OS/2 2.0 Technical Library

Presentation Manager Programming Reference Volume III

Version 2.00

IBM Programming Family

OS/2 2.0 Technical Library

Presentation Manager Programming Reference Volume III

Version 2.00

IBM Programming Family

Note

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

First Edition (March 1992)

About this Book

The *Presentation Manager Programming Reference* is a detailed technical reference, in three volumes, for application programmers creating programs using the Presentation Manager interface.

Chapter 1 contains important information. You should read it before using this book.

This reference does not give guidance on how to use the functions, nor does it contain information about how the functions are related to each other. It is intended to be used in conjunction with the *Programming Guide Volumes II and III*.

Prerequisite Knowledge

The OS/2 2.0 Technical Library is intended for professional application developers knowledgeable in at least one programming language in which OS/2 programs can be written. The information in the Technical Library assumes that you are new to programming with OS/2 and the Presentation Manager. You should understand the OS/2 services available to users.

Related Publications

The *Application Design Guide* and the *Programming Guide Volumes I, II, and III* introduce the programming concepts that you should understand before you begin developing applications to run on the OS/2 operating system. *Getting Started* describes the online programming books, tools, programming aids, and sample programs that make up the IBM Developer's Toolkit for OS/2 2.0.

Organization of this Book

This book is in three volumes. The contents of each volume are as follows:

Volume I (Functions)

Volume II (Functions and Workplace)

Volume III (Related Information and Data Types)

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 protectible 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 Commercial Relations, IBM Corporation, Purchase, NY 10577.

The following terms, denoted by an asterisk(*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM
Common User Access
CUA
Operating System/2
OS/2
Presentation Manager
SAA
System Application Architecture

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Adobe	Adobe Systems Incorporated
Helvetica	Linotype AG
LaserJet	Hewlett-Packard Company
Intel	Intel Corporation
Microsoft	Microsoft Corporation
PostScript	Adobe Systems Incorporated
Times New Roman	Monotype Corporation
Windows	Microsoft Corporation

Related Information and Data Types

Chapter 10. Functions Supplied by Applications

This chapter describes dialog procedures, window procedures, and hooks. It shows the input parameters and returns that the operating system expects an application to use in application procedures and that can be called by the operating system in response to certain events.

Procedures and hooks are application code that is called by the system in response to certain events.

The names and parameter lists of functions are contained in header files that are incorporated into the application when it is compiled. Their addresses are contained in .LIB files that are incorporated at link time.

The names of procedures and hooks are defined by the application, and their parameter lists are defined by the system. Function prototypes for these procedures and hooks are in PMWIN.H. The prototypes have sample names that can be changed by the programmer before they are inserted into the application source code.

The application passes the address of these procedures and hooks in the following ways:

Dialog procedures	During the WinLoadDlg, WinDlgBox, WinFileDlg, or WinFontDlg function
Window procedures	During the WinRegisterClass or WinSubclassWindow functions
Hooks	During the WinSetHook function
Thunks	During the WinSetClassThunkProc or WinSetWindowThunkProc functions.

The following table shows the procedures and hooks in alphabetic order.

C Name	C Name
Procedures	
DialogProc	WndProc
ThunkProc	
Hooks	
CheckMsgFilterHook	JournalRecordHook
CodePageChangeHook	LoaderHook
DestroyWindowHook	MsgCtlHook
FindWordHook	MsgFilterHook
HelpHook	RegisterUserMsg
InputHook	SendMsgHook
JournalPlaybackHook	

DialogProc — Dialog Procedure

```
#define INCL_WINDIALOGS   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
MRESULT DialogProc (HWND hwnd, USHORT usmsg, MPARAM mpParam1,
                    MPARAM mpParam2)
```

This is a window procedure that automatically subclasses each instance of a dialog box.

Parameters

hwnd (HWND) — input
Handle of the window to which the message applies.

usmsg (USHORT) — input
Message identity.

mpParam1 (MPARAM) — input
Message parameter 1.

mpParam2 (MPARAM) — input
Message parameter 2.

Returns

Message-return data.

Remarks

This procedure is the same as any other window procedure, except that it can receive predefined window messages specific to dialog box windows.

Note: It does *not* receive the WM_CREATE message, but the same information is carried by the WM_INITDLG message, that is generated during the creation of a dialog-box window.

hwnd is always the window handle of the dialog-box window.

The dialog procedure typically processes only some of the messages passed to it. Any messages that it does not process must be passed to WinDefFileDlgProc if the dialog box is the standard file selection dialog, WinDefFontDlgProc if the dialog box is the standard font selection dialog box, or for all other dialog boxes, WinDefDlgProc (not WinDefWindowProc), because these perform the standard dialog-box processing for those messages.

Related Messages

- WM_CREATE
- WM_INITDLG

ThunkProc – Pointer-Conversion Procedure

```
MRESULT ThunkProc (HWND hwnd, USHORT usmsg, MPARAM mpParam1,
                   MPARAM mpParam2, PFNWP pWndProc)
```

This procedure provides pointer conversion for application-defined messages.

Parameters

hwnd (HWND) – input
Window handle.

usmsg (USHORT) – input
Message identity.

This is an application-defined message. The value is greater than or equal to WM_USER.

mpParam1 (MPARAM) – input
Message parameter 1.

mpParam2 (MPARAM) – input
Message parameter 2.

pWndProc (PFNWP) – input
Window-procedure identifier.

Returns

Message-return data.

Remarks

Pointer conversion is normally performed automatically by the operating system. An application needs to provide its own pointer-conversion procedures only for application-defined messages which may be passed from 16-bit code to 32-bit code.

A pointer-conversion procedure is associated with a window by the WinSetWindowThunkProc and WinSetClassThunkProc functions.

The logic of the pointer-conversion procedure is as follows:

1. Convert each message parameter, if necessary. This may include converting any data structures to which the parameter points.
2. Call the window procedure referenced by the *pWndProc* parameter, supplying as arguments *hwnd*, *usmsg*, *mpParam1* and *mpParam2*.
3. Collect the return value and, if necessary, convert it.

   Note that structures to which the return value might point cannot be converted.
4. Convert any structures referenced by message parameters which might have been modified by the window procedure. Note that the pointer-conversion procedure should ensure that the original memory is still available before converting the structures.

A pointer-conversion procedure should process only those messages that it recognizes. On receiving unrecognized messages, it should set *usmsg* to 0.

WndProc — Window Procedure

```
#define INCL_WINMESSAGEMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

MRESULT WndProc  (HWND hwnd, USHORT usmsg, MPARAM mpParam1,
                  MPARAM mpParam2)
```

This defines the window procedure provided by an application.

Parameters

hwnd (HWND) — input
Window handle.

usmsg (USHORT) — input
Message identity.

mpParam1 (MPARAM) — input
Message parameter 1.

mpParam2 (MPARAM) — input
Message parameter 2.

Returns

Message-return data.

Remarks

This procedure is associated with a window by the *pWndProc* of the WinRegisterClass function.

The window procedure typically processes only some of the messages passed to it. Those messages it does not process must be passed on to the WinDefWindowProc function, which performs the standard window processing for those messages.

CheckMsgFilterHook – Check Message Filter Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */
```

BOOL CheckMsgFilterHook (HAB hab, PQMSG pQmsg, USHORT usFirst, USHORT usLast, USHORT fsOptions)

This hook is called whenever WinGetMsg, WinWaitMsg, or WinPeekMsg are used to filter message identities.

Parameters

hab (HAB) – input
Anchor-block handle.

pQmsg (PQMSG) – input
The QMSG data structure of the message currently being reviewed.

usFirst (USHORT) – input
First message identity specified on a call to the WinGetMsg, WinPeekMsg or WinWaitMsg function.

usLast (USHORT) – input
Last message identity specified on a call to the WinGetMsg, WinPeekMsg or WinWaitMsg function.

fsOptions (USHORT) – input
Message removal options:

PM_REMOVE Message is being removed from queue

PM_NOREMOVE Message is not being removed from queue.

Returns

Processing indicator:

TRUE The message is accepted by the filtering. Any further Check Message Filter Hooks in the chain are ignored, any filtering specified by the *ulFirst* and *ulLast* parameters of the WinGetMsg, WinPeekMsg or WinWaitMsg functions are ignored, and processing of the message continues.

A hook that always returns TRUE effectively switches off message filtering.

FALSE The message is passed on to the next Check Message Filter Hook in the chain. If the end of the chain has been reached, the filtering specified by the *ulFirst* and *ulLast* parameters of the WinGetMsg, WinPeekMsg or WinWaitMsg functions is applied.

Remarks

This hook enables an application to apply a very specific message filtering, for example, based on the values of message parameters.

CheckMsgFilterHook – Check Message Filter Hook

This hook is called after window handle filtering and before message filtering. Window handle filtering is controlled by the *hwndFilter* parameter of the WinGetMsg or WinPeekMsg functions. Message filtering is controlled by the *ulFirst* and *ulLast* parameters of the WinGetMsg, WinPeekMsg or WinWaitMsg functions.

This hook is called if the message passes window handle filtering and if non-null message filtering is specified. This means that, on entry to this hook:

- The *hwndFilter* parameter of the WinGetMsg or WinPeekMsg function is either NULLHANDLE or it specifies the window (or a parent of the window) referenced in the *pQmsg* structure.
- At least one of the *usFirst* and *usLast* parameters are nonzero.
- The *msg* field of the *pQmsg* structure might or might not lie inside the range specified by the *usFirst* and *usLast* parameters.

CodePageChangeHook – Code Page Change Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */
```

```
VOID CodePageChangeHook (HMQ hmq, USHORT usOldCodePage,
                         USHORT usNewCodePage)
```

This hook notifies that a message queue code page has been changed.

Parameters

hmq (HMQ) – input
Message-queue handle.

The handle of the message queue that is changing its code page.

usOldCodePage (USHORT) – input
Previous code page.

usNewCodePage (USHORT) – input
New code page.

Returns

The return value is VOID.

Remarks

This hook is sent to all hooks chained under HK_CODEPAGECHANGE, regardless of the return value.

The new code page is set before this hook is called.

