORT) – input
Origin of the call to hook.

RUMHK_DATATYPE	WinRegisterUserDatatype was called.
RUMHK_MSG	WinRegisterUserMsg was called.

msg (USHORT) – input
Message identifier.

type1 (SHORT) – input
Data type.

dir1 (SHORT) – input
Direction of message parameter 1.

type2 (SHORT) – input
Data type of message parameter 2.

dir2 (SHORT) – input
Direction of message parameter 2.

typer (SHORT) – input
Data type of message reply.

uShort (SHORT) – input
Number of data type codes.

arRMP (PSHORT) – input
Array of data type codes.

fRegistered (PBOOL) – input
Flag indicating that a message or data type was registered.

TRUE	Message/data type was registered.
FALSE	Message/data type was not registered.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Errors occurred.

WinCallMsgFilter

This function calls a message-filter hook.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */

#include <os2.h>

BOOL WinCallMsgFilter (HAB hab, PQMSG pqmsg, ULONG ulFilter)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pqmsg (PQMSG) – input
Message to be passed to the message-filter hook.

ulFilter (ULONG) – input
Filter.

MSGF_DIALOGBOX	Dialog-box mode loop.
MSGF_TRACK	Window-movement and size tracking. When this hook is used the TRACKINFO structure specified the *ptiTrackinfo* parameter of the WinTrackRect function is updated to give the current state before the hook is called. Only the *rclTrack* and the *fs* parameters are updated.
MSGF_DRAG	Direct manipulation mode loop.
MSGF_DDEPOSTMSG	DDE post message mode loop.

Returns

rc (BOOL) – returns
Message-filter hook return indicator.

TRUE	A message-filter hook returns TRUE
FALSE	All message-filter hooks return FALSE, or no message-filter hooks are defined.

WindowDCHook

This hook is called when a device context is allocated or freed.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

BOOL WindowDCHook (HAB hab, HDC hdc, HWND HWND, BOOL flAssociate)

Parameters

hab (HAB) – input
The application anchor block.

hdc (HDC) – input
The current device-context handle.

HWND (HWND) – input
The current window handle.

flAssociate (BOOL) – input
Association flag.

TRUE	Device context has been allocated.
FALSE	Device context has been freed.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Errors occurred.

WinReleaseHook

This function releases an application hook from a hook chain.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

BOOL WinReleaseHook (HAB hab, HMQ hmq, LONG lHook, PFN pAddress, HMODULE Module)

Parameters

hab (HAB) – input
Anchor-block handle.

hmq (HMQ) – input
Handle of message queue from which the hook is to be released.

HMQ_CURRENT	The hook is released from the message queue associated with the current thread (calling thread).
NULLHANDLE	The hook is released from the system hook chain.

lHook (LONG) – input
Type of hook chain.

HK_CHECKMSGFILTER	See CheckMsgFilterHook.
HK_CODEPAGECHANGE	See CodePageChangedHook.
HK_DESTROYWINDOW	See DestroyWindowHook.
HK_HELP	See HelpHook.
HK_INPUT	See InputHook.
HK_JOURNALPLAYBACK	See JournalPlaybackHook.
HK_JOURNALRECORD	See JournalRecordHook.
HK_LOADER	See LoaderHook.
HK_MSGCONTROL	See MsgControlHook.
HK_MSGFILTER	See MsgFilterHook.
HK_SENDMSG	See SendMsgHook.

pAddress (PFN) – input
Address of the hook routine.

Module (HMODULE) – input
Module handle.

NULLHANDLE	The hook procedure is in the application's .EXE file.
Module	This is the module that contains the application procedure, as returned by the DosLoadModule or DosQueryModuleHandle call.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetHook

This function installs an application procedure into a specified hook chain.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetHook (HAB hab, HMQ hmq, LONG lHookType, PFN pHookProc,
                 HMODULE Module)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hmq (HMQ) – input
Queue identity.

lHookType (LONG) – input
Hook-chain type.

HK_CHECKMSGFILTER	See CheckMsgFilterHook
HK_CODEPAGECHANGE	See CodePageChangedHook
HK_DESTROYWINDOW	See DestroyWindowHook
HK_FINDWORD	See FindWordHook
HK_FLUSHBUF	See FlushBufHook
HK_HELP	See HelpHook
HK_INPUT	See InputHook
HK_JOURNALPLAYBACK	See JournalPlaybackHook
HK_JOURNALRECORD	See JournalRecordHook
HK_LOADER	See LoaderHook
HK_LOCKUP	See LockupHook
HK_MSGCONTROL	See MsgControlHook
HK_MSGFILTER	See MsgFilterHook
HK_MSGINPUT	See MsgInputHook
HK_REGISTERUSERMSG	See RegisterUserHook
HK_SENDMSG	See SendMsgHook
HK_WINDOWDC	See WindowDCHook

pHookProc (PFN) – input
Address of the application hook procedure.

Module (HMODULE) – input
Resource identity.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

WinTrackRect

This function draws a tracking rectangle.

Syntax

```
#define INCL_WINTRACKRECT /* Or use INCL_WIN, INCL_PM, */

#include <os2.h>

BOOL WinTrackRect (HWND hwnd, HPS hps, PTRACKINFO ptiTrackinfo)
```

Parameters

hwnd (HWND) – input
Window handle where tracking is to take place.

HWND_DESKTOP	Track over the entire screen
Other	Track over specified window only.

hps (HPS) – input
Presentation-space handle.

NULLHANDLE	The *hwnd* parameter is used to calculate a presentation space for tracking. It is assumed that tracking takes place within *hwnd* and that the style of this window is not WS_CLIPCHILDREN. Thus, when the drag rectangle appears, it is not clipped by any children within the window. If the window style is WS_CLIPCHILDREN and the application causes the drag rectangle to be clipped, it must explicitly pass an appropriate presentation space.
Other	Specified presentation-space handle.

ptiTrackinfo (PTRACKINFO) – in/out
Track information.

Returns

rc (BOOL) – returns
Success indicator.

TRUE Tracking successful.

FALSE Tracking canceled, or the pointing device was already captured when this function was called.

Only one tracking rectangle can be in use at one time.

Related Data Structures

This section covers the data structures that are related to hooks.

QMSG

Message structure.

Syntax

```
typedef struct _QMSG {
HWND         hwnd;
ULONG        msg;
MPARAM       mp1;
MPARAM       mp2;
ULONG        time;
POINTL       ptl;
ULONG        reserved;
 } QMSG;

typedef QMSG *PQMSG;
```

Fields

hwnd (HWND)
Window handle.

msg (ULONG)
Message identity.

mp1 (MPARAM)
Parameter 1.

mp2 (MPARAM)
Parameter 2.

time (ULONG)
Message time.

ptl (POINTL)
Pointer position when message was generated.

reserved (ULONG)
Reserved.

SMHSTRUCT

Send-message-hook structure.

Syntax

```
typedef struct _SMHSTRUCT {
MPARAM       mp2;
MPARAM       mp1;
ULONG        msg;
HWND         hwnd;
ULONG        model;
 } SMHSTRUCT;

typedef SMHSTRUCT *PSMHSTRUCT;
```

Fields

mp2 (MPARAM)
    Parameter 2.

mp1 (MPARAM)
    Parameter 1.

msg (ULONG)
    Message identity.

hwnd (HWND)
    Window handle.

model (ULONG)
    Message identity.

SWP

Set-window-position structure.

Syntax

```
typedef struct _SWP {
ULONG      fl;
LONG       cy;
LONG       cx;
LONG       y;
LONG       x;
HWND       hwndInsertBehind;
HWND       hwnd;
ULONG      ulReserved1;
ULONG      ulReserved2;
 } SWP;

typedef SWP *PSWP;
```

Fields

fl (ULONG)
Options.

In alphabetic order:

SWP_ACTIVATE
SWP_DEACTIVATE
SWP_HIDE
SWP_MAXIMIZE
SWP_MINIMIZE
SWP_MOVE
SWP_NOADJUST
SWP_NOERASEWINDOW
SWP_NOREDRAW
SWP_RESTORE
SWP_SHOW
SWP_SIZE
SWP_ZORDER

cy (LONG)
Window height.

cx (LONG)
Window width.

y (LONG)
Y-coordinate of origin.

x (LONG)
X-coordinate of origin.

hwndInsertBehind (HWND)
Window behind which this window is placed.

hwnd (HWND)
Window handle.

ulReserved1 (ULONG)
Reserved value, must be 0.

ulReserved2 (ULONG)
Reserved value, must be 0.

TRACKINFO

Tracking-information structure.

Syntax

```
typedef struct _TRACKINFO {
LONG         cxBorder;
LONG         cyBorder;
LONG         cxGrid;
LONG         cyGrid;
LONG         cxKeyboard;
LONG         cyKeyboard;
RECTL        rclTrack;
RECTL        rclBoundary;
POINTL       ptlMinTrackSize;
POINTL       ptlMaxTrackSize;
ULONG        fs;
 } TRACKINFO;

typedef TRACKINFO *PTRACKINFO;
```

Fields

cxBorder (LONG)
Border width.

The width of the left and right tracking sides.

cyBorder (LONG)
Border height.

The height of the top and bottom tracking sides.

cxGrid (LONG)
Grid width.

The horizontal bounds of the tracking movements.

cyGrid (LONG)
Grid height.

The vertical bounds of the tracking movements.

cxKeyboard (LONG)
Character cell width movement for arrow key.

cyKeyboard (LONG)
Character cell height movement for arrow key.

rclTrack (RECTL)
Starting tracking rectangle.

This is modified as the rectangle is tracked and holds the new tracking position, when tracking is complete.

rclBoundary (RECTL)
Boundary rectangle.

This is an absolute bounding rectangle that the tracking rectangle cannot extend; see also TF_ALLINBOUNDARY.

ptlMinTrackSize (POINTL)
Minimum tracking size.

ptlMaxTrackSize (POINTL)
Maximum tracking size.

fs (ULONG)
Tracking options.

In alphabetic order:

TF_ALLINBOUNDARY
The default tracking is such that some part of the tracking rectangle is within the bounding rectangle defined by *rclBoundary*. This minimum size is defined by *cxBorder* and *cyBorder*.

If TF_ALLINBOUNDARY is specified, the tracking is performed so that no part of the tracking rectangle ever falls outside of the bounding rectangle.

TF_BOTTOM
Track the bottom side of the rectangle.

TF_GRID
Tracking is restricted to the grid defined by *cxGrid* and *cyGrid*.

TF_LEFT
Track the left side of the rectangle.

TF_MOVE
Track all sides of the rectangle.

TF_RIGHT
Track the right side of the rectangle.

TF_SETPOINTERPOS
The pointer is repositioned according to other flags as follows:

none	Pointer is centered in the tracking rectangle.
TF_MOVE	Pointer is centered in the tracking rectangle.
TF_LEFT	Pointer is vertically centered at the left of the tracking rectangle.
TF_TOP	Pointer is horizontally centered at the top of the tracking rectangle.
TF_RIGHT	Pointer is vertically centered at the right of the tracking rectangle.
TF_BOTTOM	Pointer is horizontally centered at the bottom of the tracking rectangle.

TF_STANDARD
cx, *cy*, *cxGrid*, and *cyGrid* are all multiples of *cxBorder* and *cyBorder*.

TF_TOP
Track the top side of the rectangle.

Summary

Following are the OS/2 functions and data structures used with hook controls:

Table 13-3. Hook Functions

Function Name	Description
MsgFilterHook	Filters messages from inside a mode loop.
RegisterUserHook	Called whenever a user message or data structure is registered.
WinCallMsgFilter	Calls a message-filter hook.
WindowDCHook	Called when a device context is allocated or freed.
WinReleaseHook	Releases an application hook from a hook chain.
WinSetHook	Installs an application procedure in a specified hook chain.
WinTrackRect	Draws a tracking rectangle.

Table 13-4. Hook Data Structures

Data Structure Name	Description
QMSG	Message data structure.
SMHSTRUCT	Send-message-hook data structure.
SWP	Set-window-position data structure.
TRACKINFO	Tracking-information data structure.

Chapter 14. Dynamic Data Exchange

The Dynamic Data Exchange (DDE) protocol uses messages to communicate between applications that share data and uses shared memory as the means of exchanging data between applications. Applications can use DDE for one-time data transfers and for ongoing exchanges in which the applications send updates to each other as new data becomes available. This chapter explains how to use DDE in PM applications.

About Dynamic Data Exchange

DDE is different from the clipboard data-transfer component that is also part of this operating system. The clipboard is almost always used as a one-time response to a specific action by the user, such as choosing **Paste** from a menu. DDE, on the other hand, is often initiated by a user but typically continues without the user's further involvement. DDE is separate from and does not use the clipboard.

DDE always takes place between two applications: a *client* application and a *server* application. The client initiates the exchange by requesting that the server perform a particular action, such as supply data. The client's request to the server is called a *transaction*. If it is able, the server responds by performing the requested action. The important distinction between a client and a server is that the client always initiates DDE transactions.

A server can have many clients simultaneously, and a client can request data from multiple servers. An application can be both a client and a server at the same time. For example, one application could receive data from another application as a client and then act as a server by passing the data to yet another application.

Client and Server Interaction

A DDE conversation actually takes place between two windows: one for each of the participating applications. Applications open a window for each conversation in which they engage. Because a window is identified by its handle, these windows are not necessarily visible. The window belonging to the server application is the *server window*. The window belonging to the client application is the *client window*.

DDE System Example

DDE has many potential uses in real-time data acquisition applications. Consider the example of a DDE-based, real-time system for tracking portfolios. Two hypothetical PM applications cooperate in this example. One application, named the *collector*, is a specialized interface that draws data from an online data service. The other application is a spreadsheet. Both applications use the DDE protocol. Figure 14-1 on page 14-2 shows the sample spreadsheet layout.

	A	B	C	D
1	**Stock**	**Shares**	**Price**	**Extension**
2	ABCD	1000	148	148000
3	EFGH	2000	26	52000
4	IJKL	200	24	4800
5	MNOP	2000	93	186000
6				390800

Figure 14-1. Sample Spreadsheet Layout

Without DDE, this spreadsheet could be updated by using the clipboard to manually copy numbers from the screen display of the collector application to the spreadsheet. This would require screen sharing or switching between applications. The user also would have to pay close attention to the price data, then undertake the data exchange personally whenever the price data changes.

With DDE, this system could be much more automatic, providing the spreadsheet with the current values for multiple data items, without user intervention. DDE enables the user to set up an exchange between the two applications that updates the spreadsheet whenever a change occurs in the value of specified stocks. After this connection is established, the cell values in the spreadsheet always reflect the most current data available from the collector. This system facilitates the timely analysis of real-time data.

The usefulness of the DDE protocol is not restricted to specialized real-time data-acquisition applications. Productivity software, in general, can benefit significantly from the protocol. For example, a monthly report is prepared using word processor, and the report includes graphs generated in a separate business-graphics package. Without DDE, someone must manually copy and paste each month's new graphs into each month's report. With DDE, the word processor can establish a permanent link to the graphics application so that any changes made to the graphs are reflected in the word-processing document, either automatically or on request.

Figure 14-2 on page 14-3 shows a detailed view of the workings of the DDE protocol and describes the collector and spreadsheet interaction and illustrates the forwarding of stock quotes from the collector application to the spreadsheet. For simplicity, this example is limited to the exchange of quotes for a single stock, ABCD.

Spreadsheet (Client)		Collector (Server)
		Start application
Start application		
Load stock-portfolio document		
Calls WinDdeInitiate which sends WM_DDE_INITIATE	→	
	←	Accepts and calls WinDdeRespond with positive WM_DDE_INITIATEACK
Calls WinDdePostMsg which posts WM_ADVISE	→	
Request to send info each time data item ABCD changes; send in format DDEFMT_TEXT		
	←	Records request in database and calls WinDdePostMsg which posts WM_DDE_ACK
	←	Whenever data changes for item ABCD, calls WinDdePostMsg which posts WM_DDE_DATA
Retrieves information from shared-memory object indicated by pointer in WM_DDE_DATA		
Updates value of ABCD in spreadsheet		

Figure 14-2 (Part 1 of 2). Detailed DDE Example

```
When ready to end updates,                .
calls WinDdePostMsg which posts           .
WM_DDE_UNADVISE ------------------------->.
                                          .
                                          .          Removes request from
                                          .             database and calls
                                          .       WinDdePostMsg which posts
                                          .<------------------- WM_DDE_ACK
                                          .
To end DDE transaction,                   .
calls WinDdePostMsg which posts           .
WM_DDE_TERMINATE ------------------------>.
                                          .
                                          .             Responds by calling
                                          .       WinDdePostMsg which posts
                                          .<------------- WM_DDE_TERMINATE
```

Figure 14-2 (Part 2 of 2). Detailed DDE Example

The collector DDE server application is started first. Typically, applications designed to operate as dedicated DDE servers have a user interface for initialization, and then run minimized. As part of the initialization process, the collector DDE server application performs the necessary tasks (such as entering passwords and testing) to ensure that it can provide data to clients.

The spreadsheet is started next, and the stock-portfolio document is loaded. At this time, the spreadsheet calls WinDdeInitiate, which sends a WM_DDE_INITIATE message to all top-level frame windows, that is, frame windows that have HWND_DESKTOP as their parent. The WM_DDE_INITIATE message is a request to initiate an exchange with an application on a specified topic—in this case, STOCKS. An application can accept this message by responding with a positive WM_DDE_INITIATEACK message, or decline the message by passing the message on to WinDefWindowProc. If no application accepts the request, the spreadsheet assigns an error value to the external reference and its DDE activity concludes.

If the collector acknowledges the request, the spreadsheet can use the newly established exchange to request that the collector provide continuous updates on a specified data item. To make this request, the spreadsheet posts a WM_DDE_ADVISE message to the collector (actually, to a window within the collector that acts as the message recipient for DDE messages), indicating that updates must be sent every time there is a new value available for the data item, ABCD, and that the updates should be in a particular format—for example, DDEFMT_TEXT.

Upon receiving this message, the collector application records the request in its database and posts a WM_DDE_ACK message to the spreadsheet. From then on, whenever the collector receives a new ABCD stock quote, it posts a WM_DDE_DATA message to the window in the spreadsheet that initiated the exchange. Each of these messages carries a pointer to a shared-memory object that contains the data, rendered in the requested format. When the spreadsheet receives such a message, it retrieves the data from the referenced memory object and uses the data to update the value of the cell containing the external reference.

The periodic updates continue until the spreadsheet document is closed. At that point, the spreadsheet application posts a WM_DDE_UNADVISE message to the collector application, indicating that further updating is unnecessary. Upon receipt of this message, the collector application removes the corresponding data request from its database and posts a positive WM_DDE_ACK message back to the spreadsheet.

Finally, unless the spreadsheet initiates other data exchanges under this same topic, it posts a WM_DDE_TERMINATE message to the collector application, indicating the end of the DDE transaction. The collector application responds with a WM_DDE_TERMINATE message.

Note: At any time during the transaction, both the spreadsheet and collector are free to post a WM_DDE_TERMINATE message to the other application.

Applications, Topics, and Items

DDE uses the three-level hierarchy—*application, topic*, and *item*—to uniquely identify a unit of data. An *application* is the name of the server from which the data is desired. A *topic* is a logical data context. For applications that operate on file-based documents, topics are usually file names; for other applications, they are other application-specific strings. An *item* is a data object that can be passed in a DDE transaction. For example, an item might be a single integer, a string, several paragraphs of text, or a bit map. In the collector and spreadsheet model described in the previous section, the application name is *collector*, the topic name is *STOCKS*, and the item name is *ABCD*.

The System Topic

The *system topic* provides a context for information that might be of general interest to any partners in a DDE transaction. Server applications are encouraged to support the system topic at all times. The string used for the system topic is defined in the PM header files as SZDDESYS_TOPIC.

DDE applications should request an exchange on the system topic with a zero-length application name when they start up, to find out what kinds of information other DDE-capable programs can provide.

The system topic must support the items in Table 14-1 as well as any other items the application uses.

Table 14-1 (Page 1 of 2). DDE System Topics

Item	Description
SZDDESYS_ITEM_FORMATS	A list of strings equivalent to CF_CONSTANTS with the CF_ prefix removed. For example, CF_TEXT = TEXT.
SZDDESYS_ITEM_HELP	A text description of the server's DDE services.

Table 14-1 (Page 2 of 2). DDE System Topics	
Item	**Description**
SZDDESYS_ITEM_PROTOCOLS	A list of protocol names the server supports. A protocol is a set of DDE execute commands, each having a standard meaning.
SZDDESYS_ITEM_RESTART	A string that a client can pass to DosExecPgm to invoke a server that is not running.
SZDDESYS_ITEM_RTNMSG	Supporting detail for the most recently issued WM_DDE_ACK message. This is useful when more than 8 bits of application-specific return code are required.
SZDDESYS_ITEM_SECURITY	A security-sensitive server application. Any client can initiate a conversation with a security-sensitive server, but the server responds only to the Security topic. Typically, the server requires a password from the client before any further data exchange can take place.
SZDDESYS_ITEM_STATUS	An indication of the current status of the server: "Ready" or "Busy."
SZDDESYS_ITEM_SYSITEMS	A list of the items supported under the system topic by this server.
SZDDESYS_ITEM_TOPICS	A list of the topics currently supported by the application. This can vary from moment to moment.

Individual elements of lists should be delimited by tabs, as in the DDEFMT_TEXT format.

DDE Initiation

A client application initiates a DDE conversation by calling WinDdeInitiate, specifying the server application-name string and the topic-name string. WinDdeInitiate fills a DDEINIT data structure with the specified strings, then sends a WM_DDE_INITIATE message to all frame windows that have HWND_DESKTOP as their parent. The message contains the handle of the client application and a pointer to the DDEINIT data structure. Figure 14-3 illustrates the DDEINIT data structure.

```
typedef struct _DDEINIT
{
  ULONG   cb;
  PSZ     pszAppName;
  PSZ     pszTopic;
  USHORT  usConvContext;
 } DDEINIT;
```

Figure 14-3. DDEINIT Data Structure.

Because the message is sent rather than posted, WinDdeInitiate requires a response from all recipients of the message before it returns control to the client application. Figure 14-4 on page 14-7 illustrates the initial flow of a DDE conversation.

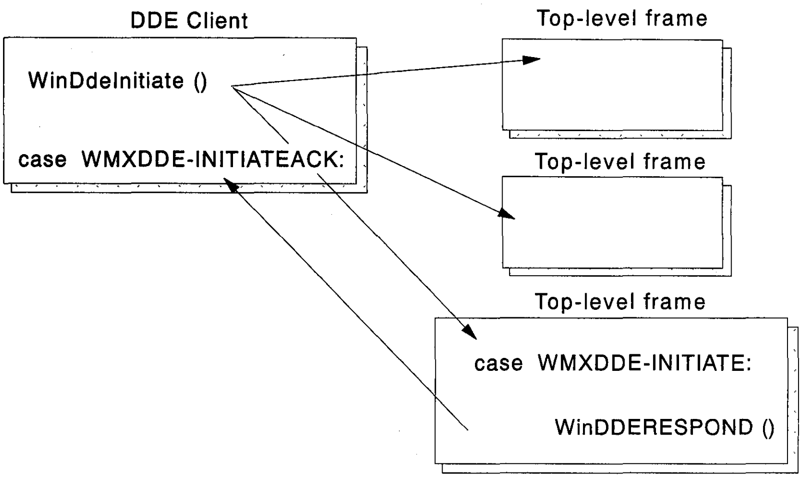

Figure 14-4. Initiating a DDE Conversation

Any potential server must contain a *server window*, a top-level frame window that has been subclassed to receive and process WM_DDE_INITIATE messages. When a server window receives WM_DDE_INITIATE, it examines the application-name and topic-name strings in the DDEINIT data structure. If the application-name string matches and the server supports the requested topic, the server acknowledges the client's request.

Either the application-name or topic-name string can be zero-length. If the application-name string is zero-length, all servers check the topic-name string. Each server that supports the topic sends a separate acknowledgment to the client. If the topic-name string is zero-length, the server sends an acknowledgment for each supported topic. Using zero-length strings, a client can obtain the names of all the active servers in the system or the names of all the topics a server supports.

Figure 14-5 shows how servers respond to WM_DDE_INITIATE messages.

```
If (specific app requested and server is instance of app) or
   (specific app not requested)
{

     If (specific topic requested)
          If (server can support topic)

               acknowledge the requested topic

     else
          acknowledge each supported topic
}
```

Figure 14-5. How Servers Respond to DDE_INITIATE Messages

A server acknowledges its support of a specific topic by calling WinDdeRespond, specifying the handle of its server window, its application name, and the name of the supported topic. WinDdeRespond fills a DDEINIT data structure with the specified strings, then sends a WM_DDE_INITIATEACK message to the client. The message contains the handle of the server window and a pointer to the DDEINIT data structure. The client examines the topic-name string in the DDEINIT data structure and decides whether to begin a transaction on the topic.

If two applications agree on some unspecified protocol and can exchange window handles by some means, they can use DDE messages on those window handles without going through an initiate sequence.

An application does not need to fill in a DDEINIT data structure; WinDdeInitiate and WinDdeRespond automatically fill the data structure. However, applications must extract the application name and topic name from the DDEINIT data structure when receiving a WM_DDE_INITIATE or WM_DDE_INITIATEACK message.

Shared-Memory Object

After initiating a conversation, the client interacts with the server by issuing transactions. A *transaction* is a client's request that the server perform a particular action.

To issue a transaction, the client allocates a shared-memory object, writes data about its request to the object using a DDESTRUCT data structure, then calls WinDdePostMsg to post a transaction message to the server. The transaction message contains the client-window handle and a pointer to the shared-memory object. When the server receives the message, it uses the pointer to access the shared-memory object.

The server responds by allocating a shared-memory object, writing its response to the object using a DDESTRUCT data structure, then calling WinDdePostMsg to post a response message to the client. The response message contains the server-window handle and a pointer to the shared-memory object.

A DDESTRUCT data structure occupies the first part of the memory object. Next comes the item-name string, followed by the actual data being exchanged. The offset fields of the DDESTRUCT data structure must be set to point to the name string and the beginning of the data. The *cbData* field also must be set to indicate the number of bytes of data.

The sender of a DDE transaction message must allocate a shared-memory object using DosAllocSharedMem, then call DosGiveSharedMem to share the object with the receiving application. To share an object, the sender must know the process identifier of the recipient. The process identifier can be obtained by calling WinQueryWindowProcess for the recipient's window handle. WinDdePostMsg also gives the memory object.

The sender should not try to access the object after sending it to the recipient in a DDE message. After posting a transaction message, WinDdePostMsg automatically frees the shared-memory object from the sender's virtual address space. An application need not call DosFreeMem for this purpose. However, the recipient must call DosFreeMem when it is finished using the object.

Transaction Status Flags

DDE client and server applications can specify status flags in the DDESTRUCT data structure. These flags are constant values that applications use to control various aspects of a DDE transaction. They can be combined in the *fsStatus* word of the DDESTRUCT data structure by using the OR operator. Table 14-2 lists the DDE status flags.

Table 14-2. DDE Status Flags

Flag Name	Description
DDE_FACK	Indicates a positive acknowledgment.
DDE_FACKREQ	Requests an acknowledgment from the receiving application.
DDE_FAPPSTATUS	Indicates that the upper 8 bits of the status word are used for application-specific data.
DDE_FBUSY	Indicates that the application received a request but cannot respond because it is busy filling an earlier request.
DDE_FNODATA	Indicates that no data is to be transferred in response to the WM_DDE_ADVISE message.
DDE_FRESERVED	Reserved; must be 0.
DDE_FRESPONSE	Indicates a response to a WM_DDE_REQUEST message.
DDE_NOTPROCESSED	Indicates that the message received is not supported.

Transaction and Response Messages

DDE applications use WinDdePostMsg to communicate during data-exchange transactions. A client application posts transaction messages to a server, which responds by posting acknowledgment messages to the client. Transaction and acknowledgment messages have the same data structure. The first message parameter contains the handle of the sending window; the second contains a pointer to the shared-memory object that contains message information.

The DDE protocol defines five transaction types:

- Advise
- Unadvise
- Request
- Poke
- Execute.

These transactions are permitted only within an exchange begun by using the WM_DDE_INITIATE message. Each transaction type has a corresponding message that a client uses to initiate the transaction with a server:

- WM_DDE_ADVISE
- WM_DDE_UNADVISE
- WM_DDE_REQUEST
- WM_DDE_POKE
- WM_DDE_EXECUTE.

A server acknowledges a transaction message by posting a WM_DDE_ACK message to the client. The client must examine the status field of the DDESTRUCT data structure to determine whether the response is positive or negative.

A server application posts a WM_DDE_DATA message to the client to indicate that requested data is available. If the status bit of the DDESTRUCT structure has the DDE_FACKREQ flag set, the client must acknowledge receipt of the data by sending a WM_DDE_ACK message to the server.

The fifth parameter of WinDdePostMsg is a flag used to specify whether to try to post a message again if the first attempt failed because the destination queue was full (server returns the DDE_FBUSY flag). If the retry flag is set, WinDdePostMsg posts the message at 1-second intervals until the message is posted successfully.

The following sections explain the five basic types of DDE transactions and the messages involved with each. These messages are posted with WinDdePostMsg, which automatically builds and fills a DDEINIT data structure.

Request and Poke Transactions

A client application can use the DDE protocol to obtain a data item from a server (WM_DDE_REQUEST) or to submit a data item to a server (WM_DDE_POKE).

The client posts a WM_DDE_REQUEST message to the server, specifying an item and format by allocating a shared-memory object, filling in a DDESTRUCT data structure, and passing the data structure to WinDdePostMsg.

If the server is unable to satisfy the request, it sends the client a negative WM_DDE_ACK message. If the server can satisfy the request, it renders the item in the requested format, includes it with a DDESTRUCT data structure in a shared-memory object, and posts a WM_DDE_DATA message to the client.

Upon receiving a WM_DDE_DATA message, the client processes the data item. At the beginning of the shared-memory object, the DDESTRUCT data structure contains a status word indicating whether the sender requested an acknowledgment message. If the DDE_FACKREQ bit of the status word is set, the client must send the server a positive WM_DDE_ACK message.

Upon receiving a negative WM_DDE_ACK message, the client can ask for the same item again, specifying a different DDE format. Typically, a client first asks for the most complex format it can support, then steps down, if necessary, through progressively simpler formats, until it finds one the server can provide.

Advise and Unadvise Transactions

A client application can use DDE to establish a link to an item in a server application. When such a link is established, the server sends periodic updates about the linked item to the client (typically, whenever the data associated with the item in the server application has changed). A permanent *data stream* is established between the two applications and remains in place until it is explicitly disconnected.

The client sends the server a WM_DDE_ADVISE message to set up the data link. The advise message contains a shared-memory pointer containing a DDESTRUCT data structure with the item name, format information, and status information.

If the server has access to the requested item and can render it in the desired format, the server records the new link, then sends the client a positive WM_DDE_ACK message. Until the client issues a WM_DDE_UNADVISE message, the server sends data messages to the client every time a change occurs in the source data associated with the item in the server application.

If the server is unable to satisfy the request, it sends the client a negative WM_DDE_ACK message.

When a link is established with the DDE_FNODATA status bit cleared, the client is sent the data each time the data changes. In such cases, the server renders the new version of the item in the previously specified format and posts a WM_DDE_DATA message to the client.

When the client receives a WM_DDE_DATA message, it extracts data from the shared-memory object by using the DDESTRUCT data structure at the beginning of the object. If the DDE_FACKREQ status bit in the status word of the DDESTRUCT data structure is set, the client must post a positive WM_DDE_ACK message to the server.

When a link is established with the DDE_FNODATA status flag set, a notification, not the data itself, is posted to the client each time the data changes. In this case, the server does not render the new version of the item when the source data changes, but simply posts a WM_DDE_DATA message with 0 bytes of data and the DDE_FNODATA status flag set.

The client can request the latest version of the data by performing a regular one-time WM_DDE_REQUEST transaction, or it can simply ignore the data-change notice from the server. In either case, if the DDE_FACKREQ status bit is set, the client should send a positive WM_DDE_ACK message to the server.

When a client sends a WM_DDE_ADVISE message on a topic/item pair that is already engaged in an advise loop but has a different format specified, the server interprets this as a request to add an advise loop with the given format requested. Therefore, several advise loops can exist for a given topic/item pair. If a server does not support this extent of advise loops, it rejects the advise request.

Correspondingly, when a server receives a WM_DDE_UNADVISE message, the server must compare the format field with the current format of the advise loop. Only if the specified format is 0, meaning all advise loops, or matches an active advise loop does the server stop the advise loop and return a positive acknowledgment.

To terminate a specific item link, the client posts a WM_DDE_UNADVISE message to the server. The server ensures that the client currently has a link to the specified item in this exchange. If the link exists, the server sends a positive WM_DDE_ACK message to the client and no longer sends updates on the item in this exchange. If the server has no such link, it sends a negative WM_DDE_ACK message.

To terminate all links for a particular exchange, the client application posts a WM_DDE_UNADVISE message with a zero-length item name to the server. The server ensures that the exchange has at least one link currently established. If so, the server posts a positive WM_DDE_ACK message to the client, and no longer sends any updates in the exchange. If the server has no links in the exchange, it posts a negative WM_DDE_ACK message.

Execute Transaction

A PM application can use the DDE protocol to cause commands to be executed in another application. Such remote executions are performed by the WM_DDE_EXECUTE transaction.

To execute a remote command, the client application posts to the server a WM_DDE_EXECUTE message containing a pointer to a shared-memory object that contains a DDESTRUCT data structure and a command string.

The server attempts to execute the specified string according to some agreed-upon protocol. If successful, the server posts a positive WM_DDE_ACK message to the client. If unsuccessful, a negative WM_DDE_ACK message is posted.

DDE Termination

At any time, either the client or the server may terminate an exchange by issuing a WM_DDE_TERMINATE message. Similarly, both the client application and server application must be able to receive a WM_DDE_TERMINATE message at any time.

An application must end its exchanges before terminating. The application posts a WM_DDE_TERMINATE message with a zero-length shared-memory pointer. A WM_DDE_TERMINATE message stops all transactions for a given exchange.

The WM_DDE_TERMINATE message means that the sender sends no further messages in that exchange and that the recipient can destroy its DDE window. The recipient must always send a WM_DDE_TERMINATE message promptly in response; it is not permissible to send a negative, busy, or positive WM_DDE_ACK message instead.

If the original sender of the termination request receives any other message before the WM_DDE_TERMINATE message arrives from the recipient of the request, it should not respond, because the sender of the other message might have already destroyed the window to which the response would be sent.

Unique Data Formats

Whenever an application exchanges data using the DDE protocol, it must specify the format of the data in the *usFormat* field of the DDESTRUCT data structure. The system-defined standard format for exchanging text data is DDEFMT_TEXT. Applications can also use constant names to specify the format of data to be exchanged listed in Table 14-3 on page 14-13.

Table 14-3. DDE Data Formats

Data Format Name	Description
SZFMT_BITMAP	Specifies that the data is a bit map.
SZFMT_CPTEXT	Specifies text whose format is defined by a CPTEXT data structure. Applications can use this format to pass multiple-language strings without changing the conversation context.
SZFMT_DIF	Specifies that the data is in Data Image Format (DIF).
SZFMT_DSPBITMAP	Specifies that the data is a bit-map representation of a private data format.
SZFMT_DSPMETAFILE	Specifies that the data is a metafile representation of a private data format.
SZFMT_DSPMETAFILEPICT	Specifies that the data is a metafile picture representation of a private data format.
SZFMT_DSPTEXT	Specifies that the data is a text representation of a private data format.
SZFMT_LINK	Specifies that the data is in link-file format.
SZFMT_METAFILE	Specifies that the data is a metafile.
SZFMT_METAFILEPICT	Specifies that the data is a metafile picture defined by an MFP data structure.
SZFMT_OEMTEXT	Specifies that the data is in OEM Text format.
SZFMT_PALETTE	Specifies that the data is in palette format.
SZFMT_SYLK	Specifies that the data is in Synchronous Link format.
SZFMT_TEXT	Specifies that the data is an array of text characters. These characters can include new-line characters to indicate linebreaks. The zero-length character indicates the end of the text data.
SZFMT_TIFF	Specifies that the data is in Tag Image File Format (TIFF).

Applications can define their own data formats. However, each nonstandard DDE format must have a unique identification number. To receive an identification number for a nonstardard format, the application must register the name of the format in the system atom table. Other applications that have the name of the format can then query the system atom table for the format's identification number. This method ensures that all applications use the same atom to identify a format.

Synchronization Rules

A window processing DDE requests from another window must process them strictly in the order in which the requests were received.

A window does not need to apply this first-in first-out (FIFO) rule between requests from different windows—that is, it may provide asynchronous support for multiple processes. For example, a window might have the following requests in its queue:

1. Request message from window x
2. Request message from window y
3. Request message from window x.

The window must process request message 1 before request message 3, but it does not have to process request message 2 before request message 3. If y has a lower priority than x, the window follows the order 1, 3, 2.

If a server is unable to process an incoming request because it is waiting for an external process, it must post a busy WM_DDE_ACK message to the client, to prevent deadlock. A busy WM_DDE_ACK message can also be sent if the server is unable to process an incoming request quickly.

Language-Sensitive DDE Applications

DDE applications written for the international market must be able to exchange data in several different languages. The CONVCONTEXT data structure, along with WinDdeInitiate and WinDdeRespond, provide this support.

A language-sensitive DDE application defines the context of a conversation by filling a CONVCONTEXT data structure with the appropriate country code and code-page identifiers. The CONVCONTEXT data structure also contains a context flag. If this flag is set to DDECTXT_CASESENSITIVE, applications must compare strings in a case-sensitive manner. Language-sensitive DDE applications use WinDdeInitiate and WinDdeRespond to establish a DDE conversation. These functions pass a pointer to a CONVCONTEXT data structure.

Using Dynamic Data Exchange

This section explains how to perform the following tasks:

- Initiate a DDE conversation
- Create a shared-memory object for DDE
- Send positive acknowledgment messages
- Send negative acknowledgment messages
- Perform a one-time data transfer
- Establish a permanent data link
- Execute commands in a remote application
- Terminate a DDE conversation.

Note: Much of the sample codes in this section are part of a complete program for either a client application or a server application. Both programs are illustrated in "Sample Code for Dynamic Data Exchange" on page 14-22.

Initiating a DDE Conversation

The client application initiates a DDE conversation by calling WinDdeInitiate, specifying the server application-name string and the topic-name string.