DestroyWindowHook – Destroy Window Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */

VOID DestroyWindowHook (HAB hab, HWND Hwnd, ULONG flReserved)
```

This hook is called whenever a window is destroyed.

Parameters

hab (HAB) – input
Anchor-block handle.

Hwnd (HWND) – input
The handle of the window being destroyed.

flReserved (ULONG) – input
Reserved.

Returns

The return value is VOID.

Remarks

This hook is sent after the WM_DESTROY message has been sent and just before the window becomes invalid.

Related Messages

- WM_DESTROY

```
#define INCL_WINHOOKS  /* Or use INCL_WIN or INCL_PM */
```

```
BOOL FindWordHook (USHORT usCodepage, PSZ pszText, ULONG cb, ULONG ich,
                   PULONG pichStart, PULONG pichEnd, PULONG pichNext)
```

This hook allows an application to control where the WinDrawText function breaks a character string that is too long for the drawing rectangle.

Parameters

usCodepage (USHORT) – input
Codepage to use.

This parameter contains the codepage identifier of the string to be formatted.

pszText (PSZ) – input
Text to break.

This parameter contains a pointer to the actual string.

cb (ULONG) – input
Maximum text size.

This parameter contains a value specifying the number of bytes in the string.

ich (ULONG) – input
Break near here.

This parameter contains the index of the character in the string that intersects the right edge of the drawing rectangle.

pichStart (PULONG) – output
Where break began.

This parameter contains the index of the starting character of the intersecting word.

pichEnd (PULONG) – output
Where break ended.

This parameter contains the index of the ending character of the intersecting word.

pichNext (PULONG) – output
Where next word begins.

This parameter contains the index of the starting character of the next word in the string.

Returns

Success indicator:

TRUE If the find-word hook function returns TRUE, WinDrawText will only draw the string up to, but not including, the specified word.

FALSE If the find-word hook function returns FALSE, WinDrawText formats the string in the default manner.

Remarks

The system calls this hook from within the WinDrawText function, if the DT_WORDBREAK flag is set. It lets the application have control of where the function WinDrawText should break for a string that is too long.

HelpHook — Help Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */

BOOL HelpHook (HAB hab, SHORT sMode, SHORT sTopic, SHORT sSubTopic,
               PRECTL prclPosition)
```

This hook processes help requests.

Parameters

hab (HAB) — input
Anchor-block handle.

sMode (SHORT) — input
Help mode.

This has one of the following values, indicating the mode from which help has been requested:

HFM_MENU	Menu mode
HFM_MB	Message-box mode
HFM_WINDOW	Standard (standard window)
HFM_APPLICATION	Application mode.

sTopic (SHORT) — input
Topic identifier.

- In menu mode this is a pull-down window identity
- In message-box mode this is the message-box identity
- In standard mode this is a window identity.

sSubTopic (SHORT) — input
Subtopic identifier.

- In menu mode this is a command identity
- In message-box mode this is a control identity
- In standard mode this is the identity of the window with the focus (**—1** if none).

prclPosition (PRECTL) — input
Rectangle.

This indicates the screen area (in screen coordinates) from where the help was requested. It is provided to enable the help library to avoid covering that area.

- In menu mode it is the bounding rectangle of the selected item, or o the top level menu if value of the *sSubTopic* parameter is **—1**.
- In message-box mode it is the bounding rectangle of the button.
- In standard mode it is the bounding rectangle of the window with the focus, or of the window sent the message if the value of the *sSubTopic* parameter is **—1**.

Note: The data type WRECT can also be used, if supported by the language.

Returns

Indicator as to whether next hook in the chain is called.

The message is always passed to the application.

TRUE	The next hook in the chain is not called.
FALSE	The next hook in the chain is called.

Remarks

This hook can be called directly by an application or in the default-processing associated with windows, menus, and message boxes.

Help-processing is done in two stages. The first stage is the creation of the WM_HELP message. This is done:

- From a WM_CHAR message by ACCELERATOR table translation, when the HELP accelerator option is specified.
- From an action-bar selection, when the MIS_HELP style is specified on the action-bar button.
- From a dialog-box pushbutton, when the BS_HELP style is specified on the pushbutton.
- From a message box, when the MB_HELP style is specified on the message box.

The WM_HELP message is sent to the active window, but will be seen by a modal loop if one is active.

The second stage of processing of help is the processing of the WM_HELP message.

The frame window procedure sees the WM_HELP message because the frame is usually the active window. It processes the WM_HELP message as follows:

- If the window with the focus is the FID_CLIENT frame control, it passes WM_HELP to the FID_CLIENT window.
- If the parent of the window with the focus is the FID_CLIENT frame control, it calls the help hook, specifying:

  sMode = HFM_WINDOW
  sTopic = frame-window id
  sSubTopic = focus-window id.

- If the parent of the focus window is not the FID_CLIENT frame control (for example, it may be the frame itself, or a second-level dialog control), it calls the hook, specifying:

  sMode = HFM_WINDOW
  sTopic = focus-window parent id
  sSubTopic = focus-window id.

The message box window procedure sees the WM_HELP message, because it subclasses the frame window. It processes the WM_HELP message by calling the help hook, specifying:

sMode = HFM_MESSAGE
sTopic = message id
sSubTopic = control id.

HelpHook – Help Hook

The menu window procedure sees the WM_HELP message because it runs a modal loop. It processes the WM_HELP message by calling the help hook, specifying:

sMode = HFM_MENU
sTopic = menu id of pulldown
sSubTopic = menu id of item.

The WinDefWindowProc function sees the WM_HELP message for a FID_CLIENT window if the client does not handle it itself. It calls the help hook, specifying:

sMode = HFM_WINDOW
sTopic = active-window id
sSubTopic = focus-window id.

An application sees the WM_HELP message in its dialog procedure. The application can ignore the WM_HELP message, in which case the frame-window procedure action occurs (as described above) or it can simulate a call to the help hook itself, using:

sMode = HFM_APPLICATION
sTopic = any value
sSubTopic = any value.

The input focus is never given to any of the standard frame controls, so help for these cannot be obtained.

Related Messages

- WM_CHAR
- WM_HELP

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */
```

BOOL InputHook (HAB hab, PQMSG pQmsg, USHORT fsOptions)

This hook filters messages from the input queue.

Parameters

hab (HAB) – input
Anchor-block handle.

pQmsg (PQMSG) – input
A QMSG data structure.

fsOptions (USHORT) – input
Message removal options:

PM_REMOVE	Message is being removed from queue
PM_NOREMOVE	Message is not being removed from queue.

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.

Remarks

This hook is called when messages are removed from an application queue, before being returned by WinGetMsg or WinPeekMsg. It is called from within these functions just before resuming the application with the message that is returned. There are no restrictions on calls that may be made at this time.

JournalPlaybackHook – Journal Playback Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */

LONG JournalPlaybackHook (HAB hab, BOOL fSkip, PQMSG pqmsg)
```

This hook plays back recorded messages.

Parameters

hab (HAB) – input
Anchor-block handle.

fSkip (BOOL) – input
Indicator as to whether the next message should be played back:

TRUE The journal playback hook skips to the next message. The *pqmsg* parameter is NULL in this case. The next hook in the chain is not called.

FALSE The journal playback hook returns the next available message. The same message is returned each time, until it is skipped with a call where this parameter is TRUE.

pqmsg (PQMSG) – input
Data structure where the message to be played back is returned.

When this hook is called, the *time* field of the QMSG structure is initialized to the current time. This can be used to determine whether the next message is ready or not. This value must be used for any delta calculations performed by the hook procedure, rather than the result of WinGetCurrentTime

Returns

Waiting time.

The time to wait (in milliseconds) before processing the current message.

Remarks

This hook is called whenever a message is required to be played back.

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */
```

BOOL JournalRecordHook (HAB hab, PQMSG pqmsg)

This hook records user-input messages.

Parameters

hab (HAB) – input
Anchor-block handle.

pqmsg (PQMSG) – input
Data structure that contains the message to be recorded.

The *hwnd* field of the QMSG structure is also set when the hook is called.

Returns

The return value from this hook is ignored.

Remarks

This hook is called *after* raw input is translated to WM_CHAR or WM_BUTTON1DBLCLK messages.

The next hook in the chain is always called, and the message is always passed to the application.

JournalPlaybackHook hook does not receive any input played back by this hook. This prevents feedback situations where input is played back a number of times.

Related Messages

- WM_CHAR
- WM_BUTTON1DBLCLK

LoaderHook — Loader Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL LoaderHook (HAB hab, SHORT sContext, PSZ pszlibname, PHLIB phlibLibhandle,
                 PSZ pszprocname, PFN pwndproc, PBOOL pfSuccess)
```

This hook allows the library and procedure loading and deleting calls to be intercepted.

Parameters

hab (HAB) — input
Anchor-block handle.

sContext (SHORT) — input
Origin of call to hook.

LHK_DELETELIB WinDeleteLibrary

LHK_DELETEPROC WinDeleteProcedure

LHK_LOADLIB WinLoadLibrary

LHK_LOADPROC WinLoadProcedure

pszlibname (PSZ) — input
Library name.

This is the same as the library name in the *pszLibname* parameter of the WinLoadLibrary function.

phlibLibhandle (HLIB) — input/output
Library handle.

This is the same as the library handle in the *hlibLibhandle* parameter of the WinLoadProcedure function or the *hlibLibhandle* parameter of the WinDeleteLibrary function.

If the *sContext* parameter is set to LHK_LOADLIB, then this hook must set the value of this parameter to the handle of the loaded library or to NULLHANDLE if the load fails.

pszprocname (PSZ) — input
Procedure name.

This is the same as the procedure name in the *pszProcname* parameter of the WinLoadProcedure function.

pwndproc (PFN) — input
Window procedure identifier.

This is the same as the library name in the *pwndproc* parameter of the WinDeleteProcedure function.

If the *sContext* parameter is set to LHK_LOADPROC, then this hook must set the value of this parameter to the handle of the loaded procedure or to NULL if the load fails.

pfSuccess (PBOOL) — input/output
Success indicator:

TRUE Library or procedure loaded or deleted successfully.

FALSE Library or procedure not loaded or deleted successfully.

Returns

Processing indicator:

TRUE Do not call next hook in chain

FALSE Call next hook in chain.

Remarks

If the hook attempts a load or deletion which is unsuccessful, then the hook must establish the relevant error information.

MsgCtlHook – Message Control Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL MsgCtlHook (HAB hab, SHORT sContext, HWND hwnd, PSZ pszClassName,
                 USHORT usMsgClass, SHORT sControl, PBOOL pfSuccess)
```

This hook allows the call which determine the flow of messages to be intercepted.

Parameters

hab (HAB) – input
Anchor-block handle.

sContext (SHORT) – input
Origin of call to hook.

MCHK_CLASSMSGINTEREST	WinSetClassMsgInterest
MCHK_MSGINTEREST	WinSetMsgInterest
MCHK_MSGMODE	WinSetMsgMode
MCHK_SYNCHRONISATION	WinSetSynchroMode

hwnd (HWND) – input
Window handle.

This is the same as the window handle in the *hwnd* parameter of the WinSetMsgInterest function.

pszClassName (PSZ) – input
Window class name.

This is the same as the window class name in the *pszClassName* parameter of the WinSetClassMsgInterest function.

usMsgClass (USHORT) – input
Message class.

This is the same as the message class in the *ulMsgClass* parameter of the WinSetMsgInterest and the WinSetClassMsgInterest functions.

sControl (SHORT) – input
Control setting.

The setting varies with the value of the *sContext* parameter.

For MCHK_CLASSMSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_AUTODISPATCH.

For MCHK_MSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_RESET, or SMI_AUTODISPATCH.

For MCHK_MSGMODE, it can be SMD_DELAYED or SMD_IMMEDIATE.

For MCHK_SYNCHRONISATION, it can be SSM_SYNCHRONOUS, or SSM_ASYNCHRONOUS, or SSM_MIXED.

pfSuccess (PBOOL) – input/output
Success indicator:

TRUE	Mode or interest successfully set.
FALSE	Mode or interest not successfully set.

Returns

Processing indicator:

TRUE Do not call next hook in chain

FALSE Call next hook in chain.

Remarks

If the hook is unable to alter the message control state, then the hook must establish the relevant error information.

MsgFilterHook – Message Filter Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */

BOOL MsgFilterHook (HAB hab, PQMSG pQmsg, USHORT usContext)
```

This hook filters messages from inside a mode loop.