The client application in "Sample Code for Dynamic Data Exchange" on page 14-22 allows the user to initiate a DDE conversation from a context menu. The sample code illustrated in Figure 14-6 shows how the client application processes that request.

```
/* User starts DDE conversation */
case IDM_POLL:
   WinPostMsg(hListWnd, LM_DELETEALL, 0, 0);
   ShowMessage("Polling...");
   context.cb = sizeof(CONVCONTEXT);
   context.fsContext = 0;
   WinDdeInitiate(hwnd, szApp, szTopic, &context);
   ShowMessage("Polling complete.");
   break;
```

Figure 14-6. How the Client Application Respond to a DDE Conversation

The sample code illustrated in Figure 14-7 shows how the server application determines whether to send a positive or negative acknowledgment to the WinDdeInitiate call.

```
/*******************************************************************/
/*  Check incoming poll - if the App and Topic match,             */
/*  we must acknowledge. If both are zero-length, the client is    */
/*  searching for anyone to talk to - send our names               */
/*******************************************************************/
szClientApp   = pDDEinit->pszAppName;
szClientTopic = pDDEinit->pszTopic;
ShowMessage(szClientApp);
ShowMessage(szClientTopic);
```

Figure 14-7 (Part 1 of 2). How the Server Determines the Acknowledgment to Send

```
if (!strcmpi(szClientApp, szApp) ||
    !strcmpi(szClientApp, NULL))
{
    if (!strcmpi(szClientTopic, szTopic) ||
        !strcmpi(szClientTopic, NULL))
    {
        context.cb = sizeof(CONVCONTEXT);
        context.fsContext = 0;
        WinDdeRespond(hClientWnd,
                      hwnd,
                      szApp,
                      szTopic,
                      &context);
    }
}
break;
```

Figure 14-7 (Part 2 of 2). How the Server Determines the Acknowledgment to Send

Creating a Shared-Memory Object for DDE

The sample code illustrated in Figure 14-8 shows how to create a shared-memory object for a DDE transaction. The parameters include the destination window for the DDE message, item name for the transaction, status word, format of the data, actual data to be transferred (if any), and the length of the data. The allocated object must be big enough to hold the DDESTRUCT data structure, item name, and the actual data to be transferred. The sample returns a pointer (PDDESTRUCT) to a shared-memory object that is ready to post as part of a DDE message.

```
/*  Get some sharable memory */
DosAllocSharedMem((PVOID)&mem,
                  NULL,
                  sizeof(DDESTRUCT)+21,
                  PAG_COMMIT |
                  PAG_READ |
                  PAG_WRITE |
                  OBJ_GIVEABLE);
```

Figure 14-8 (Part 1 of 2). How to Create a Shared-Memory Object

```
/*  Get the server's ID and give it access to the */
/*  shared memory                                 */
WinQueryWindowProcess(hServerWnd, &pid, &tid);
DosGiveSharedMem(&mem, pid, PAG_READ | PAG_WRITE);

/*  Setup DDE data structures                                      */
/*  (11 byte name length, 10 plus NULL, 10 byte data length) */
pDDEdata           = (PDDESTRUCT)mem;
pDDEdata->cbData   = 10;              /* Data length          */
pDDEdata->fsStatus = 0;               /* Status               */
pDDEdata->usFormat = DDEFMT_TEXT;     /* Text format          */

/*  Go past end of data structure for the name  */
pDDEdata->offszItemName = sizeof(DDESTRUCT);

/*  Go past end of structure (plus past the name)  */
/*  for the data                                   */
pDDEdata->offabData = sizeof(DDESTRUCT)+11;
strcpy((BYTE *)(pDDEdata+(pDDEdata->offszItemName)),
                "STATUS");

/* Post our request to the server program */
WinDdePostMsg(hServerWnd,
              hwnd,
              WM_DDE_REQUEST,
              pDDEdata,
              DDEPM_RETRY);
```

Figure 14-8 (Part 2 of 2). How to Create a Shared-Memory Object

Sending a Positive Acknowledgment

You can send a positive acknowledgment by posting a WM_DDE_ACK message with the DDE_FACK and DDE_FRESPONSE flags set in the status word of the shared-memory data structure. The sample code illustrated in Figure 14-9 on page 14-18 shows how to do so.

```
/*  Specify the status flags, when allocating shared memory */
pDDEdata->fstatus = DDE_FACK | DDE_FRESPONSE;
    .
    .
    .

/*  Post the message */
WinDdePostMsg(hwndDest,             /* Handle of destination  */
              hwndSource,           /* Handle of source       */
              WM_DDE_ACK,           /* Message                */
              pDDEdata,             /* Shared-memory pointer  */
              DDEPM_RETRY);         /* Retry                  */
```

Figure 14-9. How to Send a Positive Acknowledgment

Sending a Negative Acknowledgment

You can send a negative acknowledgment by posting a WM_DDE_ACK message with the DDE_NOTPROCESSED flag set in the status word of the shared-memory data structure. By not specifying DDE_FACK, it is legal to specify DDE_NOTPROCESSED, but only if the message is not supported, such as WM_DDE_POKE for the specified item. DDE_NOTPROCESSED is not the negative respond. The sample code illustrated in Figure 14-10 shows how to do so.

```
/*  Specify the status flag, when allocating shared memory */
pDDEdata->fstatus &= 0;
    .
    .
    .

/*  Post the message */
WinDdePostMsg(hwndDest,             /* Handle of destination  */
              hwndSource,           /* Handle of source       */
              WM_DDE_ACK,           /* Message                */
              pDDEdata,             /* Shared-memory pointer  */
              DDEPM_RETRY);         /* Retry                  */
```

Figure 14-10. How to Send a Negative Acknowledgment

If an application is busy when it receives a DDE message, it can post a WM_DDE_ACK message with the DDE_FBUSY flag set.

Performing a One-Time Data Transfer

A client application posts a WM_DDE_REQUEST or WM_DDE_POKE message to perform a one-time data transfer with a server application. The item-name portion of the shared-memory object passed with the message contains the name of the desired item.

When the client posts a WM_DDE_POKE message, the data portion of the shared-memory object contains the data being sent to the server.

If the server can satisfy the request, it renders the item in the requested format and includes it, with a DDESTRUCT data structure, in a shared-memory object and posts a WM_DDE_DATA message to the client, as shown the sample code illustrated in Figure 14-11.

```
/* The DDE data structure is passed, and    */
/* the client should have shared it with us */
pDDEdata  = (PDDESTRUCT)mp2;
szReqItem = (BYTE *)(pDDEdata+(pDDEdata->offszItemName));
ShowMessage(szReqItem);

/*  We support item status, but not anything else */
if (!strcmpi(szReqItem, szItem))
   {
     ShowMessage("sending...");

      /* Get some sharable memory */
      DosAllocSharedMem((PVOID)&mem,
                        NULL,
                        sizeof(DDESTRUCT)+21,
                        PAG_COMMIT |
                        PAG_READ |
                        PAG_WRITE |
                        OBJ_GIVEABLE);

      /*  Get the server's id and give it access to the */
      /*  shared memory                                 */
      WinQueryWindowProcess(hClientWnd, &pid, &tid);
      DosGiveSharedMem(&mem, pid, PAG_READ | PAG_WRITE);

      /*  Setup DDE data structures                                    */
      /*  (11 byte name length, 10 plus NULL, 10 byte data length)    */
      pDDEdata           = (PDDESTRUCT)mem;
      pDDEdata->cbData   = 10;               /* Data length            */
      pDDEdata->fsStatus = 0;                /* Status                 */
      pDDEdata->usFormat = DDEFMT_TEXT;      /* Text format            */
```

Figure 14-11 (Part 1 of 2). How to Perform a One-Time Data Transfer

```
        /*  Go past end of structure for the name */
        pDDEdata->offszItemName = sizeof(DDESTRUCT);

        /*  Go past end of structure (and name) for the data  */
        pDDEdata->offabData = sizeof(DDESTRUCT)+11;
        strcpy((BYTE *)(pDDEdata+(pDDEdata->offabData)), szStatus);
        WinDdePostMsg(hClientWnd,
                      hwnd,
                      WM_DDE_DATA,
                      pDDEdata,
                      DDEPM_RETRY);
    }

    else
    {
        ShowMessage("rejecting...");
        pDDEdata->cbData   = 0;                     /* Data length */
        pDDEdata->fsStatus = DDE_NOTPROCESSED;      /* Status      */
        pDDEdata->usFormat = DDEFMT_TEXT;           /* Text format */
        WinDdePostMsg(hClientWnd,
                      hwnd,
                      WM_DDE_ACK,
                      pDDEdata,
                      DDEPM_RETRY);
    }
    ShowMessage("sent...");
```

Figure 14-11 (Part 2 of 2). How to Perform a One-Time Data Transfer

Establishing a Permanent Data Link

The client posts a WM_DDE_ADVISE message to the server to set up a permanent data link. The advise message contains a shared-memory pointer containing a DDESTRUCT data structure with the item name, format information, and status information. The sample code illustrated in Figure 14-12 shows how to establish a link.

```
WinDdePostMsg(hwndServer,                /* Handle of server     */
              hwndClient,                /* Handle of client     */
              WM_DDE_ADVISE,             /* Message              */
              pddeStruct,                /* Shared-memory pointer */
              DDEPM_RETRY);              /* Retry                */
```

Figure 14-12. How to Establish a Link

When a link is established with the DDE_FNODATA status flag set, a notification, not the data itself, is posted to the client each time the data changes. In this case, the server does not render the new version of the item when the source data changes, but simply posts a WM_DDE_DATA message with 0 bytes of data and the DDE_FNODATA status flag set, as shown in the sample code illustrated in Figure 14-13.

```
/*  Specify the data length and status flag, */
/*  when allocating shared memory            */
pDDEdata->cdData  = 0;
pDDEdata->fstatus = DDE_FNODATA;
    .
    .
    .

/*  Post the message */
WinDdePostMsg(hwndClient,                /* Handle of client     */
              hwndServer,                /* Handle of server     */
              WM_DDE_DATA,               /* Message              */
              pddeStruct,                /* Shared-memory pointer */
              DDEPM_RETRY);              /* Retry                */
```

Figure 14-13. When the Link Is Established with DDE_FNODATA

Terminating a Permanant Link

The client terminates a data link by posting a WM_DDE_UNADVISE message to the server, as shown in the sample code illustrated in Figure 14-14.

```
WinDdePostMsg(hwndServer,                /* Handle of server     */
              hwndClient,                /* Handle of client     */
              WM_DDE_UNADVISE,           /* Message              */
              pddeStruct,                /* Shared-memory pointer */
              DDEPM_RETRY);              /* Retry                */
```

Figure 14-14. How to Terminate a Permanent Link

Executing Commands in a Remote Application

To execute a remote command, the client application posts to the server a WM_DDE_EXECUTE message containing a pointer to a shared-memory object that contains a DDESTRUCT data structure and a command string, as shown in the sample code illustrated in Figure 14-15 on page 14-22.

```
WinDdePostMsg(hwndServer,            /* Handle of server      */
              hwndClient,            /* Handle of client      */
              WM_DDE_EXECUTE,        /* Message               */
              pddeStruct,            /* Shared-memory pointer */
              DDEPM_RETRY);          /* Retry                 */
```

Figure 14-15. How to Execute a Command

Terminating a DDE Conversation

At any time, either the client or the server may terminate a DDE conversation by posting a WM_DDE_TERMINATE message, as shown in the sample code illustrated in Figure 14-16.

```
WinDdePostMsg(hwndDest,              /* Handle of destination    */
              hwndSource,            /* Handle of source         */
              WM_DDE_TERMINATE,      /* Message                  */
              NULL,                  /* No shared-memory pointer */
              DDEPM_RETRY);          /* Retry                    */
```

Figure 14-16. How to Terminate a DDE Conversation

Sample Code for Dynamic Data Exchange

This section illustrates a complete sample program for both client and server applications involved in dynamic data exchange (DDE). Several parts of this program are explained in "Using Dynamic Data Exchange" on page 14-14.

Client Application Sample Code

The client application includes the following files:

- DDEC.C
- DDEC.RC
- DDEC.H
- DDEC.DEF
- DDEC.LNK
- DDEC.MAK

Figure 14-17 on page 14-23 shows the client application sample code.

```
===============
DDEC.C
===============

#define  INCL_WIN
#define  INCL_DOS

#include <os2.h>
#include <stdio.h>
#include "ddec.h"

#pragma  linkage (main,optlink)
INT      main(VOID);
void     ShowMessage(PSZ);

/*********************************************************************/
/*  Main() - program entry point.                                    */
/*********************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);

HAB         hab;
HWND        hFrameWnd, hListWnd, hServerWnd;
PFNWP       SysWndProc;

INT main (VOID)
{
    HMQ         hmq;
    FRAMECDATA  fcd;
    QMSG        qmsg;

    if (!(hab = WinInitialize(0)))
      return FALSE;

    if (!(hmq = WinCreateMsgQueue(hab, 0)))
      return FALSE;
```

Figure 14-17 (Part 1 of 9). Sample Code for a Client Application

```
/*************************************************************************/
/*  Setup the frame control data for the frame window.                  */
/*************************************************************************/
    fcd.cb = sizeof(FRAMECDATA);
    fcd.flCreateFlags = FCF_TITLEBAR         |
                        FCF_SYSMENU          |
                        FCF_MENU             |
                        FCF_SIZEBORDER       |
                        FCF_SHELLPOSITION    |
                        FCF_MINMAX           |
                        FCF_TASKLIST;

    fcd.hmodResources = NULLHANDLE;

/*************************************************************************/
/*  Set our resource key (so PM can find menus, icons, etc).            */
/*************************************************************************/
    fcd.idResources = DDEC;

/*************************************************************************/
/*  Create the frame - it will hold the container control.              */
/*************************************************************************/
    hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                                WC_FRAME,
                                "DDE Client",
                                0, 0, 0, 0, 0,
                                NULLHANDLE,
                                HWND_TOP,
                                DDEC,
                                &fcd,
                                NULL);

/*************************************************************************/
/*  Verify that the frame was created; otherwise, stop.                 */
/*************************************************************************/
    if (!hFrameWnd)
      return FALSE;
```

Figure 14-17 (Part 2 of 9). Sample Code for a Client Application

```
/*********************************************************************/
/*  Set an icon for the frame window.                                */
/*********************************************************************/
    WinSendMsg(hFrameWnd,
               WM_SETICON,
               (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                          SPTR_FOLDER,
                                          FALSE),
                NULL);

/*********************************************************************/
/*  Create a list window child.                                      */
/*********************************************************************/
    hListWnd = WinCreateWindow(hFrameWnd,
                               WC_LISTBOX,
                               NULL,
                               LS_HORZSCROLL,
                               0, 0, 0, 0,
                               hFrameWnd,
                               HWND_BOTTOM,
                               FID_CLIENT,
                               NULL,
                               NULL);

/*********************************************************************/
/*  We must intercept the frame window's messages                    */
/*  (to capture any input from the container control).               */
/*  We save the return value (the current WndProc),                  */
/*  so we can pass it all the other messages the frame gets.         */
/*********************************************************************/
    SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

    WinShowWindow(hFrameWnd, TRUE);

/*********************************************************************/
/*  Standard PM message loop - get it, dispatch i.t                  */
/*********************************************************************/
    while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
    {
       WinDispatchMsg(hab, &qmsg);
    }
```

Figure 14-17 (Part 3 of 9). Sample Code for a Client Application

```
/**********************************************************************/
/*  Clean up on the way out.                                          */
/**********************************************************************/
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);

    return TRUE;
}

/**********************************************************************/
/*  LocalWndProc() - window procedure for the frame window.           */
/*  Called by PM whenever a message is sent to the frame.             */
/**********************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
    PSZ         szData;

    /*  DDE strings */
    PSZ         szApp   = "DDEdemo", szTopic = "System";
    PSZ         szInApp, szInTopic;

    /* System-defined DDE structures */
    CONVCONTEXT context;
    PDDEINIT    pDDEinit;
    PDDESTRUCT  pDDEdata;

    /* Server process and thread IDs */
    PID         pid;
    TID         tid;

    /* Pointer to memory we'll allocate */
    ULONG       mem;

    switch(msg)
     {
```

Figure 14-17 (Part 4 of 9). Sample Code for a Client Application

```
/* All answers to the WinDDEInitate call arrive here */
case WM_DDE_INITIATEACK:
    pDDEinit   = (PDDEINIT)mp2;
    szInApp    = pDDEinit->pszAppName;
    szInTopic  = pDDEinit->pszTopic;
    ShowMessage("server answered...");
    hServerWnd = (HWND)mp1;
    break;

/* All answers to DDE requests arrive here */
case WM_DDE_DATA:
    ShowMessage("data in");
    pDDEdata = (PDDESTRUCT)mp2;
    DosGetSharedMem(pDDEdata, PAG_READ | PAG_WRITE);
    szData = (BYTE *)(pDDEdata+(pDDEdata->offabData));
    ShowMessage(szData);
    break;

/* Menu item processing */
case WM_COMMAND:
    switch (SHORT1FROMMP(mp1))
    {

        /* User starts DDE conversation */
        case IDM_POLL:
            WinPostMsg(hListWnd, LM_DELETEALL, 0, 0);
            ShowMessage("Polling...");
            context.cb = sizeof(CONVCONTEXT);
            context.fsContext = 0;
            WinDdeInitiate(hwnd, szApp, szTopic, &context);
            ShowMessage("Polling complete.");
            break;
```

Figure 14-17 (Part 5 of 9). Sample Code for a Client Application

```
/* User requests data from the server */
case IDM_DATA:

/* Get some sharable memory */
DosAllocSharedMem((PVOID)&mem,
                  NULL,
                  sizeof(DDESTRUCT)+21,
                  PAG_COMMIT |
                  PAG_READ |
                  PAG_WRITE |
                  OBJ_GIVEABLE);

/* Get the server's ID and give it access */
/* to the shared memory                    */
WinQueryWindowProcess(hServerWnd, &pid, &tid);
DosGiveSharedMem(&mem, pid, PAG_READ | PAG_WRITE);

/* Setup DDE data structures                        */
/* (11 byte name length, 10 plus NULL,              */
/*  10 byte data length)                            */
pDDEdata = (PDDESTRUCT)mem;
pDDEdata->cbData   = 10;          /* Data length */
pDDEdata->fsStatus = 0;           /* Status      */
pDDEdata->usFormat = DDEFMT_TEXT;  /* Text format */

/* Go past end of structure for the name */
pDDEdata->offszItemName = sizeof(DDESTRUCT);

/* Go past end of data structure     */
/* (plus past the name) for the data */
pDDEdata->offabData = sizeof(DDESTRUCT)+11;
strcpy((BYTE *)(pDDEdata+(pDDEdata->offszItemName)),
                "STATUS");

/* Post our request to the server program */
WinDdePostMsg(hServerWnd,
              hwnd,
              WM_DDE_REQUEST,
              pDDEdata,
              DDEPM_RETRY;
break;
```

Figure 14-17 (Part 6 of 9). Sample Code for a Client Application

```
            /* User terminates the conversation */
            case IDM_CLOSE:
               WinDdePostMsg(hServerWnd,
                             hwnd,
                             WM_DDE_TERMINATE,
                             NULL,
                             DDEPM_RETRY);
               break;

            /* User closes the window */
            case IDM_EXIT:
               WinPostMsg(hwnd, WM_CLOSE, 0, 0);
               break;
         }
         break;

         /* Send the message to the usual WC_FRAME WndProc */
         default:
            return (*SysWndProc)(hwnd, msg, mp1, mp2);
            break;
   }

   return FALSE;
}

/*********************************************************************/
/*  ShowMessage().                                                   */
/*********************************************************************/
void ShowMessage(PSZ szText)
{
   WinPostMsg(hListWnd,
              LM_INSERTITEM,
              MPFROMSHORT(LIT_END),
              szText);
}

===============
DDEC.RC
===============
#include <os2.h>
#include "ddec.h"
```

Figure 14-17 (Part 7 of 9). Sample Code for a Client Application

```
MENU    DDEC
BEGIN
    SUBMENU         "Commands",    IDM_MENU
    BEGIN
        MENUITEM    "Initiate",    IDM_POLL
        MENUITEM    "Data",        IDM_DATA
        MENUITEM    "Close",       IDM_CLOSE
        MENUITEM    "Exit",        IDM_EXIT
    END
END

===============
DDEC.H
===============
#define DDEC          100
#define IDM_MENU      101
#define IDM_POLL      102
#define IDM_INITIATE  103
#define IDM_DATA      104
#define IDM_CLOSE     105
#define IDM_EXIT      106

===============
DDEC.DEF
===============
NAME DDEC WINDOWAPI

DESCRIPTION 'PM DDE Client Sample'

CODE  MOVEABLE
DATA  MOVEABLE MULTIPLE

STACKSIZE   24576
HEAPSIZE    10240

PROTMODE
```

Figure 14-17 (Part 8 of 9). Sample Code for a Client Application

```
===============
DDEC.LNK
===============
ddec.obj
ddec.exe
ddec.map
ddec.def

===============
DDEC.MAK
===============

CC      = icc /c /Ge /Gd- /Se /Re /ss /Gm+
LINK    = link386
HEADERS = ddec.h

#-------------------------------------------------------------------
#   A list of all of the object files.
#-------------------------------------------------------------------
ALL_OBJ1 = ddec.obj

all: ddec.exe

ddec.res: ddec.rc ddec.h

ddec.obj: ddec.c $(HEADERS)

ddec.exe: $(ALL_OBJ1) ddec.def ddec.lnk ddec.res
          $(LINK) @ddec.lnk
          rc -p -x ddec.res ddec.exe
```

Figure 14-17 (Part 9 of 9). Sample Code for a Client Application

Server Application Sample Code

The server application includes the following files:

- DDES.C
- DDES.RC
- DDES.H
- DDES.DEF
- DDES.LNK
- DDES.MAK

Figure 14-18 on page 14-32 shows the server application sample code.

```
===============
DDES.C
===============

#define  INCL_WIN
#define  INCL_WINDDE
#define  INCL_DOS

#include <os2.h>
#include <stdio.h>
#include <string.h>
#include "ddes.h"

#pragma  linkage (main, optlink)
INT      main(VOID);
void     ShowMessage(PSZ);

/*************************************************************************/
/*  Main() - program entry point.                                        */
/*************************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);

HAB     hab;
HWND    hFrameWnd, hListWnd, hClientWnd;
PFNWP   SysWndProc;

INT main (VOID)
{
    HMQ         hmq;
    FRAMECDATA  fcd;
    QMSG        qmsg;

    if (!(hab = WinInitialize(0)))
      return FALSE;

    if (!(hmq = WinCreateMsgQueue(hab, 0)))
      return FALSE;
```
Figure 14-18 (Part 1 of 10). Sample Code for a Server Application

```
/**********************************************************************/
/*  Setup the frame control data for the frame window.               */
/**********************************************************************/
    fcd.cb = sizeof(FRAMECDATA);
    fcd.flCreateFlags = FCF_TITLEBAR      |
                        FCF_SYSMENU       |
                        FCF_MENU          |
                        FCF_SIZEBORDER    |
                        FCF_SHELLPOSITION |
                        FCF_MINMAX        |
                        FCF_TASKLIST;

    fcd.hmodResources = NULLHANDLE;

/**********************************************************************/
/*  Set our resource key (so PM can find menus, icons, etc).          */
/**********************************************************************/
    fcd.idResources = DDES;

/**********************************************************************/
/*  Create the frame window.                                          */
/**********************************************************************/
    hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                                WC_FRAME,
                                "DDE Server",
                                0, 0, 0, 0, 0,
                                NULLHANDLE,
                                HWND_TOP,
                                DDES,
                                &fcd,
                                NULL);

/**********************************************************************/
/*  Verify that the frame was created; otherwise, stop.               */
/**********************************************************************/
    if (!hFrameWnd)
      return FALSE;
```

Figure 14-18 (Part 2 of 10). Sample Code for a Server Application

```
/***********************************************************************/
/*  Set an icon for the frame window.                                  */
/***********************************************************************/
    WinSendMsg(hFrameWnd,
               WM_SETICON,
               (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                          SPTR_FOLDER,
                                          FALSE),
               NULL);

/***********************************************************************/
/*  Create a list window child.                                        */
/***********************************************************************/
    hListWnd = WinCreateWindow(hFrameWnd,
                               WC_LISTBOX,
                               NULL,
                               LS_HORZSCROLL,
                               0, 0, 0, 0,
                               hFrameWnd,
                               HWND_BOTTOM,
                               FID_CLIENT,
                               NULL,
                               NULL);

/***********************************************************************/
/*  We must intercept the frame window's messages                      */
/*  (to capture any input from the container control).                 */
/*  We save the return value (the current WndProc),                    */
/*  so we can pass it all the other messages the frame gets.           */
/***********************************************************************/
    SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

    WinShowWindow(hFrameWnd, TRUE);

/***********************************************************************/
/*  Standard PM message loop - get it, dispatch it.                    */
/***********************************************************************/
    while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
    {
       WinDispatchMsg(hab, &qmsg);
    }
```

Figure 14-18 (Part 3 of 10). Sample Code for a Server Application

```
/**********************************************************************/
/*  Clean up on the way out.                                          */
/**********************************************************************/
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);

    return TRUE;
}

/**********************************************************************/
/*  LocalWndProc() - window procedure for the frame window.           */
/*  Called by PM whenever a message is sent to the frame.             */
/**********************************************************************/
MRESULT EXPENTRY LocalWndProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
    /* Our inbound DDE stuff */
    PSZ         szClientApp;
    PSZ         szClientTopic;
    PSZ         szReqItem;

    /* Our supported DDE stuff */
    PSZ         szApp    = "DDEdemo";
    PSZ         szTopic  = "System";
    PSZ         szItem   = "Status";
    PSZ         szStatus = "RUNNING";

    /* System DDE structures */
    CONVCONTEXT context;
    PDDEINIT    pDDEinit;
    PDDESTRUCT  pDDEdata;

    /* Miscellaneous */
    PID         pid;
    TID         tid;
    PVOID       mem;

    switch(msg)
    {
        /* All WinDDEInitate calls arrive here */
        case WM_DDE_INITIATE:
           ShowMessage("init");
           hClientWnd = (HWND)mp1;
           pDDEinit = (PDDEINIT)mp2;
```

Figure 14-18 (Part 4 of 10). Sample Code for a Server Application

```
        /* Check incoming poll - if the App and Topic match,     */
        /* we must acknowledge. If both are NULL, the client is  */
        /* searching for anyone - send our names                 */
        szClientApp   = pDDEinit->pszAppName;
        szClientTopic = pDDEinit->pszTopic;
        ShowMessage(szClientApp);
        ShowMessage(szClientTopic);

        if (!strcmpi(szClientApp, szApp) ||
            !strcmpi(szClientApp, NULL))
        {
            if (!strcmpi(szClientTopic, szTopic) ||
                !strcmpi(szClientTopic, NULL) )
            {
                context.cb = sizeof(CONVCONTEXT);
                context.fsContext = 0;
                WinDdeRespond(hClientWnd,
                              hwnd,
                              szApp,
                              szTopic,
                              &context);
            }
        }
        break;

    /* Incoming DDE request - get the item name, send the data out. */
    case WM_DDE_REQUEST:
        ShowMessage("request in...");
        hClientWnd = (HWND)mp1;

        /* The DDE structure is passed, and          */
        /*  the client should have shared it with us  */
        pDDEdata  = (PDDESTRUCT)mp2;
        szReqItem = (BYTE *)(pDDEdata+(pDDEdata->offszItemName));
        ShowMessage(szReqItem);
```

Figure 14-18 (Part 5 of 10). Sample Code for a Server Application

```
/* We support item status, but not anything else */
if (!strcmpi(szReqItem, szItem))
{
   ShowMessage("sending...");

   /* Get some sharable memory */
   DosAllocSharedMem((PVOID)&mem,
                     NULL,
                     sizeof(DDESTRUCT)+21,
                     PAG_COMMIT |
                     PAG_READ |
                     PAG_WRITE |
                     OBJ_GIVEABLE);

   /* Get the server's id and give it access */
   /* to the shared memory                   */
   WinQueryWindowProcess(hClientWnd, &pid, &tid);
   DosGiveSharedMem(&mem, pid, PAG_READ | PAG_WRITE);

   /* Setup DDE data structures                        */
   /* (11 byte name length, 10 plus NULL,              */
   /*  10 byte data length)                            */
   pDDEdata = (PDDESTRUCT)mem;
   pDDEdata->cbData   = 10;            /* Data length */
   pDDEdata->fsStatus = 0;             /* Status      */
   pDDEdata->usFormat = DDEFMT_TEXT;   /* Text format */

   /* Go past end of structure for the name */
   pDDEdata->offszItemName = sizeof(DDESTRUCT);

   /* Go past end of structure (and name) for the data  */
   pDDEdata->offabData = sizeof(DDESTRUCT)+11;
   strcpy((BYTE *)(pDDEdata+(pDDEdata->offabData)), szStatus);
   WinDdePostMsg(hClientWnd,
                 hwnd,
                 WM_DDE_DATA,
                 pDDEdata,
                 DDEPM_RETRY;
}
```

Figure 14-18 (Part 6 of 10). Sample Code for a Server Application

```
         else
         {
            ShowMessage("rejecting...");
            pDDEdata->cbData   = 0;                /* Data length */
            pDDEdata->fsStatus = DDE_NOTPROCESSED; /* Status      */
            pDDEdata->usFormat = DDEFMT_TEXT;      /* Text format */
            WinDdePostMsg(hClientWnd,
                          hwnd,
                          WM_DDE_ACK,
                          pDDEdata,
                          DDEPM_RETRY;
         }
         ShowMessage("sent...");
         break;

         /* Menu item processing */
         case WM_COMMAND:
         switch (SHORT1FROMMP(mp1))
         {
            case IDM_EXIT:
               WinPostMsg(hwnd, WM_CLOSE, 0, 0);
               break;
            default:
               return (*SysWndProc)(hwnd, msg, mp1, mp2);
               break;
         }
         break;

         /* Send the message to the usual WC_FRAME WndProc */
         default:
            return (*SysWndProc)(hwnd, msg, mp1, mp2);
            break;
   }
   return (MRESULT)FALSE;
}
```

Figure 14-18 (Part 7 of 10). Sample Code for a Server Application

```
/*************************************************************************/
/*  ShowMessage().                                                       */
/*************************************************************************/
void ShowMessage(PSZ szText)
{
    WinPostMsg(hListWnd,
               LM_INSERTITEM,
               MPFROMSHORT(LIT_END),
               szText);
}

===============
DDES.RC
===============
#include <os2.h>
#include "ddes.h"

MENU    DDES
BEGIN
    SUBMENU       "Commands",  IDM_MENU
    BEGIN
        MENUITEM  "Exit",      IDM_EXIT
    END
END

===============
DDES.H
===============
#define DDES        100
#define IDM_MENU    1000
#define IDM_EXIT    1001
```

Figure 14-18 (Part 8 of 10). Sample Code for a Server Application

```
===============
DDES.DEF
===============
NAME       DDES WINDOWAPI

DESCRIPTION 'PM DDE Server Sample'

CODE       MOVEABLE
DATA       MOVEABLE MULTIPLE

STACKSIZE 24576
HEAPSIZE  10240

PROTMODE

===============
DDES.LNK
===============
ddes.obj
ddes.exe
ddes.map
ddes.def

===============
DDES.MAK
===============
CC      = icc /c /Ge /Gd- /Se /Re /ss /Gm+
LINK    = link386
HEADERS = ddes.h
```

Figure 14-18 (Part 9 of 10). Sample Code for a Server Application

```
#----------------------------------------------------------------
#   A list of all of the object files.
#----------------------------------------------------------------
ALL_OBJ1 = ddes.obj

all: ddes.exe

ddes.res: ddes.rc ddes.h

ddes.obj: ddes.c $(HEADERS)

ddes.exe: $(ALL_OBJ1) ddes.def ddes.lnk ddes.res
          $(LINK) @ddes.lnk
          rc -p -x ddes.res ddes.exe
```

Figure 14-18 (Part 10 of 10). Sample Code for a Server Application

Related Functions

This section covers the functions that are related to dynamic data exchange.

WinDdeInitiate

This function is issued by a client application to one or more other applications, to request initiation of a dynamic data exchange conversation with a national language conversation context.

Syntax

```
#define INCL_WINDDE /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDdeInitiate  (HWND hwndClient, PSZ pszAppName, PSZ pszTopicName,
                      PCONVCONTEXT pContext)
```

Parameters

hwndClient (HWND) – input
Client's window handle.

pszAppName (PSZ) – input
Application name.

pszTopicName (PSZ) – input
Topic name.

pContext (PCONVCONTEXT) – input
Conversation context.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion. The WM_DDE_INITIATE message is successfully sent to all appropriate windows.
FALSE	Error occurred.

WinDdePostMsg

This function is issued by an application to post a message to another application with which it is carrying out a dynamic data exchange conversation with a national language conversation context.

Syntax

```
#define INCL_WINDDE /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

BOOL WinDdePostMsg (HWND hwndTo, HWND hwndFrom, ULONG usMsgId, PDDESTRUCT pData, ULONG ulOptions)

Parameters

hwndTo (HWND) – input
Window handle of target.

hwndFrom (HWND) – input
Window handle of originator.

usMsgId (ULONG) – input
Message identifier.

pData (PDDESTRUCT) – input
Pointer to the DDE control structure being passed.

ulOptions (ULONG) – input
Options.

DDEPM_RETRY
: This controls what happens if the message cannot be posted because the destination queue is full.

  If this option is set, then message posting is retried at 1-second intervals, until the message is posted successfully. In this case, this function dispatches any messages in the queue of the application issuing this function, by calling the WinPeekMsg and WinDispatchMsg functions in a loop, so that messages sent by other applications can be received. This means that the application can continue to receive DDE messages (or other kinds of messages), while attempting to post DDE messages, thereby preventing deadlock between two applications whose queues are full and who are both attempting to post a message to each other with this option set.

Applications which rely on inspecting messages prior to issuing the WinPeekMsg function can either, use the WinSetHook function and detect the above situation in the invoked hook procedure by testing the MSGF_DDEPOSTMSG value of the *msgf* parameter, or not use this option, in order to avoid the deadlock situation.

If this option is not set, then this function returns FALSE without retrying.

Note: If the message posting fails for any other reason (for example, an invalid window handle is specified), this function returns FALSE even if this option has been selected.

DDEPM_NOFREE This option prevents the WinDdePostMsg call from freeing the shared memory block passed in on the pData parameter. If this option is used, the caller is responsible for freeing the memory block at some subsequent time (for example, the same memory block could be used in multiple calls to WinDdePostMsg and then freed once at the end of those calls.

If this option is not specified, the DDE structure will be freed.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDdeRespond

This function is issued by a server application to indicate that it can support a dynamic data exchange conversation on a particular topic with a national language conversation context.

Syntax

```
#define INCL_WINDDE /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

MRESULT WinDdeRespond  (HWND hwndClient, HWND hwndServer,
                        PSZ pszAppName, PSZ pszTopicName,
                        PCONVCONTEXT pContext)
```

Parameters

hwndClient (HWND) – input
Client's window handle.

hwndServer (HWND) – input
Server's window handle.

pszAppName (PSZ) – input
Application name.

pszTopicName (PSZ) – input
Topic name.

pContext (PCONVCONTEXT) – input
Conversation context.

Returns

mresReply (MRESULT) – returns
Message return data.

Related Window Messages

This section covers the window messages that are related to dynamic data exchange.

WM_DDE_ACK

This message notifies an application of the receipt and processing of a WM_DDE_EXECUTE, WM_DDE_DATA, WM_DDE_ADVISE, WM_DDE_UNADVISE or WM_DDE_POKE message, and in some cases, of a WM_DDE_REQUEST message.

This message is always posted.

Parameters

param1

hwnd (HWND)
Window handle of the sender.

param2

pDdeStruct (PDDESTRUCT)
DDE structure.

This points to a dynamic data exchange structure. See "DDESTRUCT" on page 14-56.

The acknowledging application modifies the *fsStatus* field to return information about the status of the message received:

DDE_FACK	1=request accepted, 0=request not accepted
DDE_FBUSY	1=busy, 0=not busy
DDE_NOTPROCESSED	Reserved for application-specific return codes
DDE_FAPPSTATUS	The message was not understood and was ignored.

An application is expected to set DDE_FBUSY if it is unable to respond to the request at the time it is received. The DDE_FBUSY flag is defined only when DDE_FACK is 0.

offszItemName identifies the item for which the acknowledgment is being sent.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DDE_ADVISE

This message (posted by a client application) requests the receiving application to supply an update for a data item whenever it changes.

This message is always posted.

Parameters

param1

hwnd (HWND)
Window handle of the sender.

param2

pDdeStruct (PDDESTRUCT)
DDE structure.

This points to a dynamic data exchange structure. See "DDESTRUCT" on page 14-56.

Flags in the *fsStatus* field are set as follows:

DDE_FACKREQ	If this bit is 1, the receiving (server) application is requested to send its WM_DDE_DATA messages with the acknowledgment-requested (DDE_FACKREQ) bit set. This offers a flow control technique, whereby the client application can avoid overload from incoming WM_DDE_DATA messages.
DDE_FNODATA	If this bit is 1, the server is requested to send its WM_DDE_DATA messages with a zero length data portion. These messages are alarms that tell the client the source data has changed. Upon receiving one of these alarms, the client can choose to call for the latest version of the data by issuing a WM_DDE_REQUEST message, or the client can choose to ignore the alarm. This is typically used when there is a significant resource cost associated with actually rendering and/or assimilating the data.

offszItemName identifies which data item is being requested.

usFormat is the preferred type of data of the client. It must be a registered DDE data format number.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DDE_DATA

This message notifies a client application of the availability of data. It is always posted.

Parameters

param1

hwnd (HWND)
Window handle of the sender.

param2

pDdeStruct (PDDESTRUCT)
DDE structure.

This points to a dynamic data exchange structure. See "DDESTRUCT" on page 14-56.

Flags in the *fsStatus* field are set as follows:

DDE_FACKREQ	If this bit is 1, the receiving (client) application is expected to send a WM_DDE_ACK message after the memory object has been processed. If it is 0, the client application should not send a WM_DDE_ACK message.
DDE_FRESPONSE	If this bit is 1, this data is offered in response to a WM_DDE_REQUEST message. If it is 0, this data is offered in response to a WM_DDE_ADVISE message.

offszItemName identifies which data item is available.

offabData is the data. The format of the data is a registered DDE data format, identified by the *usFormat* field.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DDE_EXECUTE

This message posts a string to a server application to be processed as a series of commands. The server application is expected to post a WM_DDE_ACK message in response.

This message is always posted.

Parameters

param1

hwnd (HWND)
Window handle of the server.

param2

pDdeStruct (PDDESTRUCT)
DDE structure.

This points to a dynamic data exchange structure. See "DDESTRUCT" on page 14-56.

offabData contains the commands to be executed.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DDE_INITIATE

This message is sent by an application to one or more other applications, to request initiation of a conversation.

This message is always sent.

Parameters

param1

hwnd (HWND)
Window handle of the sender.

param2

pData (PDDEINIT)
Pointer to initiation data.

This points to a DDEINIT structure. *pszAppName* is the name of the desired server application; if this is a zero-length string, any application can respond. *pszTopic* is the name of the desired topic; if this is a zero-length string, each responding application responds once for each topic that it can support.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_DDE_INITIATEACK

This message is sent by a server application in response to a WM_DDE_INITIATE message, for each topic that the server application wishes to support.

Parameters

param1

hwnd (HWND)
Window handle of the sender.

param2

pData (PDDEINIT)
Pointer to initiation data.

This points to a DDEINIT structure. *pszAppName* is the name of the responding server application; it must not be a zero-length string. *pszTopic* is the name of the topic that the server is willing to support; it must not be a zero-length string.

The DDEINIT structure must be in a shareable segment; it is the responsibility of the receiving window procedure to free this segment.

Returns

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_DDE_POKE

This message requests an application to accept an unsolicited data item. It is always posted.

Parameters

param1

hwnd (HWND)
Window handle of the sender.

param2

pDdeStruct (PDDESTRUCT)
DDE structure.

This points to a dynamic data exchange structure. See "DDESTRUCT" on page 14-56.

offszItemName identifies the data item to the receiving application.

offabData is the data. The format of the data is a registered DDE data format, identified by the *usFormat* field.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DDE_REQUEST

This message is posted from client to server, to request that the server provide a data item to the client.

This message is always posted.

Parameters

param1

hwnd (HWND)
Window handle of the server.

param2

DdeStruct (PDDESTRUCT)
DDE structure.

This points to a dynamic data exchange structure. See "DDESTRUCT" on page 14-56.

offszItemName identifies which data item is being requested.

usFormat identifies in which registered DDE data format the data item is to be rendered.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DDE_TERMINATE

This message is posted by either application participating in a DDE conversation, to terminate that conversation.

This message is always posted.

Parameters

param1

hwnd (HWND)
Window handle of the sender.

param2

ulReserved (ULONG)
Reserved value, should be 0.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

WM_DDE_UNADVISE

This message is posted by a client application to a server application to indicate that the specified item should no longer be updated.

This message is always posted.

Parameters

param1

hwnd (HWND)
Window handle of a sender.

param2

DdeStruct (PDDESTRUCT)
DDE structure.

This points to a dynamic data exchange structure (see "DDESTRUCT" on page 14-56). *offszItemName* identifies which data update request is to be retracted. If this is a zero-length string, data update requests for all items are retracted.

Returns

ulReserved (ULONG)
Reserved value, should be 0.

Related Data Structures

This section covers the data structures that are related to dynamic data exchange.

CONVCONTEXT

Dynamic-data-exchange conversation context structure.

Syntax

```
typedef struct _CONVCONTEXT {
ULONG       cb;
ULONG       fsContext;
ULONG       idCountry;
ULONG       usCodepage;
ULONG       usLangID;
ULONG       usSubLangID;
 } CONVCONTEXT;

typedef CONVCONTEXT *PCONVCONTEXT;
```

Fields

cb (ULONG)
Length of structure.

This must be set to the length of the CONVCONTEXT structure.

fsContext (ULONG)
Options.

DDECTXT_CASESENSITIVE	All strings in this conversation are case sensitive.

idCountry (ULONG)
Country code.

usCodepage (ULONG)
Code-page identity.

usLangID (ULONG)
Language.

Zero is valid and means no language information.

usSubLangID (ULONG)
Sub-language.

Zero is valid and means no sub-language information.

DDEINIT

Dynamic-data-exchange initiation structure.

Syntax

```
typedef struct _DDEINIT {
ULONG       cb;
PSZ         pszAppName;
PSZ         pszTopic;
ULONG       offConvContext;
 } DDEINIT;

typedef DDEINIT *PDDEINIT;
```

Fields

cb (ULONG)
Length of structure.

This must be set to the length of the DDEINIT structure.

pszAppName (PSZ)
Application name.

Pointer to name of the server application.

Application names must not contain slashes or backslashes. These characters are reserved for future use in network implementations.

pszTopic (PSZ)
Topic.

Pointer to name of the topic.

offConvContext (ULONG)
Conversation context.

Offset to a CONVCONTEXT structure.

DDESTRUCT

Dynamic-data-exchange control structure.

Syntax

```
typedef struct _DDESTRUCT {
ULONG       cbData;
USHORT      fsStatus;
USHORT      usFormat;
USHORT      offszItemName;
USHORT      offabData;
 } DDESTRUCT;

typedef DDESTRUCT *PDDESTRUCT;
```

Fields

cbData (ULONG)
Length of the data.

This is the length of data that occurs after the *offabData* parameter. If no data exists, this field should contain a zero (0).

fsStatus (USHORT)
Status of the data exchange.

DDE_FACK	Positive acknowledgement
DDE_FBUSY	Application is busy
DDE_FNODATA	No data transfer for advise
DDE_FACKREQ	Acknowledgements are requested
DDE_FRESPONSE	Response to WM_DDE_REQUEST
DDE_NOTPROCESSED	DDE message not understood
DDE_FAPPSTATUS	A 1-byte field of bits that are reserved for application-specific returns.

usFormat (USHORT)
Data format.

One of the DDE data formats.

DDEFMT_TEXT	Text format.
Other	DDE format registered with the atom manager, using the system atom table. The predefined DDE formats are guaranteed not to conflict with the values returned by the atom manager.

offszItemName (USHORT)
Offset to item name.

This is the offset to the item name from the start of this structure. Item name is a null (0x00) terminated string. If no item name exists, there must be a single null (0x00) character in this position. (That is, ItemName is ALWAYS a null terminated string.)

offabData (USHORT)
Offset to beginning of data.

This is the offset to the data, from the start of this structure. This field should be calculated regardless of the presence of data. If no data exists, **cbData** must be zero (0).

For compatibility reasons, this data should not contain embedded pointers. Offsets should be used instead.

Summary

Following are tables that describe the OS/2 functions, window messages, and data structures used in dynamic data exchange:

Table 14-4. DDE Functions

Function Name	Description
WinDdeInitiate	Issued by a client application to one or more other applications, to request initiation of a DDE conversation with a national language conversation context.
WinDdePostMsg	Issued by an application to post a message to another application with which it is carrying out a DDE conversation with a national language conversation.
WinDdeRespond	Issued by a server application to indicate that it can support a DDE conversation on a particular topic with a national language conversation context.

Table 14-5. DDE Window Messages

Message Name	Description
WM_DDE_ACK	Notifies an application of the receipt and processing of a WM_DDE_EXECUTE, WM_DDE_DATA, WM_DDE_UNADVISE, or WM_DDE_POKE message, and in some cases, a WM_DDE_REQUEST message.
WM_DDE_ADVISE	Requests the receiving application to supply an update for a data item whenever it changes.
WM_DDE_DATA	Notifies a client application of the availability of data.
WM_DDE_EXECUTE	Posts a string to a server application to be processed as a series of commands.
WM_DDE_INITIATE	Sent by an application to one or more other applications to request initiation of a conversation.
WM_DDE_INITIATEACK	Sent by a server application in response to a WM_DDE_INITIATE message, for each topic that the server application wishes to support.
WM_DDE_POKE	Requests an application to accept an unsolicited data item.
WM_DDE_REQUEST	Posted from client to server, to request that the server provide a data item to the client.
WM_DDE_TERMINATE	Posted by either application participating in a DDE conversation to terminate that conversation.
WM_DDE_UNADVISE	Posted by a client application to a server application to indicate that the specified item should be updated no longer.

Table 14-6. DDE Data Structures

Data Structure Name	Description
CONVCONTEXT	Dynamic data exchange conversation context data structure.
DDEINIT	Dynamic data exchange initiation data structure.
DDESTRUCT	Dynamic data exchange control data structure.

Chapter 15. Atom Tables

Atom tables enable applications to generate unique identifiers and manage strings. This chapter describes how to use atom tables in PM applications.

About Atom Tables

An *atom table* is an operating system mechanism that an application uses to obtain unique, system-wide identifiers to manage strings efficiently. An application places a string, called an *atom name*, into an atom table and receives a 32-bit integer value, called an *atom*, that the application can use to access that string.

System Atom Table

The *system atom table* is available to all applications. When an application places a string in the system atom table, any application that has the atom name can obtain the atom by querying the system atom table.

An application that defines messages, clipboard-data formats, or dynamic data exchange (DDE) data formats that are intended for use among applications must place the names of the messages or formats in the system atom table. It avoids possible conflicts with messages or formats defined by the system or other applications, and makes the atoms for the messages or formats available to other applications. Applications should use names that are not likely to be used by other applications for other purposes.

Some PM functions enable applications to use atoms in parameters that normally take pointers to strings. For example, WinRegisterClass takes a pointer to a string for its *pszClassName* parameter. WinRegisterClass places the class name string in the system atom table. Afterward, an application can query the system atom table to obtain the atom, then use the atom as the *pszClientClass* parameter of WinCreateStdWindow. This process can save space in the data segment of applications that create many windows of the same private class.

Every atom table has a unique handle. An application must obtain the handle before performing any atom operations. To obtain the handle of the system atom table, an application uses WinQuerySystemAtomTable. The atom-table handle returned by this call is used for all other atom functions.

Private Atom Tables

An application can use a *private atom table* to efficiently manage a large number of strings that are used only within the application. The strings in a private atom table, and the resulting atoms, are available only to the application that created the table.

An application that must use the same string in a number of data structures can save data-segment space by using a private atom table. Rather than copying the string into each data structure, the application can place the string in the atom table and use the resultant atom in the data structures. In this way, a string that appears only once in the data segment still can be used many times in the application.

Applications also can use private atom tables to save time when searching for a particular string. To perform a search, an application must place the search string in the atom table only once, then compare the resultant atom with the atoms in the relevant data structures. This usually is faster than doing string comparisons.

Every atom table has a unique handle. An application must obtain the handle before performing any atom operations. To create a private atom table and obtain its handle, an application must use WinCreateAtomTable. The atom-table handle returned by this call must be used for all other atom functions.

An application that no longer needs its private atom table should call WinDestroyAtomTable to destroy the table and free the memory that the system allocated for the table.

Atom Types

Applications can use two *types* of atoms: string and integer.

String Atoms

Applications pass null-terminated strings to atom tables and receive *string atoms* (32-bit integers) in return. String atoms have the following properties:

- The maximum number of string atoms allowed is 16K. The values of string atoms are from *0xC000* through *0xFFFF*.
- The maximum amount of data that an atom table can store is 60K. This includes the control data that the operating system uses to manage the atom table (32 bytes for the table plus 8 bytes for each string atom).
- The maximum length of an atom name is 255 characters. A zero-length string is not a valid atom name.
- Case is significant when searching for an atom name in an atom table, and the entire string must match. No substring matching is performed.
- A usage count is associated with each atom name. The count is incremented each time the atom name is added to the table and decremented each time the atom name is deleted from the table. This allows different users of the same string atom to avoid destroying each other's atom names. When the usage count for an atom name equals zero, the system removes the atom and atom name from the table.

Integer Atoms

Integer atoms differ from string atoms as follows:

- Integer atoms are values from *0x0001* through *0xBFFF*. The values of integer atoms and string atoms do not overlap, so the two types of atoms can be intermixed.
- The string representation of an integer atom is *ddddd*, where *ddddd* are decimal digits. Leading zeros are ignored.
- There is no usage count nor storage overhead associated with an integer atom.

The operating system uses integer atoms to detect whether the same window class name is being defined more than once. The system defines the predefined window class names using integer atoms as constants. When an application registers a window class, the system enters the specified class name in the system atom table. The system then compares the resultant atom with the predefined window-class constants and with the atoms representing the application-defined class names registered earlier. To be able to do this comparison, the system must express the preregistered class names as atoms. By defining the class names as integer atoms, the system ensures that the atoms do not conflict with the string atoms it generates for application-defined class names.

Atom Creation and Usage Count

An application creates an atom by calling WinAddAtom, passing an atom-table handle and a pointer to a string. The system searches the specified atom table for the string. If the string already resides in the atom table, the system increments the usage count for the string and returns the corresponding atom to the application. Repeated calls to add the same atom string return the same atom. If the atom string does not exist in the table when WinAddAtom is called, the string is added to the table, its usage count is set to 1, and a new atom is returned.

An application can retrieve the usage count associated with a given atom using WinQueryAtomUsage. By obtaining the usage count, an application can detect whether other applications, or other threads within the application, are using the same atom.

Atom Deletion

An application calls WinDeleteAtom when it no longer needs to use an atom. WinDeleteAtom reduces the usage count of the corresponding atom by 1. When the usage count reaches zero, the system deletes the atom name from the table.

Atom Queries

An application can find out if a particular string is already in an atom table by using WinFindAtom. WinFindAtom searches the atom table for the specified string and, if the string is there, returns the corresponding atom.

There are two functions that an application can use to retrieve a string from an atom table, provided that the application has the atom corresponding to the desired string. The first, WinQueryAtomLength, returns the length of the string corresponding to the atom. This allows the application to create a buffer of the appropriate size for the string. The second, WinQueryAtomName, retrieves the string and copies it to the buffer.

Atom String Formats

The second parameter to WinAddAtom and WinFindAtom, *pszAtomName*, is a pointer to zero-terminated string. An application can specify this pointer in four ways, as shown in Table 15-1.

Table 15-1. Atom Table String Formats

Format Name	Description
"!",atom	Points to a string in which the atom is passed indirectly, as a value.
#ddddd	Points to an integer atom specified as a decimal string.
ulong: FFFF(low word)	Passes an atom directly. The atom is in the low word of the *pszAtomName* parameter. The operating system uses this format to add predefined window classes to the system atom table.
string atom name	The pointer is to a string atom name. Applications typically use this format to add an atom string to an atom table and receive an atom in return.

The *"!",atom* and *ulong: FFFF(low word)* formats are useful when incrementing the usage count of an existing atom for which the original atom string is not known. For example, the system clipboard manager uses the *ulong: FFFF(low word)* format to increment the usage count of each clipboard-format atom when that format is placed on the clipboard. By using this format, the atom is not destroyed even if the original user of the atom deletes it, because the usage count still shows that the clipboard is using the atom.

Using Atom Tables

This section explains how to create unique window-message atoms, dynamic data exchange (DDE) formats and a clipboard format.

Creating Unique Window-Message Atoms

You must create atoms for your application-defined window messages if other applications are likely to recognize those messages. For example, your application might communicate with another application by using an agreed-upon message that is not defined by the system. Both applications must use the same string identifier for the shared message type—for example, OUR_LINK_MESSAGE. Each time the applications run, they add this string to the system atom table and receive an atom in return. Both applications register the same string in the system atom table, so they both receive the same atom. Then, this atom can be used to identify the message without conflicting with other system-wide message identifiers. A consequence of using atoms to identify a window message is that the message cannot be decoded as a C-language case statement, as usually done, because the value of the atom cannot be known until run time. Instead, you must add a default case that checks the value of the message against the value of the atoms you have registered. The sample code fragment in Figure 15-1 on page 15-5 shows how to add an application-defined message string to the system atom table, then use the resultant atom to broadcast and receive the message.

```
#define IDM_BROADCAST 25

HATOMTBL hatomtblSystem;              /* System atom table handle    */
ATOM atomLinkMessage;                 /* Atom message                */

/* Message text */
UCHAR szLinkMessage[] = "OUR_LINK_MESSAGE";

MRESULT EXPENTRY ClientWndProc(HWND hwnd,ULONG msg,
  MPARAM mp1,MPARAM mp2)
{

 /* At create time obtain atom for text message */
 switch (msg)
 {
   case WM_CREATE:
     hatomtblSystem  = WinQuerySystemAtomTable();
     atomLinkMessage = WinAddAtom(hatomtblSystem, szLinkMessage);
     return FALSE;

   /* Broadcast text message */
   case WM_COMMAND:
     if (SHORT1FROMMP(mp1) == IDM_BROADCAST)
     {
         WinBroadcastMsg(HWND_DESKTOP, atomLinkMessage,
           (MPARAM) NULL, (MPARAM) NULL,
            BMSG_DESCENDANTS | BMSG_POSTQUEUE);
     }
     return FALSE;

   default;

     /* Check for the atom representing "OUR_LINK_MESSAGE" */
     if (msg == atomLinkMessage) return DoOurMessage(...);
     break;
 }

 /* Execute default window procedure */
 return WinDefWindowProc(hwnd,msg,mp1,mp2);
}
```

Figure 15-1. Sample Code for Adding a Message String into the System Atom Table

Creating DDE Formats and a Unique Clipboard Format

Applications that define their own clipboard or DDE formats must register those formats in the system atom table to avoid conflicting with the predefined formats and any formats used by other applications. The sample code fragment in Figure 15-2 on page 15-6 shows how to register a custom format.

```
#define MAX_BUF_SIZE 128

HAB hab;                                    /* Anchor block handle        */
HATOMTBL hatomtblSystem;                    /* System atom table handle   */
ATOM atomFormatID;                          /* Atom message               */
PSZ  pszSrc, pszDest;                       /* String pointers            */
BOOL fSuccess;
CHAR szClipString[MAX_BUF_SIZE];
APIRET rc;

/*********************************************************************/
/* Get the handle of the system atom table,                          */
/* then add the format name to the table.                            */
/*********************************************************************/

/* System atom table handle  */
hatomtblSystem = WinQuerySystemAtomTable();
/* Register format string    */
atomFormatID = WinAddAtom(hatomtblSystem,"SuperCAD_FORMAT")

/*********************************************************************/
/* Obtain data and write data to buffer (szClipString).              */
/*********************************************************************/

/* Open the clipboard */
if (WinOpenClipbrd(hab))
{

/* Allocate a shared memory object for the text data                 */
if (!(rc = DosAllocSharedMem(
     (PVOID)&pszDest,                       /* Pointer to shared memory   */
                                            /* object                     */
      (PSZ) NULL,                           /* Use unnamed shared memory  */
      (ULONG)strlen(
          szClipString) + 1,                /* Amount of memory           */
         PAG_WRITE  |                       /* Allow write access         */
         PAG_COMMIT |                       /* Commit the shared memory   */
         OBJ_GIVEABLE)))                    /* Make pointer giveable      */
{

   /* Setup the source pointer to point to text */
   pszSrc = szClipString;
```

Figure 15-2 (Part 1 of 2). Sample Code for Registering a Custom Format

```
   /* Copy the string to the allocated memory   */
   while (*pszDest++ = *pszSrc++);

   /* Clear old data from the clipboard         */
   WinEmptyClipbrd(hab);

   /* Pass the pointer to the clipboard in custom format.          */
   /* Notice that the pointer must be a ULONG value.               */
   fSuccess = WinSetClipbrdData(hab,   /* Anchor block handle      */
      (ULONG) pszDest,                 /* Pointer to text data     */
      atomFormatID,                    /* Custom format ID (atom)  */
      CFI_POINTER);                    /* Passing a pointer        */

   /* Close the clipboard */
   WinCloseClipbrd(hab);
}
}
```

Figure 15-2 (Part 2 of 2). Sample Code for Registering a Custom Format

Related Functions

This section covers the functions that are related to atom tables.

WinAddAtom

This function adds an atom to an atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ATOM WinAddAtom  (HATOMTBL  hatomtblAtomTbl, PSZ  AtomName)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

AtomName (PSZ) – input
Atom name.

Returns

atom (ATOM) – returns
Atom value.

Atom	The atom associated with the passed string
0	Invalid atom-table handle or invalid atom name specified.

WinCreateAtomTable

This function creates a private empty atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HATOMTBL WinCreateAtomTable  (ULONG  ulInitial, ULONG  ulBuckets)
```

Parameters

ulInitial (ULONG) – input
Initial bytes.

ulBuckets (ULONG) – input
Size of the hash table.

Returns

hatomtblAtomTbl (HATOMTBL) – returns
Atom-table handle.

NULLHANDLE	Call failed.
Other	Atom-table handle. This must be passed as a parameter in subsequent atom manager calls.

WinDeleteAtom

This function deletes an atom from an atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ATOM WinDeleteAtom (HATOMTBL hatomtblAtomTbl, ATOM atom)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

atom (ATOM) – input
Atom identifying the atom to be deleted.

Returns

rc (ATOM) – returns
Return code.

0	Call successful
Other	The call fails and the atom has not been deleted, in which case this is equal to the *atom* parameter.

WinDestroyAtomTable

This function destroys a private atom table, which is created by WinCreateAtomTable.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HATOMTBL WinDestroyAtomTable  (HATOMTBL hatomtblAtomTbl)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

Returns

rc (HATOMTBL) – returns
Return code.

0	Function successful.
Other	The call fails and the atom table has not been destroyed, in which case this is equal to the *hatomtblAtomTbl* parameter.

WinFindAtom

This function finds an atom in the atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ATOM WinFindAtom  (HATOMTBL hatomtblAtomTbl, PSZ pszAtomName)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

pszAtomName (PSZ) – input
Atom name.

Returns

atom (ATOM) – returns
Atom value.

Atom	The atom associated with the passed string
0	Invalid atom table handle or invalid atom name specified.

WinQueryAtomLength

This function queries the length of an atom represented by the specified atom.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryAtomLength (HATOMTBL hatomtblAtomTbl, ATOM atom)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

atom (ATOM) – input
Atom whose associated character-string length is to be returned.

Returns

ulretlen (ULONG) – returns
String length.

0	The specified atom or the atom table is invalid.
Other	The length of the character string associated with the atom **excluding** the null terminating byte. Integer atoms always return a length of six.

WinQueryAtomName

This function returns an atom name associated with an atom.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryAtomName (HATOMTBL hatomtblAtomTbl, ATOM atom,
                       PSZ pszBuffer, ULONG ulBufferMax)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

atom (ATOM) – input
Identifies the character string to be retrieved.

pszBuffer (PSZ) – output
Buffer to receive the character string.

ulBufferMax (ULONG) – input
Buffer size in bytes.

Returns

ulretlen (ULONG) – returns
Length of retrieved character string.

0	The specified atom or the atom table is invalid.
Other	The number of bytes copied to the buffer **excluding** the terminating zero.

WinQueryAtomUsage

This function returns the number of times an atom has been used.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */

#include <os2.h>

ULONG WinQueryAtomUsage  (HATOMTBL hatomtblAtomTbl, ATOM atom)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

atom (ATOM) – input
Atom whose use count is to be returned.

Returns

ulcount (ULONG) – returns
Use count of the atom.