Parameters

hab (HAB) – input
Anchor-block handle.

pQmsg (PQMSG) – input
A queue message data structure.

usContext (USHORT) – input
Context in which the hook has been called:

MSGF_DIALOGBOX	Dialog-box mode loop.
MSGF_MESSAGEBOX	Message-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

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.

Remarks

This hook is called inside any of the system-mode loops, for instance, during size-tracking or move-tracking, or while a dialog box or menu is displayed.

The WM_QUIT message is passed to this hook, if it occurs during a mode loop.

Related Messages

- WM_QUIT

RegisterUserMsg — Register User Message Hook

```
#define INCL_WINHOOKS  /* Or use INCL_WIN or INCL_PM */

BOOL RegisterUserMsg (HAB hab, SHORT sContext, USHORT usMsgid, SHORT sType1,
                      SHORT sDir1, SHORT sType2, SHORT sDir2, SHORT sTyper,
                      SHORT sCount, PSHORT asTypes, PBOOL pfSuccess)
```

This hook allows user messages and user data types to be registered.

Parameters

hab (HAB) — input
Anchor-block handle.

sContext (SHORT) — input
Origin of call to hook.

RUMHK_DATATYPE Register User Data type

RUMHK_MSG Register User Message

usMsgid (USHORT) — input
Message identifier.

If the origin of the call is 'Register User Data Type', this parameter is not set.

sType1 (SHORT) — input
Data type of message-parameter 1.

If the origin of the call is 'Register User Data Type', this parameter contains the data type code to be registered.

sDir1 (SHORT) — input
Direction of message-parameter 1.

If the origin of the call is 'Register User Data Type', this parameter is not set.

sType2 (SHORT) — input
Data type of message-parameter 2.

If the origin of the call is 'Register User Data type', this parameter is not set.

sDir2 (SHORT) — input
Direction of message-parameter 2.

If the origin of the call is 'Register User Data Type', this parameter is not set.

sTyper (SHORT) — input
Data type of message reply.

If the origin of the call is 'Register User Data Type', this parameter is not set.

sCount (SHORT) — input
Number of elements.

If the origin of the call is 'Register User Message', this parameter is not set.

asTypes (PSHORT) — input
Data types of structure components.

If the origin of the call is 'Register User Message', this parameter is not set.

pfSuccess (PBOOL) — input/output
Success indicator:

TRUE Successful completion

FALSE Error occurred.

RegisterUserMsg – Register User Message Hook

Returns

Processing indicator:

TRUE Do not call next hook in chain

FALSE Call next hook in chain.

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */
```

VOID SendMsgHook (HAB hab, PSMHSTRUCT psmhssmh, BOOL fInterTask)

This hook filters messages sent by the WinSendMsg function.

Parameters

hab (HAB) – input
Anchor-block handle.

psmhssmh (SMHSTRUCT) – input
Send message hook structure.

This parameter is a structure that contains the parameters to the WinSendMsg function.

fInterTask (BOOL) – input
Intertask indicator:

TRUE The message is sent between tasks (intertask)

FALSE The message is sent within a task (intratask).

Returns

The return value is VOID.

Remarks

This hook may be called whenever a window procedure is called via the WinSendMsg function.

It is called in the context of the sender, whereby if the sender has a queue hook installed it is called, but if the receiver has a queue hook installed it is not called.

The next hook in the chain is always called.

Chapter 11. Introduction to Message Processing

Messages are processed by window and dialog procedures.

Every window has a window procedure. Windows can also be combined into standard windows or dialog boxes. These are special cases of groups of windows that also have their own procedures. A window or dialog procedure must be capable of processing any message. This can be achieved by delegating some message types to the default window, or dialog, procedures by use of the WinDefWindowProc and WinDefDlgProc functions respectively.

Control windows are a special type of child windows. They take the form of objects such as buttons, scroll bars, list boxes, and text entry fields. These child windows process mouse and keyboard input and notify its owner of significant input events. Procedures for these child window controls are inside the Presentation Manager and are often called system-provided window procedures.

All messages have the following form:

QMSG Message structure.