65535	Integer atom
0	The specified atom or the atom table is invalid
Other	Use count.

WinQuerySystemAtomTable

This function returns the handle of the system atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HATOMTBL WinQuerySystemAtomTable ()
```

Parameters

None.

Returns

hatomtblAtomTbl (HATOMTBL) – returns
System atom-table handle.

WinRegisterUserDatatype

This function registers a data type and defines its structure.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinRegisterUserDatatype (HAB hab, LONG datatype, LONG count,
                              PLONG types)
```

Parameters

hab (HAB) – input
Anchor-block handle.

datatype (LONG) – input
Data type code to be defined.

count (LONG) – input
Number of elements.

types (PLONG) – input
Data type codes of structure components.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinRegisterUserMsg

This function registers a user message and defines its parameters.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>

BOOL WinRegisterUserMsg  (HAB hab, ULONG msgid, LONG datatype1,
                          LONG dir1, LONG datatype2, LONG dir2,
                          LONG datatyper)
```

Parameters

hab (HAB) – input
Anchor-block handle.

msgid (ULONG) – input
Message identifier.

datatype1 (LONG) – input
Data type of message parameter 1.

DTYP_BIT16	See BIT16 data type.
DTYP_BIT32	See BIT32 data type.
DTYP_BIT8	See BIT8 data type.
DTYP_BOOL	See BOOL data type.
DTYP_LONG	See LONG data type.
DTYP_SHORT	See SHORT data type.
DTYP_UCHAR	See UCHAR data type.
DTYP_ULONG	See ULONG data type.
DTYP_USHORT	See USHORT data type.
DTYP_P*	A pointer to a system data type. Note that not all of the system data types that exist in the CPI are valid.
< -DTYP_USER	A pointer to a user data type. The user data type must have already been defined via WinRegisterUserDatatype.

dir1 (LONG) – input
Direction of message parameter 1.

RUM_IN	Input parameter (inspected by the recipient of the message, but not altered)
RUM_OUT	Output parameter (altered by the recipient of the message, without inspecting its value first)
RUM_INOUT	Input/output parameter (inspected by the recipient of the message, and then altered).

datatype2 (LONG) – input
Data type of message parameter 2.

dir2 (LONG) – input
Direction of message parameter 2.

datatyper (LONG) – input
Data type of message reply.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Summary

Following is a table that describes the OS/2 functions used with atom table:

Table 15-2. Atom Table Functions

Function Name	Description
WinAddAtom	Adds an atom to an atom table.
WinCreateAtomTable	Creates an empty private atom table.
WinDeleteAtom	Deletes an atom from an atom table.
WinDestroyAtomTable	Destroys a private atom table.
WinFindAtom	Find an atom in the atom table.
WinQueryAtomLength	Queries the length of an atom represented by the specified atom.
WinQueryAtomUsage	Returns the number of times an atom has been used.
WinQuerySystemAtomTable	Returns the handle of the system atom table.
WinRegisterUserDatatype	Registers a data type and defines its structure.
WinRegisterUserMsg	Registers a user message and defines its parameters.

Appendix. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood NY 10594, U.S.A.

Trademarks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries:

Common User Access	CUA
Operating System/2	OS/2
PM	Presentation Manager
SAA	Systems Application Architecture

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows. Other trademarks are trademarks of their respective companies.

Helvetica	Linotype Company
Times New Roman	Monotype Corporation, Limited

Double-Byte Character Set (DBCS)

Throughout this publication, you will see reference to specific values for character strings. The values are for single-byte character set (SBCS). If you use the double-byte character set (DBCS), note that one DBCS character equals two SBCS characters.

Glossary

This glossary defines many of the terms used in this book. It includes terms and definitions from the *IBM Dictionary of Computing*, as well as terms specific to the OS/2 operating system and the Presentation Manager. It is not a complete glossary for the entire OS/2 operating system; nor is it a complete dictionary of computer terms.

Other primary sources for these definitions are:

- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyrighted 1990 by the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. These definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Glossary Listing

A

accelerator. In SAA Common User Access architecture, a key or combination of keys that invokes an application-defined function.

accelerator table. A table used to define which key strokes are treated as *accelerators* and the commands they are translated into.

access mode. The manner in which an application gains access to a file it has opened. Examples of access modes are read-only, write-only, and read/write.

access permission. All access rights that a user has regarding an object. (I)

action. One of a set of defined tasks that a computer performs. Users request the application to perform an action in several ways, such as typing a command, pressing a function key, or selecting the action name from an action bar or menu.

action bar. In SAA Common User Access architecture, the area at the top of a window that contains choices that give a user access to actions available in that window.

action point. The current position on the screen at which the pointer is pointing. Contrast with *hot spot* and *input focus*.

active program. A program currently running on the computer. An active program can be interactive (running and receiving input from the user) or noninteractive (running but not receiving input from the user). See also *interactive program* and *noninteractive program*.

active window. The window with which the user is currently interacting.

address space. (1) The range of addresses available to a program. (A) (2) The area of virtual storage available for a particular job.

alphanumeric video output. Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

American National Standard Code for Information Interchange. The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

Note: IBM has defined an extension to ASCII code (characters 128-255).

anchor. A window procedure that handles Presentation Manager* message conversions between an icon procedure and an application.

anchor block. An area of Presentation-Manager-internal resources to allocated process or thread that calls WinInitialize.

anchor point. A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI. American National Standards Institute.

APA. All points addressable.

API. Application programming interface.

application. A collection of software components used to perform specific types of work on a computer; for example, a payroll application, an airline reservation application, a network application.

application object. In SAA Advanced Common User Access architecture, a form that an application provides for a user; for example, a spreadsheet form. Contrast with *user object*.

application programming interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

application-modal. Pertaining to a message box or dialog box for which processing must be completed before further interaction with any other window owned by the same application may take place.

area. In computer graphics, a filled shape such as a solid rectangle.

ASCII. American National Standard Code for Information Interchange.

ASCIIZ. A string of ASCII characters that is terminated with a byte containing the value 0.

aspect ratio. In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous (ASYNC). (1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (T) (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions. See also *synchronous*.

atom. A constant that represents a string. As soon as a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an *atom table*. See also *integer atom*.

atom table. A table used to relate *atoms* with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

atomic operation. An operation that completes its work on an object before another operation can be performed on the same object.

attribute. A characteristic or property that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes*.

automatic link. In Information Presentation Facility (IPF), a link that begins a chain reaction at the primary window. When the user selects the primary window, an automatic link is activated to display secondary windows.

AVIO. Advanced Video Input/Output.

B

Bézier curve. (1) A mathematical technique of specifying smooth continous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bézier. (2) (D of C) In the AIX Graphics Library, a cubic spline approximation to a set of four control points that passes through the first and fourth control points and that has a continuous slope where two spline segments meet. Named after Dr. P. Bézier.

background. (1) In multiprogramming, the conditions under which low-priority programs are executed. Contrast with *foreground*. (2) An active session that is not currently displayed on the screen.

background color. The color in which the background of a graphic primitive is drawn.

background mix. An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with *mix*.

background program. In multiprogramming, a program that executes with a low priority. Contrast with *foreground program*.

bit map. A representation in memory of the data displayed on an APA device, usually the screen.

block. (1) A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or logical records. (T) (2) To record data in a block. (3) A collection of contiguous records recorded as a unit. Blocks are separated by interblock gaps and each block may contain one or more records. (A)

block device. A storage device that performs I/O operations on blocks of data called *sectors*. Data on block devices can be randomly accessed. Block devices are designated by a drive letter (for example, **C:**).

blocking mode. A condition set by an application that determines when its threads might block. For example, an application might set the Pipemode parameter for the DosCreateNPipe function so that its threads perform I/O operations to the named pipe block when no data is available.

border. A visual indication (for example, a separator line or a background color) of the boundaries of a window.

boundary determination. An operation used to compute the size of the smallest rectangle that encloses a graphics object on the screen.

breakpoint. (1) A point in a computer program where execution may be halted. A breakpoint is usually at the beginning of an instruction where halts, caused by external intervention, are convenient for resuming execution. (T) (2) A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

broken pipe. When all of the handles that access one end of a pipe have been closed.

bucket. One or more fields in which the result of an operation is kept.

buffer. (1) A portion of storage used to hold input or output data temporarily. (2) To allocate and schedule the use of buffers. (A)

button. A mechanism used to request or initiate an action. See also *barrel buttons*, *bezel buttons*, *mouse button*, *push button*, and *radio button*.

byte pipe. Pipes that handle data as byte streams. All unnamed pipes are byte pipes. Named pipes can be byte pipes or message pipes. See *byte stream*.

byte stream. Data that consists of an unbroken stream of bytes.

C

cache. A high-speed buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.

cached micro presentation space. A presentation space from a Presentation-Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

CAD. Computer-Aided Design.

call. (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (I) (A) (2) To transfer control to a procedure, program, routine, or subroutine.

calling sequence. A sequence of instructions together with any associated data necessary to execute a call. (T)

Cancel. An action that removes the current window or menu without processing it, and returns the previous window.

cascaded menu. In the OS/2 operating system, a menu that appears when the arrow to the right of a cascading choice is selected. It contains a set of choices that are related to the cascading choice. Cascaded menus are used to reduce the length of a menu. See also *cascading choice*.

cascading choice. In SAA Common User Access architecture, a choice in a menu that, when selected, produces a cascaded menu containing other choices. An arrow (→) appears to the right of the cascading choice.

CASE statement. In PM programming, provides the body of a window procedure. There is usually one CASE statement for each message type supported by an application.

CGA. Color graphics adapter.

chained list. A list in which the data elements may be dispersed but in which each data element contains information for locating the next. (T) Synonymous with *linked list*.

character. A letter, digit, or other symbol.

character box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also *character mode*. Contrast with *character cell*.

character cell. The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box*.

character code. The means of addressing a character in a character set, sometimes called *code point*.

character device. A device that performs I/O operations on one character at a time. Because character devices view data as a stream of bytes, character-device data cannot be randomly accessed. Character devices include the keyboard, mouse, and printer, and are referred to by name.

character mode. A mode that, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

character set. (1) An ordered set of unique representations called characters; for example, the 26 letters of English alphabet, Boolean 0 and 1, the set of symbols in the Morse code, and the 128 ASCII characters. (A) (2) All the valid characters for a programming language or for a computer system. (3) A group of characters used for a specific reason; for example, the set of characters a printer can print.

check box. In SAA Advanced Common User Access architecture, a square box with associated text that represents a choice. When a user selects a choice, an X appears in the check box to indicate that the choice is in effect. The user can clear the check box by selecting the choice again. Contrast with *radio button*.

check mark. (1) (D of C) In SAA Advanced Common User Access architecture, a (√) symbol that shows that a choice is currently in effect. (2) The symbol that is used to indicate a selected item on a pull-down menu.

child process. In the OS/2 operating system, a process started by another process, which is called the parent process. Contrast with *parent process*.

child window. A window that appears within the border of its parent window (either a primary window or another child window). When the parent window is resized, moved, or destroyed, the child window also is resized, moved, or destroyed; however, the child window can be moved or resized independently from the parent window, within the boundaries of the parent window. Contrast with *parent window*.

choice. (1) An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol). (2) (D of C) In SAA Common User Access architecture, an item that a user can select.

chord. (1) To press more than one button on a pointing device while the pointer is within the limits that the user has specified for the operating environment. (2) (D of C) In graphics, a short line segment whose end points lie on a circle. Chords are a means for producing a circular image from straight lines. The higher the number of chords per circle, the smoother the circular image.

class. In object-oriented design or programming, a group of objects that share a common definition and that therefore share common properties, operations, and behavior. Members of the group are called instances of the class.

class method. In System Object Model, an action that can be performed on a class object. Synonymous with factory method.

class object. In System Object Model, the run-time implementation of a class.

class style. The set of properties that apply to every window in a window class.

client. (1) A functional unit that receives shared services from a server. (T) (2) A user, as in a client process that uses a named pipe or queue that is created and owned by a server process.

client area. The part of the window, inside the border, that is below the menu bar. It is the user's work space, where a user types information and selects choices from selection fields. In primary windows, it is where an application programmer presents the objects that a user works on.

client program. An application that creates and manipulates instances of classes.

client window. The window in which the application displays output and receives input. This window is located inside the frame window, under the window title bar and any menu bar, and within any scroll bars.

clip limits. The area of the paper that can be reached by a printer or plotter.

clipboard. In SAA Common User Access architecture, an area of computer memory, or storage, that temporarily holds data. Data in the clipboard is available to other applications.

clipping. In computer graphics, removing those parts of a display image that lie outside a given boundary. (I) (A)

clipping area. The area in which the window can paint.

clipping path. A clipping boundary in world-coordinate space.

clock tick. The minimum unit of time that the system tracks. If the system timer currently counts at a rate of X Hz, the system tracks the time every 1/X of a second. Also known as *time tick*.

CLOCK$. Character-device name reserved for the system clock.

code page. An assignment of graphic characters and control-function meanings to all code points.

code point. (1) Synonym for *character code*. (2) (D of C) A 1-byte code representing one of 256 potential characters.

code segment. An executable section of programming code within a load module.

color dithering. See *dithering*.

color graphics adapter (CGA). An adapter that simultaneously provides four colors and is supported by all IBM Personal Computer and Personal System/2 models.

command. The name and parameters associated with an action that a program can perform.

command area. An area composed of a command field prompt and a command entry field.

command entry field. An entry field in which users type commands.

command line. On a display screen, a display line, sometimes at the bottom of the screen, in which only commands can be entered.

command mode. A state of a system or device in which the user can enter commands.

command prompt. A field prompt showing the location of the command entry field in a panel.

Common Programming Interface (CPI). Definitions of those application development languages and services that have, or are intended to have, implementations on and a high degree of commonality across the SAA environments. One of the three SAA architectural areas. See also *Common User Access architecture*.

Common User Access (CUA) architecture. Guidelines for the dialog between a human and a workstation or terminal. One of the three SAA architectural areas. See also *Common Programming Interface*.

compile. To translate a program written in a higher-level programming language into a machine language program.

composite window. A window composed of other windows (such as a frame window, frame-control windows, and a client window) that are kept together as a unit and that interact with each other.

computer-aided design (CAD). The use of a computer to design or change a product, tool, or machine, such as using a computer for drafting or illustrating.

COM1, COM2, COM3. Character-device names reserved for serial ports 1 through 3.

CON. Character-device name reserved for the console keyboard and screen.

container. In SAA Common User Access architecture, an object that holds other objects. A folder is an example of a container object. See also *folder* and *object*.

contextual help. In SAA Common User Access Architecture, help that gives specific information about the item the cursor is on. The help is contextual because it provides information about a specific item as it is currently being used. Contrast with *extended help*.

contiguous. Touching or joining at a common edge or boundary, for example, an unbroken consecutive series of storage locations.

control. In SAA Advanced Common User Access architecture, a component of the user interface that allows a user to select choices or type information; for example, a check box, an entry field, a radio button.

control area. A storage area used by a computer program to hold control information. (I) (A)

Control Panel. In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program. (1) The basic functions of the operating system, including DOS emulation and the support for keyboard, mouse, and video input/output. (2) A computer program designed to schedule and to supervise the execution of programs of a computer system. (I) (A)

control window. A window that is used as part of a composite window to perform simple input and output tasks. Radio buttons and check boxes are examples.

control word. An instruction within a document that identifies its parts or indicates how to format the document.

coordinate space. A two-dimensional set of points used to generate output on a video display of printer.

Copy. A choice that places onto the clipboard, a copy of what the user has selected. See also *Cut* and *Paste*.

correlation. The action of determining which element or object within a picture is at a given position on the display. This follows a *pick* operation.

coverpage window. A window in which the application's help information is displayed.

CPI. Common Programming Interface.

critical extended attribute. An extended attribute that is necessary for the correct operation of the system or a particular application.

critical section. (1) In programming languages, a part of an asynchronous procedure that cannot be executed simultaneously with a certain part of another asynchronous procedure. (I)

Note: Part of the other asynchronous procedure also is a critical section. (2) A section of code that is not reentrant; that is, code that can be executed by only one thread at a time.

CUA architecture. Common User Access architecture.

current position. In computer graphics, the position, in user coordinates, that becomes the starting point for the next graphics routine, if that routine does not explicitly specify a starting point.

cursor. A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with *pointer* and *input focus*.

Cut. In SAA Common User Access architecture, a choice that removes a selected object, or a part of an object, to the clipboard, usually compressing the space it occupied in a window. See also *Copy* and *Paste*.

D

daisy chain. A method of device interconnection for determining interrupt priority by connecting the interrupt sources serially.

data segment. A nonexecutable section of a program module; that is, a section of a program that contains data definitions.

data structure. The syntactic structure of symbolic expressions and their storage-allocation characteristics. (T)

data transfer. The movement of data from one object to another by way of the clipboard or by direct manipulation.

DBCS. Double-byte character set.

DDE. Dynamic data exchange.

deadlock. (1) Unresolved contention for the use of a resource. (2) An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by, or a response from, the other. (3) An impasse that occurs when multiple processes are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar wait state.

debug. To detect, diagnose, and eliminate errors in programs. (T)

decipoint. In printing, one tenth of a point. There are 72 points in an inch.

default procedure. A function provided by the Presentation Manager Interface that may be used to process standard messages from dialogs or windows.

default value. A value assumed when no value has been specified. Synonymous with assumed value. For example, in the graphics programming interface, the default line-type is 'solid'.

definition list. A type of list that pairs a term and its description.

delta. An application-defined threshold, or number of container items, from either end of the list.

descendant. See *child process*.

descriptive text. Text used in addition to a field prompt to give more information about a field.

Deselect all. A choice that cancels the selection of all of the objects that have been selected in that window.

Desktop Manager. In the Presentation Manager, a window that displays a list of groups of programs, each of which can be started or stopped.

desktop window. The window, corresponding to the physical device, against which all other types of windows are established.

detached process. A background process that runs independent of the parent process.

detent. A point on a slider that represents an exact value to which a user can move the slider arm.

device context. A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also *direct device context*, *information device context*, *memory device context*, *metafile device context*, *queued device context*, and *screen device context*.

device driver. A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space. (1) Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units. (2) (D of C) In computer graphics, a space defined by the complete set of addressable points of a display device. (A)

dialog. The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box. In SAA Advanced Common User Access architecture, a movable window, fixed in size, containing controls that a user uses to provide information required by an application so that it can continue to process a user request. See also *message box, primary window, secondary window*. Also known as a *pop-up window*.

Dialog Box Editor. A *WYSIWYG* editor that creates dialog boxes for communicating with the application user.

dialog item. A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog procedure. A dialog window that is controlled by a window procedure. It is responsible for responding to all messages sent to the dialog window.

dialog tag language. A markup language used by the DTL compiler to create dialog objects.

dialog template. The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context. A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also *device context*.

direct manipulation. The action of using the mouse to move objects around the screen. For example, moving files and directories around in the *Workplace Shell*.

direct memory access (DMA). A technique for moving data directly between main storage and peripheral equipment without requiring processing of the data by the processing unit.(T)

directory. A type of file containing the names and controlling information for other files or other directories.

display point. Synonym for *pel*.

dithering. (1) The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work, for example, on narrow lines. (2) (D of C) In computer graphics, a technique of interleaving dark and light pixels so that the resulting image looks smoothly shaded when viewed from a distance.

DMA. Direct memory access.

DOS Protect Mode Interface (DPMI). An interface between protect mode and real mode programs.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

doubleword. A contiguous sequence of bits or characters that comprises two computer words and is capable of being addressed as a unit. (A)

DPMI. DOS Protect Mode Interface.

drag. In SAA Common User Access, to use a pointing device to move an object; for example, clicking on a window border, and dragging it to make the window larger.

dragging. (1) In computer graphics, moving an object on the display screen as if it were attached to the pointer. (2) (D of C) In computer graphics, moving one or more segments on a display surface by translating. (I) (A)

drawing chain. See *segment chain*.

drop. To fix the position of an object that is being dragged, by releasing the select button of the pointing device.

drop. To fix the position of an object that is being dragged, by releasing the select button of the pointing device. See also *drag*.

DTL. Dialog tag language.

dual-boot function. A feature of the OS/2 operating system that allows the user to start DOS from within the operating system, or an OS/2 session from within DOS.

duplex. Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full duplex*.

dynamic data exchange (DDE). A message protocol used to communicate between applications that share data. The protocol uses shared memory as the means of exchanging data between applications.

dynamic data formatting. A formatting procedure that enables you to incorporate text, bit maps or metafiles in an IPF window at execution time.

dynamic link library. A collection of executable programming code and data that is bound to an application at load time or run time, rather than during linking. The programming code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic linking. The process of resolving external references in a program module at load time or run time rather than during linking.

dynamic segments. Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

dynamic storage. (1) A device that stores data in a manner that permits the data to move or vary with time such that the specified data is not always available for recovery. (A) (2) A storage in which the cells require repetitive application of control signals in order to retain stored data. Such repetitive application of the control signals is called a refresh operation. A dynamic storage may use static addressing or sensing circuits. (A) (3) See also *static storage*.

dynamic time slicing. Varies the size of the time slice depending on system load and paging activity.

dynamic-link module. A module that is linked at load time or run time.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters (9 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