```
typedef struct _QMSG {
HWND     hwnd;
ULONG    msg;
MPARAM   mp1;
MPARAM   mp2;
ULONG    time;
POINTL   ptl;
 } QMSG;
```

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.

Message Types

There are two types of window procedure message processing:

- Default window and dialog procedure message processing
- Control window message processing.

These types are described below along with the notation conventions used in the message descriptions. The messages are described in the following chapters.

Default Window and Dialog Procedure Message Processing

These window procedures provide default processing for application window procedures:

- Default window and dialog procedure
- Language support window and dialog procedures, which are used if the application specifies a null window procedure
- Default AVIO window procedure.

These messages are described in Chapter 12, "Default Window Procedure Message Processing" on page 12-1. The system-provided window procedures take no action on messages that are not defined in this chapter, and return NULL.

Control Window Message Processing

Controls are predefined classes of child windows that any application can use for input and output. These control classes are predefined:

WC_BUTTON Consists of buttons and boxes that the operator can select by clicking the pointing device or using the keyboard. These messages are described in Chapter 13, "Button Control Window Processing" on page 13-1.

WC_COMBOBOX Consists of an entry field control and a list box control merged into a single control. The list, which is usually limited in size, is displayed below the entry field and offset one dialog box unit to its right. These messages are described in Chapter 19, "Prompted Entry Field Control Window Processing" on page 19-1.

WC_CONTAINER Consists of a visual component whose specific purpose is to hold objects such as executable programs, word processing files, graphics images, and database records. Messages are described in Chapter 24, "Container Control Window Processing" on page 24-1.

WC_ENTRYFIELD Consists of a single line of text that the operator can edit. These messages are described in Chapter 14, "Entry Field Control Window Processing" on page 14-1.

WC_FRAME Consists of a composite window. These messages are described in Chapter 15, "Frame Control Window Processing" on page 15-1.

WC_LISTBOX Presents a list of text items from which the operator can make selections. These messages are described in Chapter 16, "List Box Control Window Processing" on page 16-1.

WC_MENU Presents a list of items, which may be text displayed horizontally as action bars or vertically as pull-down menus. Menus are usually used to provide a command interface to applications. These messages are described in Chapter 17, "Menu Control Window Processing" on page 17-1.

WC_MLE Consists of a rectangular window that displays multiple lines of text that the operator can edit. When it has the focus, the cursor marks the current **insertion** or **replacement** point. These messages are described in Chapter 18, "Multi-Line Entry Field Control Window Processing" on page 18-1.

WC_NOTEBOOK Consists of a visual component whose specific purpose is to organize information on individual pages so that a user can find and display that information quickly and easily. Messages are described in Chapter 25, "Notebook Control Window Processing" on page 25-1.

WC_SCROLLBAR Consists of window scroll bars that allow the operator to make a request to scroll the contents of an associated window. These messages are described in Chapter 20, "Scroll Bar Control Window Processing" on page 20-1.

WC_SLIDER Consists of a visual component whose specific purpose is to allow a user to set, display, or modify a value by moving the slider arm along the slider shaft. Messages are described in Chapter 26, "Slider Control Window Processing" on page 26-1.

WC_SPINBUTTON Presents a scrollable ring of choices from which the operator can select. These messages are described in Chapter 21, "Spin Button Control Window Processing" on page 21-1.

WC_STATIC Consists of simple display items that do not respond to keyboard or pointing device events. These messages are described in Chapter 22, "Static Control Window Processing" on page 22-1.

WC_TITLEBAR Displays the window title or caption and allows the operator to move its owner. These messages are described in Chapter 23, "Title Bar Control Window Processing" on page 23-1.

WC_VALUESET Consists of a visual component whose specific purpose is to allow a user to select one choice from a group of mutually exclusive choices. A value set can use graphical images (bit maps or icons), as well as colors, text, and numbers, to represent the items that a user can select. Messages are described in Chapter 27, "Value Set Control Window Processing" on page 27-1.

Owner-Notification Messages: Controls are useful because they notify their owners when significant events take place. A control notifies its owner by sending a WM_CONTROL message or by posting a WM_COMMAND or WM_HELP message.

- WM_CONTROL
- WM_COMMAND

  Param2 contains information that indicates the source of the WM_COMMAND message:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control
CMDSRC_MENU	Posted by a menu control
CMDSRC_ACCELERATOR	Posted by WinTranslateAccel
CMDSRC_FONTDLG	Posted by a font dialog.
CMDSRC_OTHER	Other source.

- WM_HELP

  Param2 contains information that indicates the source of the WM_HELP message:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control
CMDSRC_MENU	Posted by a menu control
CMDSRC_ACCELERATOR	Posted by WinTranslateAccel
CMDSRC_OTHER	Other source.

Notation Conventions

Each message description contains:

Name The message name; a 2-byte identity unique to a message. Messages generated by the system have an identity below the constant WM_USER; see "Reserved Messages" on page 12-1.

Applications generating their own messages must use a value higher than WM_USER.

For all messages, the first two or three characters of the name indicate the type of window that is related to the message; for example:

LM	List box control
SBM	Scroll bar control.

Cause The principal reason that caused the generation of the message.

Parameters Input and output parameters pertinent to the message.

There are always two parameters (**param1** and **param2**) and one **return** value. Any or all of the parameters can be NULL.

Remarks An explanation of the relationship between the parameters in the context of the message and an indication of the expected processing of the message.

Default A definition of how the default window procedures (provided by the system) process the message.

Note: A message is not equivalent to a call of the same name.

Chapter 12. Default Window Procedure Message Processing

This system-provided window procedure processes the actions that control the operation of windows.

Purpose

General window messages are used for standard processing. These messages can be requested from the system or sent to the system for information, or for actions such as create window, validate window, track mouse movement, and select and deselect actions.

Reserved Messages

These message ranges are reserved:

WM_USER All messages below this value are reserved for system use. Private messages should have an identifier with a value of WM_USER or higher.

General Window Styles

The *window* is the mechanism by which the application communicates with the operator. Each window can have a window *style* that controls the appearance and behavior of the window. There are also *class* styles that apply to all the windows of a particular class (class being FRAME, BUTTON, and so on).

Window Class Styles

These window class styles are available:

CS_SIZEREDRAW Determines whether a window will be redrawn when sized. This style is to be used for a window whose contents are sensitive to the size of the window. For example, the data in some windows can be scaled up or down to fit the size of the Client Area. In other windows, the data remains the same size whatever the size of the window; it is merely clipped if the window is made smaller. The CS_SIZEREDRAW style is to be used in the first instance but not in the second. For more information, see WM_CALCVALIDRECTS.

CS_SYNCPAINT Window is synchronously repainted. This style causes WS_SYNCPAINT to be set for all windows of this class.

CS_MOVENOTIFY This class style should be used by a child window if it wants to be notified with a WM_MOVE message when its parent is moved. For more detail, see the WM_MOVE message description.

CS_CLIPCHILDREN Causes a window of style WS_CLIPCHILDREN to be created, regardless of whether this style bit is specified on the create window function.

CS_CLIPSIBLINGS Causes a window of style WS_CLIPSIBLINGS to be created, regardless of whether this style bit is specified on the create window function.

CS_PARENTCLIP Causes a window of style WS_PARENTCLIP to be created, regardless of whether this style bit is specified on the create window function.

CS_SAVEBITS Causes a window of style WS_SAVEBITS to be created, regardless of whether this style bit is specified on the create window function.

CS_PUBLIC Causes a public window class to be registered. It is an error if this parameter is specified on any process other than the shell process.

CS_HITTEST If set, causes a WM_HITTEST message to be sent to the window, before sending any pointing device message.

If not set, no WM_HITTEST message is sent, and it is assumed that the window returns HT_NORMAL if the window is not disabled, and HT_ERROR if the window is disabled.

Top-level frame windows do not have CS_HITTEST set.

CS_FRAME If set, all windows of this class are expected to behave as frame windows.

Window Styles

These window styles are available:

WS_SYNCPAINT Window is synchronously repainted.

This style is set for windows that have Class Style CS_SYNCPAINT. Applications can then turn this style on and off to vary the window processing.

System-Provided Window Styles:

WS_CLIPCHILDREN This specifies that the area occupied by the children of a window is to be excluded when drawing in that window. Normally, it is included.

WS_CLIPSIBLINGS This specifies that the area occupied by the siblings of a window is to be excluded when drawing in that window. Normally, it is included.

WS_DISABLED This specifies that the window is disabled. The default is enabled.

WS_MAXIMIZED This specifies that the frame window is to be created maximized.

When a window is moved or sized in the normal way at least one border should remain on the screen. When a window is maximized and the maximum size is as large as the screen all borders should be positioned just outside the screen.

WS_MINIMIZED This specifies that the frame window is to be created minimized.

WS_PARENTCLIP This controls how a window is clipped when a drawing action takes place into the window.

Generally, a WS_PARENTCLIP window is not to draw outside its window rectangle.

WS_SAVEBITS This specifies that the screen image of the area under a window of this style be saved when the window is made visible.

WS_VISIBLE This specifies that the window is visible. The default is invisible.

Note: A window can still be visible, in this sense, even if it cannot be seen because it is covered by other windows.

Styles for Windows in Dialogs

WS_GROUP This identifies the dialog items that make up a group.

This style is to be specified on the first window of any group. Subsequent windows of the group must not have this style. The windows of the group must be adjacent siblings. This can be done by listing the windows consecutively in templates (see "Dialog Template" on page 32-19) or by inserting each new window in the group behind the previous one (WinCreateWindow).

WS_TABSTOP This identifies a dialog item as one to which the operator can TAB.

General Window Messages

This section describes the window procedure actions upon receiving the following messages.

PL_ALTERED

This message is broadcast to all frame windows when the PrfReset function is issued.

Parameters

param1

hiniUser *(HINI)*
Handle of the new user profile.

param2

hiniSystem *(HINI)*
Handle of the new system profile.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, must be 0.

Remarks

Applications should refresh their defaults from the user or system profile.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_ACTIVATE

This message occurs when an application causes the activation or deactivation of a window.

Parameters

param1

usactive *(USHORT)*
Active indicator:

TRUE The window is being activated
FALSE The window is being deactivated.

param2

hwndhwnd *(HWND)*
Window handle.

In the case of activation, *hwndhwnd* identifies the window being activated. In the case of deactivation, *hwndhwnd* identifies the window being deactivated.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value.

Remarks

A deactivation message (that is, a WM_ACTIVATE message with *usactive* set to FALSE) is sent first to the window procedure of the main window being deactivated, before an activation message (that is, a WM_ACTIVATE message with *usactive* set to TRUE) is sent to the window procedure of the main window being activated.

Any WM_SETFOCUS messages with *usfocus* set to FALSE, are sent before the deactivation message. Any WM_SETFOCUS messages with *usfocus* set to TRUE, are sent after the activation message.

If WinSetFocus is called during the processing of a WM_ACTIVATE message, a WM_SETFOCUS message with *usfocus* set to FALSE is not sent, as no window has the focus.

If a window is activated before any of its children have the focus, this message is sent to the frame window or to its FID_CLIENT, if it exists.

Note: Except in the instance of a WM_ACTIVATE message, with *usactive* set to TRUE, an application processing a WM_ACTIVATE, or a WM_SETFOCUS message should not change the focus window or the active window. If it does, the focus and active windows must be restored before the window procedure returns from processing the message. For this reason, any dialog boxes or windows brought up during the processing of a WM_ACTIVATE, or a WM_SETFOCUS message should be system modal.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_APPTERMINATENOTIFY

This message is posted when an application (started by another application) terminates.

Parameters

param1

happhapp *(HAPP)*
Application handle.

param2

flretcode *(ULONG)*
Return code from the terminating application.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value; must be 0.

Remarks

The WM_APPTERMINATENOTIFY message provides the capability for the starting application to be notified when the started application terminates.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_ADJUSTWINDOWPOS

This message is sent by the WinSetWindowPos call to enable the window to adjust its new position or size whenever it is about to be moved.

Parameters

param1

pIpswp *(PSWP)*
SWP structure pointer.

The structure has been filled in by the WinSetWindowPos function with the proposed move or size data. The control can adjust this new position by changing the contents of the SWP structure. It can change the *x* or *y* fields to adjust its new position; or the *cx* or *cy* fields to adjust its new size, or the *hwndInsertBehind* field to adjust its new z-order.

param2

fIzero *(ULONG)*
Zero.

Returns

reply

fIResult *(ULONG)*
Window-adjustment status indicators.

These indicators are passed on to the WM_WINDOWPOSCHANGED message that is sent after the window state change has occurred. Bits 0 through 15 of this parameter are reserved for system use and bits 16 through 31 are available for application use.

0	No changes have been made
AWP_MINIMIZED	The frame window has been minimized.
AWP_MAXIMIZED	The frame window has been maximized.
AWP_RESTORED	The frame window has been restored.
AWP_ACTIVATE	The frame window has been activated.
AWP_DEACTIVATE	The frame window has been deactivated.

Remarks

Frame controls can respond to this message to reposition themselves or resize themselves in the window frame.