edge-triggered. Pertaining to an event semaphore that is posted then reset before a waiting thread gets a chance to run. The semaphore is considered to be posted for the rest of that thread's waiting period; the thread does not have to wait for the semaphore to be posted again.

EGA. Extended graphics adapter.

element. An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

EMS. Expanded Memory Specification.

encapsulation. Hiding an object's implementation, that is, its private, internal data and methods. Private variables and methods are accessible only to the object that contains them.

entry field. In SAA Common User Access architecture, an area where a user types information. Its boundaries are usually indicated. See also *selection field*.

entry panel. A defined panel type containing one or more entry fields and protected information such as headings, prompts, and explanatory text.

entry-field control. The component of a user interface that provides the means by which the application receives data entered by the user in an entry field. When it has the input focus, the entry field displays a flashing pointer at the position where the next typed character will go.

environment segment. The list of environment variables and their values for a process.

environment strings. ASCII text strings that define the value of environment variables.

environment variables. Variables that describe the execution environment of a process. These variables are named by the operating system or by the application. Environment variables named by the operating system are PATH, DPATH, INCLUDE, INIT, LIB, PROMPT, and TEMP. The values of environment variables are defined by the user in the CONFIG.SYS file, or by using the SET command at the OS/2 command prompt.

error message. An indication that an error has been detected. (A)

event semaphore. A semaphore that enables a thread to signal a waiting thread or threads that an event has occurred or that a task has been completed. The waiting threads can then perform an action that is dependent on the completion of the signaled event.

exception. An abnormal condition such as an I/O error encountered in processing a data set or a file.

exclusive system semaphore. A system semaphore that can be modified only by threads within the same process.

executable file. (1) A file that contains programs or commands that perform operations or actions to be taken. (2) A collection of related data records that execute programs.

exit. To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T) Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel*.

expanded memory specification (EMS). Enables DOS applications to access memory above the 1MB real mode addressing limit.

extended attribute. An additional piece of information about a file object, such as its data format or category. It consists of a name and a value. A file object may have more than one extended attribute associated with it.

extended help. In SAA Common User Access architecture, a help action that provides information about the contents of the application window from which a user requested help. Contrast with *contextual help*.

extended-choice selection. A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection*.

extent. Continuous space on a disk or diskette that is occupied by or reserved for a particular data set, data space, or file.

external link. In Information Presentation Facility, a link that connects external online document files.

F

family-mode application. An application program that can run in the OS/2 environment and in the DOS environment; however, it cannot take advantage of many of the OS/2-mode facilities, such as multitasking, interprocess communication, and dynamic linking.

FAT. File allocation table.

FEA. Full extended attribute.

field-level help. Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used; the information is dependent upon the context within the work session.

FIFO. First-in-first-out. (A)

file. A named set of records stored or processed as a unit. (T)

file allocation table (FAT). In IBM personal computers, a table used by the operating system to allocate space on a disk for a file, and to locate and chain together parts of the file that may be scattered on different sectors so that the file can be used in a random or sequential manner.

file attribute. Any of the attributes that describe the characteristics of a file.

File Manager. In the Presentation Manager, a program that displays directories and files, and allows various actions on them.

file specification. The full identifier for a file, which includes its drive designation, path, file name, and extension.

file system. The combination of software and hardware that supports storing information on a storage device.

file system driver (FSD). A program that manages file I\O and controls the format of information on the storage media.

fillet. A curve that is tangential to the end points of two adjoining lines. See also *polyfillet*.

filtering. An application process that changes the order of data in a queue.

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

flag. (1) An indicator or parameter that shows the setting of a switch. (2) A character that signals the occurrence of some condition, such as the end of a word. (A) (3) (D of C) A characteristic of a file or directory that enables it to be used in certain ways. See also *archive flag*, *hidden flag*, and *read-only flag*.

focus. See *input focus*.

folder. A container used to organize objects.

font. A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

Font Editor. A utility program provided with the IBM Developers Toolkit that enables the design and creation of new fonts.

foreground program. (1) The program with which the user is currently interacting. Also known as *interactive program*. Contrast with *background program*. (2) (D of C) In multiprogramming, a high-priority program.

frame. The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles. Standard window layouts provided by the Presentation Manager.

FSD. File system driver.

full-duplex. Synonym for *duplex*.

full-screen application. An application that has complete control of the screen.

function. (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a call. (2) A set of related control statements that cause one or more programs to be performed.

function key. A key that causes a specified sequence of operations to be performed when it is pressed, for example, F1 and Alt-K.

function key area. The area at the bottom of a window that contains function key assignments such as F1=Help.

G

GDT. Global Descriptor Table.

general protection fault. An exception condition that occurs when a process attempts to use storage or a module that has some level of protection assigned to it, such as I/O privilege level. See also *IOPL code segment*.

Global Descriptor Table (GDT). A table that defines code and data segments available to all tasks in an application.

global dynamic-link module. A dynamic-link module that can be shared by all processes in the system that refer to the module name.

global file-name character. Either a question mark (?) or an asterisk (*) used as a variable in a file name or file name extension when referring to a particular file or group of files.

glyph. A graphic symbol whose appearance conveys information.

GPI. Graphics programming interface.

graphic primitive. In computer graphics, a basic element, such as an arc or a line, that is not made up of smaller parts and that is used to create diagrams and pictures. See also *graphics segment*.

graphics. (1) A picture defined in terms of graphic primitives and graphics attributes. (2) (D of C) The making of charts and pictures. (3) Pertaining to charts, tables, and their creation. (4) See *computer graphics, coordinate graphics, fixed-image graphics, interactive graphics, passive graphics, raster graphics*.

graphics attributes. Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also *segment attributes*.

graphics field. The clipping boundary that defines the visible part of the presentation-page contents.

graphics mode. One of several states of a display. The mode determines the resolution and color content of the screen.

graphics model space. The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as *model space*.

Graphics programming interface. The formally defined programming language that is between an IBM graphics program and the user of the program.

graphics segment. A sequence of related graphic primitives and graphics attributes. See also *graphic primitive*.

graying. The indication that a choice on a pull-down is unavailable.

group. A collection of logically connected controls. For example, the buttons controlling paper size for a printer could be called a group. See also *program group*.

H

handle. (1) An identifier that represents an object, such as a device or window, to the Presentation Interface. (2) (D of C) In the Advanced DOS and OS/2 operating systems, a binary value created by the system that identifies a drive, directory, and file so that the file can be found and opened.

hard error. An error condition on a network that requires either that the system be reconfigured or that the source of the error be removed before the system can resume reliable operation.

header. (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message, such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

heading tags. A document element that enables information to be displayed in windows, and that controls entries in the contents window controls placement of push buttons in a window, and defines the shape and size of windows.

heap. An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

help function. (1) A function that provides information about a specific field, an application panel, or information about the help facility. (2) (D of C) One or more display images that describe how to use application software or how to do a system operation.

Help index. In SAA Common User Access architecture, a help action that provides an index of the help information available for an application.

help panel. A panel with information to assist users that is displayed in response to a help request from the user.

help window. A Common-User-Access-defined secondary window that displays information when the user requests help.

hidden file. An operating system file that is not displayed by a directory listing.

hide button. In the OS/2 operating system, a small, square button located in the right-hand corner of the title bar of a window that, when selected, removes from the screen all the windows associated with that window. Contrast with *maximize button*. See also *restore button*.

hierarchical inheritance. The relationship between parent and child classes. An object that is lower in the inheritance hierarchy than another object, inherits all the characteristics and behaviors of the objects above it in the hierarchy.

hierarchy. A tree of segments beginning with the root segment and proceeding downward to dependent segment types.

high-performance file system (HPFS). In the OS/2 operating system, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. The file system also supports the coexistence of multiple, active file systems on a single personal computer, with the capability of multiple and different storage devices. File names used with the HPFS can have as many as 254 characters.

hit testing. The means of identifying which window is associated with which input device event.

hook. A point in a system-defined function where an application can supply additional code that the system processes as though it were part of the function.

hook chain. A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot spot. The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point*.

HPFS. high-performance file system.

hypergraphic link. A connection between one piece of information and another through the use of graphics.

hypertext. A way of presenting information online with connections between one piece of information and another, called *hypertext links*. See also *hypertext link*.

hypertext link. A connection between one piece of information and another.

I

I/O operation. An input operation to, or output operation from a device attached to a computer.

I-beam pointer. A pointer that indicates an area, such as an entry field in which text can be edited.

icon. In SAA Advanced Common User Access architecture, a graphical representation of an object, consisting of an image, image background, and a label. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions, and minimized programs.

icon area. In the Presentation Manager, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor. The Presentation Manager-provided tool for creating icons.

image font. A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of one of the symbols. Contrast with *outline font*.

indirect manipulation. Interaction with an object through choices and controls.

information device context. A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also *device context*.

information panel. A defined panel type characterized by a body containing only protected information.

Information Presentation Facility (IPF). A facility provided by the OS/2 operating system, by which application developers can produce online documentation and context-sensitive online help panels for their applications.

input focus. (1) The area of a window where user interaction is possible using an input device, such as a mouse or the keyboard. (2) The position in the *active window* where a user's normal interaction with the keyboard will appear.

input router. An internal OS/2 process that removes messages from the system queue.

input/output control. A device-specific command that requests a function of a device driver.

installable file system (IFS). A file system in which software is installed when the operating system is started.

instance. A single occurrence of an object class that has a particular behavior.

instruction pointer. In system/38, a pointer that provides addressability for a machine interface instruction in a program.

integer atom. An *atom* that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by Presentation Manager are expressed as integer atoms.

interactive graphics. Graphics that can be moved or manipulated by a user at a terminal.

interactive program. (1) A program that is running (active) and is ready to receive (or is receiving) input from a user. (2) A running program that can receive input from the keyboard or another input device. Compare with *active program* and contrast with *noninteractive program*.

Also known as a *foreground program*.

interchange file. A file containing data that can be sent from one Presentation Manager interface application to another.

interpreter. A program that translates and executes each instruction of a high-level programming language before it translates and executes.

interprocess communication (IPC). In the OS/2 operating system, the exchange of information between processes or threads through semaphores, pipes, queues, and shared memory.

interval timer. (1) A timer that provides program interruptions on a program-controlled basis. (2) An electronic counter that counts intervals of time under program control.

IOCtl. Input/output control.

IOPL. Input/output privilege level.

IOPL code segment. An IOPL executable section of programming code that enables an application to directly manipulate hardware interrupts and ports without replacing the device driver. See also *privilege level*.

IPC. Interprocess communication.

IPF. Information Presentation Facility.

IPF compiler. A text compiler that interpret tags in a source file and converts the information into the specified format.

IPF tag language. A markup language that provides the instructions for displaying online information.

item. A data object that can be passed in a DDE transaction.

J

journal. A special-purpose file that is used to record changes made in the system.

K

Kanji. A graphic character set used in Japanese ideographic alphabets.

KBD$. Character-device name reserved for the keyboard.

kernel. The part of an operating system that performs basic functions, such as allocating hardware resources.

kerning. The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

keyboard accelerator. A keystroke that generates a command message for an application.

keyboard augmentation. A function that enables a user to press a keyboard key while pressing a mouse button.

keyboard focus. A temporary attribute of a window. The window that has a keyboard focus receives all keyboard input until the focus changes to a different window.

Keys help. In SAA Common User Access architecture, a help action that provides a listing of the application keys and their assigned functions.

L

label. In a graphics segment, an identifier of one or more elements that is used when editing the segment.

LAN. local area network.

language support procedure. A function provided by the Presentation Manager Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

lazy drag. See *pickup and drop*.

lazy drag set. See *pickup set*.

LDT. In the OS/2 operating system, Local Descriptor Table.

LIFO stack. A stack from which data is retrieved in last-in, first-out order.

linear address. A unique value that identifies the memory object.

linked list. Synonym for *chained list*.

list box. In SAA Advanced Common User Access architecture, a control that contains scrollable choices from which a user can select one choice.

Note: In CUA architecture, this is a programmer term. The end user term is selection list.

list button. A button labeled with an underlined down-arrow that presents a list of valid objects or choices that can be selected for that field.

list panel. A defined panel type that displays a list of items from which users can select one or more choices and then specify one or more actions to work on those choices.

load time. The point in time at which a program module is loaded into main storage for execution.

load-on-call. A function of a linkage editor that allows selected segments of the module to be disk resident while other segments are executing. Disk resident segments are loaded for execution and given control when any entry point that they contain is called.

local area network (LAN). (1) A computer network located on a user's premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. (T)

Note: A LAN does not use store and forward techniques. (2) A network inwhich a set of devices are connected to one another for communication and that can be connected to a larger network.

Local Descriptor Table (LDT). Defines code and data segments specific to a single task.

lock. A serialization mechanism by means of which a resource is restricted for use by the holder of the lock.

logical storage device. A device that the user can map to a physical (actual) device.

LPT1, LPT2, LPT3. Character-device names reserved for parallel printers 1 through 3.

M

main window. The window that is positioned relative to the *desktop window*.

manipulation button. The button on a pointing device a user presses to directly manipulate an object.

map. (1) A set of values having a defined correspondence with the quantities or values of another set. (I) (A) (2) To establish a set of values having a defined correspondence with the quantities or values of another set. (I)

marker box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol. A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

marquee box. The rectangle that appears during a selection technique in which a user selects objects by drawing a box around them with a pointing device.

Master Help Index. In the OS/2 operating system, an alphabetic list of help topics related to using the operating system.

maximize. To enlarge a window to its largest possible size.

media window. The part of the physical device (display, printer, or plotter) on which a picture is presented.

memory block. Part memory within a heap.

memory device context. A logical description of a data destination that is a memory bit map. See also *device context*.

memory management. A feature of the operating system for allocating, sharing, and freeing main storage.

memory object. Logical unit of memory requested by an application, which forms the granular unit of memory manipulation from the application viewpoint.

menu. In SAA Advanced Common User Access architecture, an extension of the menu bar that displays a list of choices available for a selected choice in the menu bar. After a user selects a choice in menu bar, the corresponding menu appears. Additional pop-up windows can appear from menu choices.

menu bar. In SAA Advanced Common User Access architecture, the area near the top of a window, below the title bar and above the rest of the window, that contains choices that provide access to other menus.

menu button. The button on a pointing device that a user presses to view a pop-up menu associated with an object.

message. (1) In the Presentation Manager, a packet of data used for communication between the Presentation Manager interface and Presentation Manager applications (2) In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process.

message box. (1) A dialog window predefined by the system and used as a simple interface for applications, without the necessity of creating dialog-template resources or dialog procedures. (2) (D of C) In SAA Advanced Common User Access architecture, a type of window that shows messages to users. See also *dialog box, primary window, secondary window*.

message filter. The means of selecting which messages from a specific window will be handled by the application.

message queue. A sequenced collection of messages to be read by the application.

message stream mode. A method of operation in which data is treated as a stream of messages. Contrast with *byte stream*.

metacharacter. See *global file-name character*.

metaclass. The conjunction of an object and its class information; that is, the information pertaining to the class as a whole, rather than to a single instance of the class. Each class is itself an object, which is an instance of the metaclass.

metafile. A file containing a series of attributes that set color, shape and size, usually of a picture or a drawing. Using a program that can interpret these attributes, a user can view the assembled image.

metafile device context. A logical description of a data destination that is a metafile, which is used for graphics interchange. See also *device context*.

metalanguage. A language used to specify another language. For example, data types can be described using a metalanguage so as to make the descriptions independent of any one computer language.

mickey. A unit of measurement for physical mouse motion whose value depends on the mouse device driver currently loaded.

micro presentation space. A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize. To remove from the screen all windows associated with an application and replace them with an icon that represents the application.

mix. An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as *foreground mix*. Contrast with *background mix*.

mixed character string. A string containing a mixture of one-byte and *Kanji* or Hangeul (two-byte) characters.

mnemonic. (1) A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item. (2) (D of C) In SAA Advanced Common User Access architecture, usually a single character, within the text of a choice, identified by an underscore beneath the character. If all characters in a choice already serve as mnemonics for other choices, another character, placed in parentheses immediately following the choice, can be used. When a user types the mnemonic for a choice, the choice is either selected or the cursor is moved to that choice.

modal dialog box. In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that requires a user to enter information before continuing to work in the application window from which it was displayed. Contrast with *modeless dialog box*. Also known as a *serial dialog box*. Contrast with *parallel dialog box*.

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

model space. See *graphics model space*.

modeless dialog box. In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that allows users to continue their dialog with the application without entering information in the dialog box. Also known as a *parallel dialog box*. Contrast with *modal dialog box*.

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

module definition file. A file that describes the code segments within a load module. For example, it indicates whether a code segment is loadable before module execution begins (preload), or loadable only when referred to at run time (load-on-call).

mouse. In SAA usage, a device that a user moves on a flat surface to position a pointer on the screen. It allows a user to select a choice o function to be performed or to perform operations on the screen, such as dragging or drawing lines from one position to another.

MOUSE$. Character-device name reserved for a mouse.

multiple-choice selection. In SAA Basic Common User Access architecture, a type of field from which a user can select one or more choices or select none. See also *check box*. Contrast with *extended-choice selection*.

multiple-line entry field. In SAA Advanced Common User Access architecture, a control into which a user types more than one line of information. See also *single-line entry field*.

multitasking. The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

mutex semaphore. (Mutual exclusion semaphore). A semaphore that enables threads to serialize their access to resources. Only the thread that currently owns the mutex semaphore can gain access to the resource, thus preventing one thread from interrupting operations being performed by another.

muxwait semaphore. (Multiple wait semaphore). A semaphore that enables a thread to wait either for multiple event semaphores to be posted or for multiple mutex semaphores to be released. Alternatively, a muxwait semaphore can be set to enable a thread to wait for any ONE of the event or mutex semaphores in the muxwait semaphore's list to be posted or released.

N

named pipe. A named buffer that provides client-to-server, server-to-client, or full duplex communication between unrelated processes. Contrast with *unnamed pipe*.

national language support (NLS). The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature, MRI, or documentation of a product.

nested list. A list that is contained within another list.

NLS. national language support.

non-8.3 file-name format. A file-naming convention in which file names can consist of up to 255 characters. See also *8.3 file-name format*.

noncritical extended attribute. An extended attribute that is not necessary for the function of an application.

nondestructive read. Reading that does not erase the data in the source location. (T)

noninteractive program. A running program that cannot receive input from the keyboard or other input device. Compare with *active program*, and contrast with *interactive program*.

nonretained graphics. Graphic primitives that are not remembered by the Presentation Manager interface when they have been drawn. Contrast with *retained graphics*.

null character (NUL). (1) Character-device name reserved for a nonexistent (dummy) device. (2) (D of C) A control character that is used to accomplish media-fill or time-fill and that may be inserted into or removed from a sequence of characters without affecting the meaning of the sequence; however, the control of equipment or the format may be affected by this character. (I) (A)

null-terminated string. A string of (n+1) characters where the (n+1)th character is the 'null' character (0x00). Also known as 'zero-terminated' string and 'ASCIIZ' string.

O

object. A set of data and actions that can be performed on that data.

Object Interface Definition Language (OIDL). Specification language for SOM class definitions.

object window. A window that does not have a parent but which might have child windows. An object window cannot be presented on a device.

OIDL. Object Interface Definition Language.

open. To start working with a file, directory, or other object.

ordered list. Vertical arrangements of items, with each item in the list preceded by a number or letter.

outline font. A set of symbols, each of which is created as a series of lines and curves. Synonymous with *vector font*. Contrast with *image font*.

output area. An area of storage reserved for output. (A)

owner window. A window into which specific events that occur in another (owned) window are reported.

ownership. The determination of how windows communicate using messages.

owning process. The process that owns the resources that might be shared with other processes.

P

page. (1) A 4KB segment of contiguous physical memory. (2) (D of C) A defined unit of space on a storage medium.

page viewport. A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint. (1) The action of drawing or redrawing the contents of a window. (2) In computer graphics, to shade an area of a display image; for example, with crosshatching or color.

panel. In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

panel area. An area within a panel that contains related information. The three major Common User Access-defined panel areas are the action bar, the function key area, and the panel body.

panel area separator. In SAA Basic Common User Access architecture, a solid, dashed, or blank line that provides a visual distinction between two adjacent areas of a panel.

panel body. The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body can contain protected information, selection fields, and entry fields. The layout and content of the panel body determine the panel type.

panel body area. See *client area*.

panel definition. A description of the contents and characteristics of a panel. A panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID. In SAA Basic Common User Access architecture, a panel identifier, located in the upper-left corner of a panel. A user can choose whether to display the panel ID.