Menu controls respond to this message as follows:

MS_ACTIONBAR not specified: The SWP *cx* and SWP *cy* fields are set so that the menu window exactly contains all of the items in the menu. The SWP *x* and SWP *y* fields are not changed.

MS_ACTIONBAR specified and MS_TITLEBUTTON not specified: The items in the menu are arranged such that all of the items are visible within the width specified by the SWP *cx* field. This formatting may cause the menu items to be arranged in multiple lines. The SWP *cx* field is set to include all of the lines of the menu. The SWP *x* and SWP *y* fields are not changed.

MS_ACTIONBAR specified and MS_TITLEBUTTON specified: The SWP *cx* value is set to the accumulated width of the items in the menu. The height specified in the SWP *cy* field is not changed. In both instances, the SWP *cx* and SWP *cy* fields are only altered if SWP_SIZE is specified in the *fl* field. Instead, the width of MS_TITLEBUTTON menus is determined by the accumulated width of the items in the menu.

A list box does two things:

- Changes the height so as to accommodate an exact number of items.
- Automatically outsets its border. This means, for example, that the x, y, width, and height fields in the resource file specify the working area of the listbox. The border is drawn outside this area.

The entry field control, if ES_MARGIN is specified, outsets its margin. This means that in the resource file, the numbers specified as the x-, and y-position of an entry field control are taken to be the position where the first character of text is drawn, not where the lower-left corner of the surrounding box is drawn. Similarly, the height and width parameters apply to the editable area of the control; consequently, they do not include the margin.

When a dialog is created with WinCreateDlg or WinLoadDlg, a WM_ADJUSTWINDOWPOS message is sent to each child window after the dialog window is created, with a pointer to a SWP structure containing *fl* equal to SWP_SIZE | SWP_MOVE and the *x*, *y*, *cy*, and *cx* fields initialized to the current size and position of the window. The message enables the control to adjust its size or position, usually to compensate for its border, or margin, or both.

Default Processing

The default window procedure takes no action on this message, other than to set *flResult* to 0.

WM_BEGINDRAG

This message occurs when the operator initiates a drag operation.

Parameters

param1

usPointer *(USHORT)*
Input device flag:

TRUE Message resulted from pointer event
FALSE Message resulted from keyboard event

param2

ptspointerpos *(POINTS)*
Pointer position

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *usPointer* is not set to TRUE.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed.
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_BEGINDRAG.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_BEGINSELECT

This message occurs when the operator initiates a swipe selection.

Parameters

param1

usPointer *(USHORT)*
Input device flag:

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event

param2

ptspointerpos *(POINTS)*
Pointer position

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *usPointer* is not set to TRUE.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE	Message processed.
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_BEGINSELECT.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_BUTTON1CLICK

This message occurs when the operator presses and then releases button 1 of the pointing device within a specified period of time, and without moving the mouse.

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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *fresult* to FALSE.

WM_BUTTON2CLICK

This message occurs when the operator presses and then releases button 2 of the pointing device within a specified period of time, and without moving the mouse.

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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *fresult* to FALSE.

WM_BUTTON3CLICK

This message occurs when the operator presses and then releases button 3 of the pointing device within a specified period of time, and without moving the mouse.

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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *fresult* to FALSE.

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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

A double-click is recognized if all of the following are true:

- Two clicks are of the same button.
- No intervening pointing device button is pressed.
- The two clicks occur within the double-click time interval as defined by the SV_DBLCLKTIME system value.
- The two clicks occur within a small spatial distance. This is defined by the rectangle, the length of whose sides parallel to the x- and y-axes are respectively, the SV_CXDBLCLICK and SV_CYDBLCLICK system values. The first click is assumed to be at the center of this rectangle.

The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *fresult* to FALSE.

WM_BUTTON2DBLCLK

This message occurs when the operator presses button 2 of the pointing device twice within a specified time, as detailed in "WM_BUTTON1DBLCLK" on page 12-10.

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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information. The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure processes this message identically to WM_BUTTON1DBLCLK.

WM_BUTTON3DBLCLK

This message occurs when the operator presses button 3 of the pointing device twice within a specified time, as detailed in "WM_BUTTON1DBLCLK" on page 12-10.

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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information. The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure processes this message identically to WM_BUTTON1DBLCLK.

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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

It is the responsibility of the application to ensure that the appropriate frame window is activated and that the focus is to the appropriate window, by using the WinSetFocus function. The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure activates the window using WinSetActiveWindow, and then sets *fresult* to FALSE.

WM_BUTTON1MOTIONEND

This message occurs when the operator completes a drag operation which was initiated by pressing button one on the pointing device.

Parameters

param2

fshittestres *(USHORT)*
Hit-test result.

hittestres 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.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE	Message processed.
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *result* to FALSE.

WM_BUTTON1MOTIONSTART

This message occurs when the operator initiates a drag operation by moving the mouse while pressing button one on the pointing device.

Parameters

param2

fshittestres *(USHORT)*
Hit-test result.

hittestres 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.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE	Message processed.
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *result* to FALSE.

WM_BUTTON2DOWN

This message occurs when the operator presses button 2 on 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 determined the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information.

It is the responsibility of the application to ensure that the appropriate frame window is activated and that the focus is to the appropriate window, by using the WinSetFocus function. The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure processes this message identically to "WM_BUTTON1DOWN" on page 12-13.

WM_BUTTON2MOTIONEND

This message occurs when the operator completes a drag operation which was initiated by pressing button two on the pointing device.

Parameters

param2

fshittestres *(USHORT)*
Hit-test result.

hittestres 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.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed.
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *result* to FALSE.

WM_BUTTON2MOTIONSTART

This message occurs when the operator initiates a drag operation by moving the mouse while pressing button two on the pointing device.

Parameters

param2

fshittestres *(USHORT)*
Hit-test result.

hittestres 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.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed.
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *result* to FALSE.

WM_BUTTON3DOWN

This message occurs when the operator presses button 3 on 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 determined the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information.

It is the responsibility of the application to ensure that the appropriate frame window is activated and that the focus is to the appropriate window, by using the WinSetFocus function. The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure processes this message identically to "WM_BUTTON1DOWN" on page 12-13.

WM_BUTTON3MOTIONEND

This message occurs when the operator completes a drag operation which was initiated by pressing button three on the pointing device.

Parameters

param2

fshittestres *(USHORT)*
Hit-test result.

hittestres 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.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE	Message processed.
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *result* to FALSE.

WM_BUTTON3MOTIONSTR

This message occurs when the operator initiates a drag operation by moving the mouse while pressing button three on the pointing device.

Parameters

param2

fshittestres *(USHORT)*
Hit-test result.

hittestres 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.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE	Message processed.
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *result* to FALSE.

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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information. The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set *fresult* to FALSE.

WM_BUTTON2UP

This message occurs when the operator releases button 2 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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information. The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set *fresult* to FALSE.

WM_BUTTON3UP

This message occurs when the operator releases button 3 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" on page 12-37.

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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information. The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure processes this message identically to WM_BUTTON1UP.

WM_CALCFRAMERECT

This message occurs when an application uses the WinCalcFrameRect function.

Parameters

param1

pRect *(PRECTL)*
Rectangle structure.

This points to a RECTL structure.

param2

usFrame *(USHORT)*
Frame indicator:

TRUE	Frame rectangle provided
FALSE	Client area rectangle provided.

Returns

reply

fSuccess *(BOOL)*
Rectangle-calculated indicator:

TRUE	Successful completion
FALSE	Error occurred or the calculated rectangle is empty.

Remarks

This message is sent to the frame control to perform the appropriate calculation. If the low word of MP2 is TRUE, the RECTL structure in MP1 contains a frame window and this message calculates the RECTL of the client. If the low word of MP2 is FALSE, MP1 contains a client window and this message calculates the RECTL of the frame.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

WM_CALCVALIDRECTS

This message is sent from WinSetWindowPos and WinSetMultWindowPos to determine which areas of a window can be preserved if a window is sized, and which should be redisplayed.

Parameters

param1

pOldNew *(PRECTL)*
Window-rectangle structures.

This points to two RECTL structures. The first structure contains the rectangle of the window before the move, the second contains the rectangle of the window after the move. The coordinates of the rectangles are relative to the parent window.

param2

pNew *(PSWP)*
New window position.

This points to a SWP structure that contains information about the window after it is resized (see the WinSetWindowPos function).

Returns

reply

usAlign *(USHORT)*
Alignment control.

This instructs WinSetWindowPos how to align valid window bits. This value is made up from CVR_* flags, as follows:

CVR_ALIGNLEFT	Align with the left edge of the window.
CVR_ALIGNBOTTOM	Align with the bottom edge of the window.
CVR_ALIGNTOP	Align with the top edge of the window.
CVR_ALIGNRIGHT	Align with the right edge of the window.
CVR_REDRAW	The whole window is invalid. If CVR_REDRAW, is set, the whole window is assumed invalid, otherwise, the remaining flags can be ORed together to get different kinds of alignment. For example: `(CVR_ALIGNLEFT | CVR_ALIGNTOP)` aligns the valid window area with the top-left of the window.
0	It is assumed the application has changed the rectangles pointed to by *pOldNew* and *pNew* itself.

Remarks

This message is *not* sent if this window has the CS_SIZEREDRAW style, indicating size-sensitive window content that must be totally redrawn if sized.

This enables the application to determine if the position of the window has changed as well as its size; this can aid alignment processing.

These rectangles can be modified by the window procedure to cause parts of the window to be redrawn and not preserved.

The window manager tries to preserve the screen image by copying the image described by the old rectangle into the image described by the new rectangle. In this way, an application can control the alignment of the preserved image as well, by changing the origin of the first rectangle.

If no change is made to either rectangle, the entire window area is preserved. If either rectangle is empty, the entire window area is completely redrawn by the operation.

Note: This functionality can be used to optimize window updating when the window is resized. For example, if the application returns that the window is to be aligned with the top-left corner, and the top border is sized, the screen data of the window moves with the top border.

In all instances, the rectangles are intersected with the area of the screen that is actually visible and the valid area of the window. That is, only the window area that contains window information is copied.

For example, consider an application that has two scroll bars, that are children of the client window. When the window is resized, the scroll bars must be completely redrawn. By returning rectangles that exclude the scroll bars, the area of the scroll bars is completely redrawn, thereby preserving only the part of the screen that is worth preserving.

Default Processing

The default window procedure processing is to align the valid area with the top-left of the window by returning:

`(CVR_ALIGNTOP | CVR_ALIGNLEFT)`

In addition, any child windows intersecting the source rectangle pointed to by *pOldNew* of this message, are also offset with the aligned window area.

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.
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

reply

fresult *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the queue associated with the window that has the focus.

The set of keys that causes a WM_CHAR message is device-dependent.

When this message is processed, precedence should normally be given to a valid virtual key if there is one contained in the message.

There are several instances when a window procedure may receive this message with the KC_KEYUP bit set, although it did not receive this message for the down transition of the key.

For example,

- The down transition of the key is translated by the function WinTranslateAccel, into a WM_COMMAND, WM_SYSCOMMAND, WM_HELP, or a WM_NULL message.
- The key down causes the input focus to change (tab to another window, dismiss a dialog, exit a program, and so on).
- Some other event happens that changes the focus between the time that the key is pressed down and the time that it is released.

Applications should normally only process WM_CHAR messages that do not have the KC_KEYUP bit set.

Except for the special instance where the LONEKEY flag is set on an accelerator key definition, all translations are done on the down stroke of the character.

When the current character is a double-byte character then *param2* contains both bytes of the double-byte character. These bytes are in the order CHAR1FROMMP, CHAR2FROMMP. When the current character is a single-byte character, CHAR2FROMMP contains 0.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set *fresult* to FALSE.

WM_CHORD

This message occurs when the operator presses both button one and button two on the pointing device.

Parameters

param2

fshittestres *(USHORT)*
Hit-test result.

hittestres 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.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed.
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *result* to FALSE.

WM_CLOSE

This message is sent to a frame window to indicate that the window is being closed by the user.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

This message is sent by the frame to itself as a result of receiving a WM_SYSCOMMAND message with SC_CLOSE code set. If this message is passed to WinDefDlgProc, this function calls WinDismissDlg and passes the DID_CANCEL result code to it.

Default Processing

The default window procedure posts a WM_QUIT message to the appropriate queue and sets *fireply* to 0.

WM_COMMAND

This message occurs when a control has a significant event to notify to its owner, or when a key stroke has been translated by an accelerator table.

Parameters

param1

uscmd *(USHORT)*
Command value.

It is the responsibility of the application to be able to relate *uscmd* to an application function.

param2

ussource *(USHORT)*
Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control. *uscmd* is the window identity of the pushbutton.
CMDSRC_MENU	Posted by a menu control. *uscmd* is the identity of the menu item.
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. *uscmd* is the accelerator command value.
CMDSRC_FONTDLG	Font dialog. *uscmd* is the identity of the font dialog.
CMDSRC_FILEDLG	File dialog. *uscmd* is the identity of the file dialog.
CMDSRC_OTHER	Other source. *uscmd* gives further control-specific information defined for each control type.

uspointer *(USHORT)*
Pointer-device indicator:

TRUE	The message is posted as a result of a pointer-device operation.
FALSE	The message is posted as a result of a keyboard operation.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value.

Remarks

This message is posted to the queue of the owner of the control.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_CONTEXTMENU

This message occurs when the operator requests a pop-up menu.

Parameters

param1

usPointer *(USHORT)*
Input device flag:

TRUE Message resulted from pointer event
FALSE Message resulted from keyboard event

param2

ptspointerpos *(POINTS)*
Pointer position

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed.
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_CONTEXTMENU, or a keyboard event, specified by the system value SV_CONTEXTMENUB.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_CONTROL

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

Parameters

param1

idid *(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

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

This message is sent to the owner of the control, thereby offering it the opportunity to perform some activity before returning to the control.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 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

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

Returns

reply

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

Remarks

The recommended approach for an application, that does not have specific reasons for controlling the pointer appearance, is to pass the message to the default window procedure.

Default Processing

The default window procedure returns *hptrhptrNew*.

WM_CREATE

This message occurs when an application requests the creation of a window.

Parameters

param1

ctldata *(PVOID)*
Control data.

This points to a PVOID data structure initialized with the data provided in the *pCtlData* parameter of the WinCreateWindow function.

This pointer is also contained in the *pCREATE* parameter.

param2

pCREATE *(PCREATESTRUCT)*
Create structure.

This points to a CREATESTRUCT data structure.

Returns

reply

fresult *(BOOL)*
Error indicator:

TRUE Discontinue window creation
FALSE Continue window creation.

Remarks

This message is sent to the window procedure of the window being created, thus offering it an opportunity to initialize that window.

The window procedure receives this after the window is created but before the window becomes visible.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE, which is equivalent to continuing the creation of the window.

WM_DESTROY

This message occurs when an application requests the destruction of a window.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

This message is sent to the window procedure of the window being destroyed after it has been hidden on the device, thereby offering it an opportunity to perform some termination action for that window.

Default Processing

The default window procedure takes no action on this message, other than to set *fireply* to 0.

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

reply

fDrawn *(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.

Remarks

A control can only display some types of information, and emphasize items in a control-specific manner. Therefore, if special items are to be displayed or emphasized in a special manner, this must be done by the owner window of the control.

The control window procedure generates this message and sends it to the owner of the control, informing the owner that an item is to be drawn, offering the owner the opportunity to draw that item and to indicate that either the item has been drawn or that the control is to draw it.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fDrawn* to the default value of FALSE.

WM_ENABLE

This message sets the enable state of a window.

Parameters

param1

usNewEnabledState *(USHORT)*
New enabled state indicator:

TRUE	Set the window to enabled state
FALSE	Set the window to disabled state.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

This message is sent to the window procedure of the window whose enable state is changing, thereby offering it an opportunity to perform some action appropriate to new state of the window.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_ENDDRAG

This message occurs when the operator completes a drag operation.

Parameters

param1

usPointer *(USHORT)*
Input device flag:

TRUE Message resulted from pointer event
FALSE Message resulted from keyboard event

param2

ptspointerpos *(POINTS)*
Pointer position

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed.
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_ENDDRAG.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_ENDSELECT

This message occurs when the operator either makes a selection or completes a swipe selection.

Parameters

param1

usPointer *(USHORT)*
Input device flag:

TRUE Message resulted from pointer event
FALSE Message resulted from keyboard event

param2

ptspointerpos *(POINTS)*
Pointer position

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed.
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_ENDSELECT.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_ERASEWINDOW

This message is sent to a window when it is invalidated.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

fresult *(BOOL)*
Erased indicator:

TRUE Window erased.
FALSE Message not processed.

Remarks

If the application processes the message, it can erase the invalid portion of the window.

If the application does not process the message, it is resent at WinBeginPaint time.

Children of asynchronous paint non clip children windows are not erased synchronously, regardless of the WS_SYNCPAINT style. This is because the painting order must be enforced: the parent window must redraw before the child, or else the redraw latency on the part of the parent will draw over any previously-painted children.

Note: The WM_ERASEWINDOW message is sent across processes.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_ERROR

This message occurs when an error is detected in a WinGetMsg or a WinPeekMsg function.

Parameters

param1

userrorcode *(USHORT)*
Error code.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

The application can detect the error situation after the WinGetMsg or the WinPeekMsg function and before the WinDispatchMsg function.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_FOCUSCHANGE

This message occurs when the window possessing the focus is changed.

Parameters

param1

hwndFocus *(HWND)*
Focus window handle.

param2

usSetFocus *(USHORT)*
Focus flag:

TRUE The window is receiving the focus and *hwndFocus* identifies the window losing the focus.

FALSE The window is losing the focus and *hwndFocus* identifies the window receiving the focus.

fsFocusChange *(USHORT)*
Focus changing indicators.

The indicators are passed from the WinFocusChange function with the exception of the FC_SETACTIVEFOCUS value, which is removed before this message is sent.

Returns

flReply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

This message is sent to both the windows gaining and losing the focus.

Default Processing

The default window procedure sends this message to the owner or parent, if it exists and is not the desktop. Otherwise, it sets *flReply* to 0.

WM_FORMATFRAME

This message is sent to a frame window to calculate the sizes and positions of all of the frame controls and the client window.

Parameters

param1

pswp *(PSWP)*
Structure array.

This points to an array that is to hold the SWP structures.

param2

pprectl *(PRECTL)*
Pointer to client window rectangle.

This is typically the window rectangle of *pswp*, but where the window has a wide border, as specified by FCF_DLGBORDER for example, the rectangle is inset by the size of the border.

Returns

reply

ccount *(USHORT)*
Count of the number of *SWP* arrays returned.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *ccount* to the default value of 0.

WM_HELP

This message occurs when a control has a significant event to notify to its owner or when a key stroke has been translated by an accelerator table into a WM_HELP.

Parameters

param1

uscmd *(USHORT)*
Command value.

It is the responsibility of the application to be able to relate *uscmd* to an application function.

param2

ussource *(USHORT)*
Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control. *uscmd* is the window identity of the pushbutton.
CMDSRC_MENU	Posted by a menu control. *uscmd* is the identity of the menu item.
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. *uscmd* is the accelerator command value.
CMDSRC_OTHER	Other source. *uscmd* gives further control-specific information defined for each control type.

uspointer *(USHORT)*
Pointer-device indicator:

TRUE	If the message is posted as a result of a pointer-device operation
FALSE	If the message is posted as a result of a keyboard operation.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message is identical to a WM_COMMAND message, but implies that the application should respond to this message by displaying help information.

This message is posted to the queue of the owner of the control.

Default Processing

The default window procedure sends this message to the parent window, if it exists and is not the desktop. Otherwise, it sets *flreply* to 0.

WM_HITTEST

This message is sent to determine which window is associated with an input from 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 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

ulresult *(ULONG)*
Hit-test indicator.

The application may return one of these values:

HT_NORMAL	The message should be processed as normal. A WM_MOUSEMOVE, WM_BUTTON2DOWN, or WM_BUTTON1DOWN message is posted to the window.
HT_TRANSPARENT	The part of the window underneath the pointer is transparent; hit-testing should continue on windows underneath this window, as if the window did not exist.
HT_DISCARD	The message should be discarded; no message is posted to the application.
HT_ERROR	As HT_DISCARD, except that if the message is a button-down message, an alarm sounds and the window concerned is brought to the foreground.

Remarks

This message occurs when an application requests a message by issuing a WinPeekMsg or a WinGetMsg function.

If the message that is to be retrieved represents a pointer related event, this message is sent to a window to determine whether the message is in fact destined for that window.

This message is only sent if the window class has the CS_HITTEST style set.

Note: The handling of this message determines whether a disabled window can process pointing device events.

Default Processing

The default window procedure takes no action on this message, other than to set *ulresult* to HT_ERROR if the window is disabled, or to HT_NORMAL otherwise.

WM_HSCROLL

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

Parameters

param1

usIdentifier *(USHORT)*
Scroll bar control window identifier.

param2

sSlider *(SHORT)*
Slider position:

0 Either the operator is not moving the slider with the pointer device, or for the instance where *uscmd* is SB_SLIDERPOSITION the pointer is outside the tracking rectangle when the button is released.

Other Slider position.

uscmd *(USHORT)*
Command:

SB_LINELEFT	Sent if the operator clicks on the left arrow of the scroll bar, or depresses the VK_LEFT key.
SB_LINERIGHT	Sent if the operator clicks on the right arrow of the scroll bar, or depresses the VK_RIGHT key.
SB_PAGELEFT	Sent if the operator clicks on the area to the left of the slider, or depresses the VK_PAGELEFT key.
SB_PAGERIGHT	Sent if the operator clicks on the area to the right of the slider, or depresses the VK_PAGERIGHT key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider.
SB_SLIDERTRACK	If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

Returns

fIreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Default Processing

The default window procedure takes no action on this message, other than to set *fIreply* to 0.

WM_INITDLG

This message occurs when a dialog box is being created.

Parameters

param1

hwndhwnd *(HWND)*
Focus window handle.

The handle of the control window that is to receive the input focus.

param2

pcreate *(PCREATEPARAMS)*
Application-defined data area.

This points to the data area and is passed by the WinLoadDlg, WinCreateDlg, and WinDlgBox functions in their *pCreateParams* parameter.

Returns

reply

fresult *(BOOL)*
Focus set indicator:

TRUE	Focus window is changed. The dialog procedure can change the window to receive the focus, by issuing a WinSetFocus whose *hwndNewFocus* specifies the handle of another control within the dialog box.
FALSE	Focus window is not changed.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_INITMENU

This message occurs when a menu control is about to become active.

Parameters

param1

smenuid *(SHORT)*
Menu-control identifier.

param2

hwndhwnd *(HWND)*
Menu-window handle.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_JOURNALNOTIFY

This message is used to maintain correct operation during journal playback.

Parameters

param1

ulCommand *(ULONG)*
Command to journal.

JRN_QUEUESTATUS	The WinQueryQueueStatus command must be journaled.
JRN_PHYSKEYSTATE	The WinGetPhysKeyState command must be journaled.

param2
Data.

Data values depend on which command is to be journaled.

If *ulCommand* is set to JRN_QUEUESTATUS:

fsQueueStatus *(USHORT)*
Queue status.

See the *Summary* parameter of the WinQueryQueueStatus function.

If *ulCommand* has the value JRN_PHYSKEYSTATE:

usScanCode *(USHORT)*
Scan code.

See the *lScancode* parameter of the WinGetPhysKeyState function.