panel title. In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

paper size. The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box. See *modeless dialog box*.

parameter list. A list of values that provides a means of associating addressability of data defined in a called program with data in the calling program. It contains parameter names and the order in which they are to be associated in the calling and called program.

parent process. In the OS/2 operating system, a process that creates other processes. Contrast with *child process*.

parent window. In the OS/2 operating system, a window that creates a child window. The child window is drawn within the parent window. If the parent window is moved, resized, or destroyed, the child window also will be moved, resized, or destroyed. However, the child window can be moved and resized independently from the parent window, within the boundaries of the parent window. Contrast with *child window*.

partition. (1) A fixed-size division of storage. (2) On an IBM personal computer fixed disk, one of four possible storage areas of variable size; one may be accessed by DOS, and each of the others may be assigned to another operating system.

Paste. A choice in the Edit pull-down that a user selects to move the contents of the clipboard into a preselected location. See also *Copy* and *Cut*.

path. The route used to locate files; the storage location of a file. A fully qualified path lists the drive identifier, directory name, subdirectory name (if any), and file name with the associated extension.

PDD. Physical device driver.

peeking. An action taken by any thread in the process that owns the queue to examine queue elements without removing them.

pel. (1) The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for *display point*, *pixel*, and *picture element*. (2) (D of C) Picture element.

physical device driver (PDD). A system interface that handles hardware interrupts and supports a set of input and output functions.

pick. To select part of a displayed object using the pointer.

pickup. To add an object or set of objects to the pickup set.

pickup and drop. A drag operation that does not require the direct manipulation button to be pressed for the duration of the drag.

pickup set. The set of objects that have been picked up as part of a pickup and drop operation.

picture chain. See *segment chain*.

picture element. (1) Synonym for *pel*. (2) (D of C) In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity. (T) . (3) The area of the finest detail that can be reproduced effectively on the recording medium.

PID. Process identification.

pipe. (1) A named or unnamed buffer used to pass data between processes. A process reads from or writes to a pipe as if the pipe were a standard-input or standard-output file. See also *named pipe* and *unnamed pipe*. (2) (D of C) To direct data so that the output from one process becomes the input to another process. The standard output of one command can be connected to the standard input of another with the pipe operator (|).

pixel. (1) Synonym for *pel*. (2) (D of C) Picture element.

plotter. An output unit that directly produces a hardcopy record of data on a removable medium, in the form of a two-dimensional graphic representation. (T)

PM. Presentation Manager.

pointer. (1) The symbol displayed on the screen that is moved by a pointing device, such as a *mouse*. The pointer is used to point at items that users can select. Contrast with *cursor*. (2) A data element that indicates the location of another data element. (T)

POINTER$. Character-device name reserved for a pointer device (mouse screen support).

pointing device. In SAA Advanced Common User Access architecture, an instrument, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

pointings. Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a *mouse*.

polyfillet. A curve based on a sequence of lines. The curve is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also *fillet*.

polygon. One or more closed figures that can be drawn filled, outlined, or filled and outlined.

polyline. A sequence of adjoining lines.

polymorphism. A concept whereby the behavior of an application object is dependent solely upon the class and contents of the messages received by that object, and is not affected by any other external factor.

pop. To retrieve an item from a last-in-first-out stack of items. Contrast with *push*.

pop-up window. (1) A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window. (2) (D of C) In SAA Advanced Common User Access architecture, a movable window, fixed in size, in which a user provides information required by an application so that it can continue to process a user request.

presentation drivers. Special purpose I/O routines that handle field device-independent I/O requests from the PM and its applications.

Presentation Manager (PM). The interface of the OS/2 operating system that presents, in windows a graphics-based interface to applications and files installed and running under the OS/2 operating system.

presentation page. The coordinate space in which a picture is assembled for display.

presentation space (PS). (1) Contains the device-independent definition of a picture. (2) (D of C) The display space on a display device.

primary window. In SAA Common User Access architecture, the window in which the main interaction between the user and the application takes place. In a multiprogramming environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward. See also *secondary window*.

primitive. In computer graphics, one of several simple functions for drawing on the screen, including, for example, the rectangle, line, ellipse, polygon, and so on.

primitive attribute. A specifiable characteristic of a graphic primitive. See *graphics attributes*.

print job. The result of sending a document or picture to be printed.

Print Manager. In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

privilege level. A protection level imposed by the hardware architecture of the IBM personal computer. There are four privilege levels (number 0 through 3). Only certain types of programs are allowed to execute at each privilege level. See also *IOPL code segment*.

procedure call. In programming languages, a language construct for invoking execution of a procedure.

process. An instance of an executing application and the resources it is using.

program. A sequence of instructions that a computer can interpret and execute.

program details. Information about a program that is specified in the *Program Manager* window and is used when the program is started.

program group. In the Presentation Manager, several programs that can be acted upon as a single entity.

program name. The full file specification of a program. Contrast with *program title*.

program title. The name of a program as it is listed in the *Program Manager* window. Contrast with *program name*.

prompt. A displayed symbol or message that requests input from the user or gives operational information; for example, on the display screen of an IBM personal computer, the DOS A> prompt. The user must respond to the prompt in order to proceed.

protect mode. A method of program operation that limits or prevents access to certain instructions or areas of storage. Contrast with *real mode*.

protocol. A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. (I)

pseudocode. An artificial language used to describe computer program algorithms without using the syntax of any particular programming language. (A)

pull-down. (1) An *action bar* extension that displays a list of choices available for a selected action bar choice. After users select an action bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the actions available to users. (2) (D of C) In SAA Common User Access architecture, pertaining to a choice in an action bar pull-down.

push. To add an item to a last-in-first-out stack of items. Contrast with *pop*.

push button. In SAA Advanced Common User Access architecture, a rectangle with text inside. Push buttons are used in windows for actions that occur immediately when the push button is selected.

putback. To remove an object or set of objects from the lazy drag set. This has the effect of undoing the pickup operation for those objects

putdown. To drop the objects in the lazy drag set on the target object.

Q

queue. (1) A linked list of elements waiting to be processed in FIFO order. For example, a queue may be a list of print jobs waiting to be printed. (2) (D of C) A line or list of items waiting to be processed; for example, work to be performed or messages to be displayed.

queued device context. A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context*.

R

radio button. (1) A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from a list. Contrast with *check box*. (2) In SAA Advanced Common User Access architecture, a circle with text beside it. Radio buttons are combined to show a user a fixed set of choices from which only one can be selected. The circle is partially filled when a choice is selected.

RAS. Reliability, availability, and serviceability.

raster. (1) In computer graphics, a predetermined pattern of lines that provides uniform coverage of a display space. (T) (2) The coordinate grid that divides the display area of a display device. (A)

read-only file. A file that can be read from but not written to.

real mode. A method of program operation that does not limit or prevent access to any instructions or areas of storage. The operating system loads the entire program into storage and gives the program access to all system resources. Contrast with *protect mode*.

realize. To cause the system to ensure, wherever possible, that the physical color table of a device is set to the closest possible match in the logical color table.

recursive routine. A routine that can call itself, or be called by another routine that was called by the recursive routine.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase. (1) A word or phrase that is emphasized in a device-dependent manner to inform the user that additional information for the word or phrase is available. (2) (D of C) In hypertext, text that is highlighted and preceded by a single-character input field used to signify the existence of a hypertext link.

reference phrase help. In SAA Common User Access architecture, highlighted words or phrases within help information that a user selects to get additional information.

refresh. To update a window, with changed information, to its current status.

region. A clipping boundary in device space.

register. A part of internal storage having a specified storage capacity and usually intended for a specific purpose. (T)

remote file system. A file-system driver that gains access to a remote system without a block device driver.

resource. The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators, and mnemonics; the definitions are held in a resource file.

resource file. A file containing information used in the definition of a window. Definitions can be of fonts, templates, accelerators, and mnemonics.

restore. To return a window to its original size or position following a sizing or moving action.

retained graphics. Graphic primitives that are remembered by the Presentation Manager interface after they have been drawn. Contrast with *nonretained graphics*.

return code. (1) A value returned to a program to indicate the results of an operation requested by that program. (2) A code used to influence the execution of succeeding instructions.(A)

reverse video. (1) A form of highlighting a character, field, or cursor by reversing the color of the character, field, or cursor with its background; for example, changing a red character on a black background to a black character on a red background. (2) In SAA Basic Common User Access architecture, a screen emphasis feature that interchanges the foreground and background colors of an item.

REXX Language. Restructured Extended Executor. A procedural language that provides batch language functions along with structured programming constructs such as loops; conditional testing and subroutines.

RGB. (1) Color coding in which the brightness of the additive primary colors of light, red, green, and blue, are specified as three distinct values of white light. (2) Pertaining to a color display that accepts signals representing red, green, and blue.

roman. Relating to a type style with upright characters.

root segment. In a hierarchical database, the highest segment in the tree structure.

round-robin scheduling. A process that allows each thread to run for a specified amount of time.

run time. (1) Any instant at which the execution of a particular computer program takes place. (T) (2) The amount of time needed for the execution of a particular computer program. (T) (3) The time during which an instruction in an instruction register is decoded and performed. Synonym for *execution time*.

S

SAA. Systems Application Architecture.

SBCS. Single-byte character set.

scheduler. A computer program designed to perform functions such as scheduling, initiation, and termination of jobs.

screen. In SAA Basic Common User Access architecture, the physical surface of a display device upon which information is shown to a user.

screen device context. A logical description of a data destination that is a particular window on the screen. See also *device context*.

SCREEN$. Character-device name reserved for the display screen.

scroll bar. In SAA Advanced Common User Access architecture, a part of a window, associated with a scrollable area, that a user interacts with to see information that is not currently allows visible.

scrollable entry field. An entry field larger than the visible field.

scrollable selection field. A selection field that contains more choices than are visible.

scrolling. Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window. A window that contains information that is dependent on information in a primary window and is used to supplement the interaction in the primary window.

sector. On disk or diskette storage, an addressable subdivision of a track used to record one block of a program or data.

segment. See *graphics segment*.

segment attributes. Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain. All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonym for *picture chain*.

segment priority. The order in which segments are drawn.

segment store. An area in a normal graphics presentation space where retained graphics segments are stored.

select. To mark or choose an item. Note that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing.

select button. The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor. In SAA Advanced Common User Access architecture, a visual indication that a user has selected a choice. It is represented by outlining the choice with a dotted box. See also *text cursor*.

selection field. (1) In SAA Advanced Common User Access architecture, a set of related choices. See also *entry field*. (2) In SAA Basic Common User Access architecture, an area of a panel that cannot be scrolled and contains a fixed number of choices.

semantics. The relationships between symbols and their meanings.

semaphore. An object used by applications for signalling purposes and for controlling access to serially reusable resources.

separator. In SAA Advanced Common User Access architecture, a line or color boundary that provides a visual distinction between two adjacent areas.

serial dialog box. See *modal dialog box*.

serialization. The consecutive ordering of items.

serialize. To ensure that one or more events occur in a specified sequence.

serially reusable resource (SRR). A logical resource or object that can be accessed by only one task at a time.

session. (1) A routing mechanism for user interaction via the console; a complete environment that determines how an application runs and how users interact with the application. OS/2 can manage more than one session at a time, and more than one process can run in a session. Each session has its own set of environment variables that determine where OS/2 looks for dynamic-link libraries and other important files. (2) (D of C) In the OS/2 operating system, one instance of a started program or command prompt. Each session is separate from all other sessions that might be running on the computer. The operating system is responsible for coordinating the resources that each session uses, such as computer memory, allocation of processor time, and windows on the screen.

Settings Notebook. A control window that is used to display the settings for an object and to enable the user to change them.

shadow box. The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shared data. Data that is used by two or more programs.

shared memory. In the OS/2 operating system, a segment that can be used by more than one program.

shear. In computer graphics, the forward or backward slant of a graphics symbol or string of such symbols relative to a line perpendicular to the baseline of the symbol.

shell. (1) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices, and touch-sensitive screens, and communicate them to the operating system. (2) Software that allows a kernel program to run under different operating-system environments.

shutdown. The process of ending operation of a system or a subsystem, following a defined procedure.

sibling processes. Child processes that have the same parent process.

sibling windows. Child windows that have the same parent window.

simple list. A list of like values; for example, a list of user names. Contrast with *mixed list*.

single-byte character set (SBCS). A character set in which each character is represented by a one-byte code. Contrast with *double-byte character set*.

slider box. In SAA Advanced Common User Access architecture: a part of the scroll bar that shows the position and size of the visible information in a window relative to the total amount of information available. Also known as *thumb mark*.

SOM. System Object Model.

source file. A file that contains source statements for items such as high-level language programs and data description specifications.

source statement. A statement written in a programming language.

specific dynamic-link module. A dynamic-link module created for the exclusive use of an application.

spin button. In SAA Advanced Common User Access architecture, a type of entry field that shows a scrollable ring of choices from which a user can select a choice. After the last choice is displayed, the first choice is displayed again. A user can also type a choice from the scrollable ring into the entry field without interacting with the spin button.

spline. A sequence of one or more Bézier curves.

spooler. A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete and the required device is available. The spooler prevents output from different sources from being intermixed.

stack. A list constructed and maintained so that the next data element to be retrieved is the most recently stored. This method is characterized as last-in-first-out (LIFO).

standard window. A collection of window elements that form a panel. The standard window can include one or more of the following window elements: sizing borders, system menu icon, title bar, maximize/minimize/restore icons, action bar and pull-downs, scroll bars, and client area.

static control. The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

static storage. (1) A read/write storage unit in which data is retained in the absence of control signals. (A) Static storage may use dynamic addressing or sensing circuits. (2) Storage other than *dynamic storage*. (A)

style. See *window style*.

subdirectory. In an IBM personal computer, a file referred to in a root directory that contains the names of other files stored on the diskette or fixed disk.

swapping. (1) A process that interchanges the contents of an area of real storage with the contents of an area in auxiliary storage. (I) (A) (2) In a system with virtual storage, a paging technique that writes the active pages of a job to auxiliary storage and reads pages of another job from auxiliary storage into real storage. (3) The process of temporarily removing an active job from main storage, saving it on disk, and processing another job in the area of main storage formerly occupied by the first job.

switch. (1) In SAA usage, to move the cursor from one point of interest to another; for example, to move from one screen or window to another or from a place within a displayed image to another place on the same displayed image. (2) In a computer program, a conditional instruction and an indicator to be interrogated by that instruction. (3) A device or programming technique for making a selection, for example, a toggle, a conditional jump.

switch list. See *Task List*.

symbolic identifier. A text string that equates to an integer value in an include file, which is used to identify a programming object.

symbols. In Information Presentation Facility, a document element used to produce characters that cannot be entered from the keyboard.

synchronous. Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals. (T) See also *asynchronous*.

System Menu. In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

System Object Model (SOM). A mechanism for language-neutral, object-oriented programming in the OS/2 environment.

system queue. The master queue for all pointer device or keyboard events.

system-defined messages. Messages that control the operations of applications and provides input an other information for applications to process.

Systems Application Architecture (SAA). A set of IBM software interfaces, conventions, and protocols that provide a framework for designing and developing applications that are consistent across systems.

T

table tags. In Information Presentation Facility, a document element that formats text in an arrangement of rows and columns.

tag. (1) One or more characters attached to a set of data that contain information about the set, including its identification. (I) (A) (2) In Generalized Markup Language markup, a name for a type of document or document element that is entered in the source document to identify it.

target object. An object to which the user is transferring information.

Task List. In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

template. An ASCII-text definition of an action bar and pull-down menu, held in a resource file, or as a data structure in program memory.

terminate-and-stay-resident (TSR). Pertaining to an application that modifies an operating system interrupt vector to point to its own location (known as hooking an interrupt).

text. Characters or symbols.

text cursor. A symbol displayed in an entry field that indicates where typed input will appear.

text window. Also known as the VIO window.

text-windowed application. The environment in which the operating system performs advanced-video input and output operations.

thread. A unit of execution within a process. It uses the resources of the process.

thumb mark. The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a *slider box*.

thunk. Term used to describe the process of address conversion, stack and structure realignment, etc., necessary when passing control between 16-bit and 32-bit modules.

tilde. A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

time slice. (1) An interval of time on the processing unit allocated for use in performing a task. After the interval has expired, processing-unit time is allocated to another task, so a task cannot monopolize processing-unit time beyond a fixed limit. (2) In systems with time sharing, a segment of time allocated to a terminal job.

time-critical process. A process that must be performed within a specified time after an event has occurred.

timer. A facility provided under the Presentation Manager, whereby Presentation Manager will dispatch a message of class WM_TIMER to a particular window at specified intervals. This capability may be used by an application to perform a specific processing task at predetermined intervals, without the necessity for the application to explicitly keep track of the passage of time.

timer tick. See *clock tick*.

title bar. In SAA Advanced Common User Access architecture, the area at the top of each window that contains the window title and system menu icon. When appropriate, it also contains the minimize, maximize, and restore icons. Contrast with *panel title*.

TLB. Translation lookaside buffer.

transaction. An exchange between a workstation and another device that accomplishes a particular action or result.

transform. (1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a *transformation*.

Translation lookaside buffer (TLB). A hardware-based address caching mechanism for paging information.

Tree. In the Presentation Manager, the window in the *File Manager* that shows the organization of drives and directories.

truncate. (1) To terminate a computational process in accordance with some rule (A) (2) To remove the beginning or ending elements of a string. (3) To drop data that cannot be printed or displayed in the line width specified or available. (4) To shorten a field or statement to a specified length.

TSR. Terminate-and-stay-resident.

unnamed pipe. A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with *named pipe*.

unordered list. In Information Presentation Facility, a vertical arrangement of items in a list, with each item in the list preceded by a special character or bullet.

update region. A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window that are visually invalid or incorrect, and therefore are in need of repainting.

user interface. Hardware, software, or both that allows a user to interact with and perform operations on a system, program, or device.

User Shell. A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

utility program. (1) A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program. (T) (2) A program designed to perform an everyday task such as copying data from one storage device to another. (A)

U

There are no glossary terms for this starting letter.

V

value set control. A visual component that enables a user to select one choice from a group of mutually exclusive choices.

vector font. A set of symbols, each of which is created as a series of lines and curves. Synonymous with *outline font*. Contrast with *image font*.

VGA. Video graphics array.

viewing pipeline. The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window. A clipping boundary that defines the visible part of model space.

VIO. Video Input/Output.

virtual memory (VM). Synonymous with *virtual storage*.

virtual storage. (1) The storage space that may be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations. (I) (A) (2) Addressable space that is apparent to the user as the processor storage space, from which the instructions and the data are mapped into the processor storage locations. (3) Synonymous with *virtual memory*.

visible region. A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

volume. (1) A file-system driver that uses a block device driver for input and output operations to a local or remote device. (I) (2) A portion of data, together with its data carrier, that can be handled conveniently as a unit.

W

wildcard character. Synonymous with *global file-name character*.

window. (1) A portion of a display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. (A) (2) An area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen.

window class. The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates. A set of coordinates by which a window position or size is defined; measured in device units, or *pels*.

window handle. Unique identifier of a window, generated by Presentation Manager when the window is created, and used by applications to direct messages to the window.

window procedure. Code that is activated in response to a message. The procedure controls the appearance and behavior of its associated windows.

window rectangle. The means by which the size and position of a window is described in relation to the desktop window.

window resource. A read-only data segment stored in the .EXE file of an application o the .DLL file of a dynamic link library.

window style. The set of properties that influence how events related to a particular window will be processed.

window title. In SAA Advanced Common User Access architecture, the area in the title bar that contains the name of the application and the OS/2 operating system file name, if applicable.

workstation. (1) A display screen together with attachments such as a keyboard, a local copy device, or a tablet. (2) (D of C) One or more programmable or nonprogrammable devices that allow a user to do work.

world coordinates. A device-independent Cartesian coordinate system used by the application program for specifying graphical input and output. (I) (A)

world-coordinate space. Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG. What-You-See-Is-What-You-Get. A capability of a text editor to continually display pages exactly as they will be printed.

X

There are no glossary terms for this starting letter.

Y

There are no glossary terms for this starting letter.

Z

z-order. The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming. The progressive scaling of an entire display image in order to give the visual impression of movement of all or part of a display group toward or away from an observer. (I) (A)

8.3 file-name format. A file-naming convention in which file names are limited to eight characters before and three characters after a single dot. Usually pronounced "eight-dot-three." See also *non-8.3 file-name format*.

Index

Special Characters

A

B

C

D

E

F

G

H

I

N

S

U

V

W

X

Y

IBM®

25H7104
G25H-7104-00

Printed on recycled paper.

Printed in U.S.A.

G25H-7104-00

P25H7104