usKeyState *(USHORT)*
Key State.

See the *lKeyState* parameter of the WinGetPhysKeyState function.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

If the WinQueryQueueStatus or the WinGetPhysKeyState functions have new information since the last time they were called and there is a journal record hook installed, the journal record hook is called with this message to record this new information.

During playback, this message is interpreted by the system and the appropriate state restored.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *flreply* to 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 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

fresult *(BOOL)*
Match indicator:

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

Default Processing

The default dialog procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_MEASUREITEM

This notification is sent to the owner of a specific control to establish the height and width for an item in that control.

Parameters

param1

sIdentity *(SHORT)*
Control identifier.

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 control section.

Returns

reply

sHeight *(SHORT)*
Height of item.

sWidth *(SHORT)*
Width of item.

Remarks

When the owner receives this message, it must calculate and return the height and width (for a horizontally-scrollable list box control) of an item to the control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *reply* to the default value of 0.

WM_MENUEND

This message occurs when a menu control is about to terminate.

Parameters

param1

usmenuid *(USHORT)*
Menu-control identifier.

param2

hwndhwnd *(HWND)*
Menu-control window handle.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_MENUSELECT

This message occurs when a menu item has been selected.

Parameters

param1

usItem *(USHORT)*
Identifier of selected item.

usPostCommand *(USHORT)*
Post-command flag:

TRUE Indicates that either a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message is being posted by the menu control on return from the owner, subject to *fresult*.

FALSE Indicates that no message is being posted by the menu control on return from the owner, subject to *fresult*.

param2

hwndhwnd *(HWND)*
Menu-control window handle.

Returns

reply

fresult *(BOOL)*
Post indicator:

TRUE Indicates that either a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message is to be posted by the menu control window procedure. The menu is dismissed if the selected item does not have a style of MIA_NODISMISS.

FALSE Indicates that no message is to be posted by the menu control window procedure and that the menu is not dismissed.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to TRUE.

WM_MINMAXFRAME

This message is sent to a frame window that is being minimized, maximized, or restored.

Parameters

param1

pswp *(PSWP)*
Set window position structure.

This points to a SWP structure. The structure has the appropriate SWP_* indicators set to describe the operation that is occurring to the window.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

fOverrideDefault *(BOOL)*
Processed indicator:

TRUE The message has been processed; the default system actions for the operation specified by the *pswp* parameter to the window are not to be performed.

FALSE The message has been ignored; the default system actions for the operation specified by the *pswp* parameter to the window are to be performed.

Default Processing

The default window procedure takes no action on this message, other than to set *fOverrideDefault* to FALSE.

WM_MOUSEMOVE

This message occurs when the pointing device pointer moves.

Parameters

param1

This parameter contains the position of the pointing device in window coordinates relative to the bottom-left corner of the window.

sxMouse *(SHORT)*
Pointing device x-coordinate.

syMouse *(SHORT)*
Pointing device 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

KC_IGNOREKEY Indicates the keyboard state is to be ignored.

Returns

reply

fProcessed *(BOOL)*
Processed indicator:

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

Remarks

The keyboard control codes specified by 'flags' reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The KC_IGNOREKEY is used for mouse messages where the keyboard state is to be ignored. For example, WM_BEGINDRAG might result from a WM_BUTTON2MOTIONSTART start message with the KC_IGNOREKEY flag. This means that if a key state, such as KC_SHIFT, was active that it wouldn't be a factor in deciding whether the WM_BEGINDRAG message should be sent.

Default Processing

The default window procedure sets the pointer shape using the WinSetPointer function and sets *fProcessed* to FALSE.

WM_MOVE

This message occurs when a window with style CS_MOVENOTIFY changes its absolute position.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

The message is sent from WinSetWindowPos, WinSetMultWindowPos, and WinScrollWindow.

The message is sent to any window when it is moved relative to its parent window. In addition, a WM_MOVE message is also sent to any children of that window that have style CS_MOVENOTIFY.

The new position of the window is obtained by calling WinQueryWindowRect, and can make those rectangle coordinates relative to any window by calling WinMapWindowPoints.

Note: There are several instances where windows have cause to know if they have been moved, and these include the occasions when the window does not change position relative to its parent, but does change position relative to the screen (its absolute position).

An example is menus. When a top-level menu control (child of the frame window) moves its absolute position as a result of the frame window being moved, the top-level menu control causes the movement of any pull-down menus along with its movement. The same applies to application/dialog box positional grouping. In some instances, a dialog box might cause to be moved as the main window is moved, to make room for other applications.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_NEXTMENU

This message occurs when either the beginning or the end of the menu is reached by use of the cursor control keys.

Parameters

param1

hwndMenu *(HWND)*
Menu-control window handle.

param2

usPrev *(USHORT)*
Previous-menu indicator:

TRUE Beginning of the menu has been reached
FALSE End of the menu has been reached.

Returns

reply

hwndNewMenu *(HWND)*
New menu window handle:

NULLHANDLE	No new menu
Other	New menu window handle.

Default Processing

The default window procedure takes no action on this message, other than to set *hwndNewMenu* to NULLHANDLE.

WM_NULL

This message is posted to activate message queues or modal loops.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

On receiving this message, the application should simply let the default processing take place.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_OPEN

This message occurs when the operator makes an *OPEN* request.

Parameters

param1

usPointer *(USHORT)*
Input device flag:

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event

param2

ptspointerpos *(POINTS)*
Pointer position

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed.
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_OPEN.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_PACTIVATE

This message is posted when the Language Support Window or Dialog Procedure processes a WM_ACTIVATE message.

Parameters

param1

usactive *(USHORT)*
Active indicator:

TRUE The window was activated
FALSE The window was deactivated.

param2

hwndhwnd *(HWND)*
Window handle.

In the case of activation, *hwndhwnd* identifies the window which was activated. In the case of deactivation, *hwndhwnd* identifies the window which was deactivated.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

The activation change has already occurred when the application receives this message.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_PAINT

This message occurs when a window needs repainting.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Default Processing

The default window procedure issues the WinBeginPaint and WinEndPaint functions, and then sets *flreply* to 0.

WM_PCONTROL

This message is posted when the Language Support Window or Dialog Procedure processes a WM_CONTROL message.

Parameters

param1

Idld *(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 *(ULONG)*
Zero.

0 The control-specific information in *ulcontrolspec* of the WM_CONTROL message is not available because the information might not be valid when the application receives this message.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

The notification from the control has already been processed when the application receives this message.

Default Processing
The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_PPAINT
This message is posted when the Language Support Window or Dialog Procedure processes a WM_PAINT message.

Parameters
param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns
flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Default Processing
The default window procedure issues the WinBeginPaint and WinEndPaint functions, and then sets *flreply* to 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns
flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks
This message notifies a control when an inherited presentation parameter changes.

Default Processing
The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_PSETFOCUS

This message is posted when the Language Support Window or Dialog Procedure processes a WM_SETFOCUS message.

Parameters

param1

hwndhwnd *(HWND)*
Focus-window handle:

NULLHANDLE	No window lost or received the focus.
Other	Window handle.

param2

usfocus *(USHORT)*
Focus flag:

TRUE	The window received the focus. *hwndhwnd* is the window handle of the window which lost the focus, or NULLHANDLE if no window previously had the focus.
FALSE	The window lost the focus. *hwndhwnd* is the window handle of the window which received the focus, or NULLHANDLE if no window received the focus.

Returns

flreply *(ULONG)*
Reserved.

0	Reserved value, 0.

Remarks

The focus change has already occurred when the application receives this message.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_PSIZE

This message is posted when the Language Support Window or Dialog Procedure processes a WM_SIZE message.

Parameters

param1

scxold *(SHORT)*
Old horizontal size.

scyold *(SHORT)*
Old vertical size.

param2

scxnew *(SHORT)*
New horizontal size.

scynew *(SHORT)*
New vertical size.

Returns

flreply *(ULONG)*
Reserved.

0	Reserved value, 0.

Remarks
The size change has already occurred when the application receives this message.

Default Processing
The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_PSYSCOLORCHANGE
This message is posted when the Language Support Window or Dialog Procedure processes a WM_SYSCOLORCHANGE message.

Parameters
param1

flOptions *(ULONG)*
Options.

Copied from the *flOptions* parameter of the WinSetSysColors function.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns
flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks
All windows in the system are invalidated so that they will be redrawn with the new system color.

Default Processing
The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_QUERYACCELTABLE
This message returns the handle to the accelerator table of a window.

Parameters
param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns
reply

haccelhaccel *(HACCEL)*
Accelerator table handle:

NULLHANDLE	No accelerator table is associated with the window.
Other	The handle of the accelerator table associated with the window.

Default Processing

The default window procedure takes no action on this message, other than to set *haccelhaccel* to NULLHANDLE.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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.

Remarks

This message enables a DBCS application to determine whether the window with the input focus can handle DBCS characters. The *pCursorPos* parameter can be used as a guide for positioning any conversion window that the application requires.

Default Processing

The default window procedure returns QCP_CONVERT, and updates *pCursorPos* to the following values:

- xleft = -1
- ybottom = -1
- xright = 0
- ytop = 0

WM_QUERYHELPINFO

This message returns the help instance associated with a frame window.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

lhelpInfo *(LONG)*
Help information:

0 No help information associated with the window.
Other The help information associated with the window.

Default Processing

The default window procedure takes no action on this message, other than to set *haccelhaccel* to NULLHANDLE.

WM_QUERYTRACKINFO

The frame control generates this message on receiving a WM_TRACKFRAME (in Frame Controls) message.

Parameters

param1

ustflags *(USHORT)*
Tracking flags.

Contains a combination of one or more TF_* flags as defined in the TRACKINFO structure.

param2

ptrackInfo *(PTRACKINFO)*
Track information structure.

This points to a TRACKINFO structure. The receiver of this message must modify this structure.

Returns

reply

fresult *(BOOL)*
Continue indicator:

TRUE Continue sizing or moving
FALSE Terminate sizing or moving.

Remarks

This message is sent to the window procedure of the owner of a frame control or title bar control respectively.

The TRACKINFO data structure specified by the *ptrackinfo* parameter is not initialized before the message is sent. It must be correctly completed before returning.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

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 A-125.

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

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

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fresult *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

If this message is sent to a window of another process, the information in, or identified by, *pwndparams* must be in memory shared by both processes.

Default Processing

The default window procedure sets the *ulText*, *ulPresParams*, and *ulCtlData* parameters of the WNDPARAMS data structure identified by the *pwndparams* to 0, and sets *fresult* to FALSE.

WM_QUIT

This message is posted to terminate the application.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

It causes WinGetMsg to return *fResult* set to FALSE, rather than to TRUE, as for all other messages.

Note: Applications that call WinPeekMsg rather than WinGetMsg should test explicitly for WM_QUIT.

This message should not be dispatched to the default window procedure. The intent of this message is to cause the WinGetMsg loop to terminate.

Typically this message is posted by the application when the application exit command is selected from the action bar.

This message is also sent to all applications when the system is closing down. To reply to this, the application should either cancel the request by issuing an WinCancelShutdown function or close itself down by issuing a WinDestroyMsgQueue function.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_REALIZEPALETTE

This message is sent to an application whenever changes have been made to the display hardware physical color table as a result of another application calling WinRealizePalette.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The application should call WinRealizePalette if it has a palette, or pass it on to the default window procedure if it does not.

If the return value from WinRealizePalette is greater than 0, the application should invalidate its window to cause a repaint using the newly-realized palette.

Default Processing

The default window procedure calls WinRealizePalette with a NULL *hps* parameter. This causes the default palette to be realized. If the return value from WinRealizePalette is greater than 0, the default window procedure invalidates the window, causing it to be repainted with the newly-realized palette.

WM_SAVEAPPLICATION

This message is sent by the system to notify an application to save its current state.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

When an application receives this message, it is expected to save its current state by any convenient method, for example, in a profile or in an auxiliary file.

It is the responsibility of the application to use the saved information, as appropriate, when it is resumed.

Even if the application processes this message, it should also pass it to the default window procedure, by using the WinDefWindowProc call.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SEM1

This message is sent or posted by an application.

Parameters

flAccumBits *(ULONG)*
Semaphore value.

The semaphore values from all the WM_SEM1 messages posted to a queue, are accumulated by a logical-OR operation.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

If the message is posted, it is merged with any existing WM_SEM1 message on the queue by combining the two *flAccumBits* values using a logical-OR operation.

The WM_SEM1 messages are queued higher than any other type of message.

Default Processing
The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SEM2
This message is sent or posted by an application.

Parameters
flAccumBits *(ULONG)*
Semaphore value.

The semaphore values from all the WM_SEM2 messages posted to a queue, are accumulated by a logical-OR operation.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns
flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks
If the message is posted, it is merged with any existing WM_SEM2 message on the queue by combining the two *flAccumBits* values using a logical-OR operation.

The WM_SEM2 messages are queued above WM_SEM3 and WM_SEM4 messages, and above any WM_PAINT or WM_TIMER messages generated by the system, but lower than any other message.

Default Processing
The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SEM3
This message is sent or posted by an application.

Parameters
flAccumBits *(ULONG)*
Semaphore value.

The semaphore values from all the WM_SEM3 messages posted to a queue, are accumulated by a logical-OR operation.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns
flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks
If the message is posted, it is merged with any existing WM_SEM3 message on the queue by combining the two *flAccumBits* values using a logical-OR operation.

The WM_SEM3 messages are queued above WM_SEM4 messages, and any WM_PAINT messages generated by the system, but lower than any other message.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SEM4

This message is sent or posted by an application.

Parameters

flAccumBits *(ULONG)*

Semaphore value.

The semaphore values from all the WM_SEM4 messages posted to a queue, are accumulated by a logical-OR operation.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*

Reserved.

0 Reserved value, 0.

Remarks

If the message is posted, it is merged with any existing WM_SEM4 message on the queue by combining the two *flAccumBits* values using a logical-OR operation.

The WM_SEM4 messages are queued lower than any other type of message.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SETACCELTABLE

This message establishes the window accelerator table to be used for translation, when the window is active.

Parameters

param1

hAccelhAccelNew *(HACCEL)*

New accelerator table.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*

Success indicator:

TRUE Successful completion
FALSE Error occurred.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

WM_SETFOCUS

This message occurs when a window is to receive or lose the input focus.

Parameters

param1

hwndhwnd *(HWND)*
Focus-window handle:

NULLHANDLE	No window is losing or receiving the focus.
Other	Window handle.

param2

usfocus *(USHORT)*
Focus flag:

TRUE	The window is receiving the focus. *hwndhwnd* is the window handle of the window losing the focus, or NULLHANDLE if no window previously had the focus.
FALSE	The window is losing the focus. *hwndhwnd* is the window handle of the window receiving the focus, or NULLHANDLE if no window is receiving the focus.

Returns

flreply *(ULONG)*
Reserved.

0	Reserved value, 0.

Remarks

This message is sent to the window receiving or losing the focus, thereby giving it the opportunity to perform some appropriate processing.

Note: Except in the instance of WM_ACTIVATE, with *usactive* set to TRUE, an application processing WM_SETFOCUS or WM_ACTIVATE messages should not change the focus window or active window. If it does, the focus and active window must be restored before the application returns from processing the message. For this reason, any dialog boxes or windows brought up during the processing of WM_SETFOCUS or WM_ACTIVATE messages should be system modal.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SETHELPINFO

This message sets the help instance associated with this frame window when the window is active.

Parameters

param1

lhelpinfo *(LONG)*
New help information.

param2 *(ULONG)*
Reserved.

0	Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

WM_SETSELECTION

This message occurs when a window is selected or deselected.

Parameters

param1

usselection *(USHORT)*
Selection flag:

TRUE	The window is selected.
FALSE	The window is deselected.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The window procedure is expected to highlight or unhighlight the selected item of the window, as appropriate.

This message is sent to a window when it loses the focus to another window that it does not own. It allows an application to remove the selection when the focus is removed to another application, but to keep it if, for example, the same application displays a dialog box.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

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 A-125.

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

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fresult *(BOOL)*
Success indicator:

TRUE Successful operation
FALSE Error occurred.

Remarks

If this message is sent to a window of another process, the information in, or identified by, *pwndparams* must be in memory shared by both processes.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_SHOW

This message occurs when the WS_VISIBLE state of a window is being changed.

Parameters

param1

usshow *(USHORT)*
Show indicator:

TRUE Show the window
FALSE Hide the window.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The message is sent after the visibility state has changed.

In this context, the terms "shown" or "hidden" refer to the state of the WS_VISIBLE style bit. This message is *not* sent when a window is obscured by other windows above it.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SINGLESELECT

This message occurs when the operator selects a single object.

Parameters

param1

usPointer *(USHORT)*
Input device flag:

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event

param2

ptspointerpos *(POINTS)*
Pointer position

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *usPointer* is not set to TRUE.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE	Message processed.
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_SINGLESELECT.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *fresult* to FALSE.

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

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message is not sent by WinCreateWindow when a window is created, and so any size-related processing must be done during the WM_CREATE message processing in this instance.

This message is sent after the window has been actually sized, but before any repainting has been done. Any resizing or repositioning of child windows that might be necessary a a result of the size change is usually done during the processing of this message.

Note: It is generally unwise to output to the window during the processing of this message, because the area drawn might be redrawn, after the WM_SIZE processing is complete, by the WinSetWindowPos function.

The processing of this message for a window which is displaying an advanced VIO presentation space must be carried out by the default advanced VIO window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SUBSTITUTESTRING

This message is sent from the WinSubstituteStrings call.

Parameters

param1

lindex *(USHORT)*
Substitution index.

A value corresponding to the decimal character in the substitution phrase.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

pstring *(PSTRL)*
String to be substituted:

This points to a PSZ.

0 No substitution string
Other Substitution string.

Remarks

The WinSubstituteStrings call has encountered a substitution phrase in a string. The substitution phrase takes the form '% <digit>', where <digit> is a single decimal character; that is, 0 through 9.

Default Processing

The default window procedure takes no action on this message, other than to set *reply* to 0.

WM_SYSCOLORCHANGE

This message is sent to all main windows when a change is made to the system colors by the WinSetSysColors function.

Parameters

param1

flOptions *(ULONG)*
Options.

Copied from the *flOptions* parameter of the WinSetSysColors function and therefore specifies which palette has been changed.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

All windows are invalidated, so that they are redrawn with the new colors. When this message is received, applications that depend on the system colors can query the new color values with the WinQuerySysColor call.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SYSCOMMAND

This message occurs when a control has a significant event to notify to its owner or when a key stroke has been translated by an accelerator table into a WM_SYSCOMMAND message.

Parameters

param1

uscmd *(USHORT)*
Command value.

The command value can be one of the SC_* values. It is the responsibility of the application to be able to relate *uscmd* to an application function.

param2

ussource *(USHORT)*
Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control. *uscmd* is the window identifier of the pushbutton.
CMDSRC_MENU	Posted by a menu control. *uscmd* is the identifier of the menu item.
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. *uscmd* is the accelerator command value.
CMDSRC_OTHER	Other source. *uscmd* gives further control-specific information defined for each control type.

uspointer *(USHORT)*
Pointing-device indicator:

TRUE	The message is posted as a result of a pointing-device operation.
FALSE	The message is posted as a result of a keyboard operation.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message is posted to the queue of the owner of the control, thereby offering it the opportunity to perform some activity as a result.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SYSVALUECHANGED

This message is posted to all main windows when one of the settable system values is changed.

Parameters

param1

usChangedFirst *(USHORT)*
First system value.

The first of a contiguous set of system values that has been changed.

param2

usChangedLast *(USHORT)*
Last system value.

The last of a contiguous set of system values that has been changed.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

If *usChangedFirst* equals *usChangedLast*, only one system value has changed.

If an application changes the settable system values, it is the responsibility of the application to post this message to all main windows.

This message is processed by WC_FRAME windows by doing any frame-specific processing (such as sending WM_SETBORDERSIZE messages to the size border if SV_CX/CYSIZEBORDER system values have changed) and then sending the message to the client window if one exists.

This message is only posted when settable system values change.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_TEXTEDIT

This message occurs when the operator requests a direct name edit operation.

Parameters

param1

usPointer *(USHORT)*
Input device flag:

TRUE Message resulted from pointer event
FALSE Message resulted from keyboard event

param2

ptspointerpos *(POINTS)*
Pointer position

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed.
FALSE Message ignored.

Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from either a mouse event, specified by the system value SV_TEXTEDIT, or a keyboard event, specified by the system value SV_TEXTEDITKB

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_TIMER

This message is posted when a timer times out.

Parameters

param1

idTimer *(USHORT)*
Timer identity.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message is always queued and is processed specially by the WinGetMsg and WinPeekMsg calls, as follows:

1. Timers are processed only by the WinGetMsg and WinPeekMsg calls.
2. A timer posts only one WM_TIMER message at a time.
3. WM_TIMER messages are queued lower than all other messages except WM_SEM3, WM_PAINT, and WM_SEM4 messages.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_TRACKFRAME

This message is sent to a window whenever it is to be moved or sized.

Parameters

param1

fsTrackFlags *(USHORT)*
Tracking flags.

Contains a combination of one or more TF_* flags; for details, see the TRACKINFO data structure description.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fresult *(BOOL)*
Success indicator:

TRUE The operation is successful.
FALSE The operation is unsuccessful, or the operation is terminated.

Remarks

Respond to this message by causing a tracking rectangle to be drawn to move or size the window. For information, see WinTrackRect..

Default Processing

None.

WM_TRANSLATEACCEL

This message is sent to the focus window whenever a WM_CHAR message occurs.

Parameters

param1

pqmsg *(PQMSG)*
QMSG structure.

This points to a QMSG structure.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fTranslated *(BOOL)*
Translated indicator:

TRUE The character exists in the accelerator table and has been translated in the QMSG structure.
FALSE The character does not exist in the accelerator table or the window does not have an accelerator table.

Remarks

Normally, this message is not processed by the focus window, but is d passed to its parent, which passes it to its parent, until a frame window is reached.

Default Processing

The default window procedure takes no action on this message, other than to set *fTranslated* to FALSE.

WM_TRANSLATEMNEMONIC

This message occurs during frame control processing of a WM_TRANSLATEACCEL message.

Parameters

param1

pqmsg *(PQMSG)*
QMSG structure.

This points to a QMSG structure.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE The character has been translated into an accelerator.
FALSE The character has not been translated into an accelerator.

Remarks

This message is sent by the frame control to itself during the processing of a WM_TRANSLATEACCEL message, if the frame control does not translate a character into an accelerator by use of the frame window or queue accelerator tables.

When the frame control receives this message, it sends it to the application menu window, that is the window with identity FID_MENU.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

WM_UPDATEFRAME

This message is sent by an application after frame controls have been added or removed from the window frame.

Parameters

param1

flCreateFlags *(ULONG)*
Frame-creation flags.

Contains the FCF_* flags that indicate which frame controls have been added or removed.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_VSCROLL

This message occurs when a vertical scroll-bar control has a significant event to notify to its owner.

Parameters

param1

usIdentifier *(USHORT)*
Scroll bar-control window identifier.

param2

sSlider *(SHORT)*
Slider position:

0 Either the operator is not moving the slider with the pointer device, or for the instance when *uscmd* is SB_SLIDERPOSITION the pointer is outside the tracking rectangle when the button is released.
Other Slider position.

uscmd *(USHORT)*
Command:

SB_LINEUP Sent if the operator clicks on the up arrow of the scroll bar, or presses the VK_UP key.

SB_LINEDOWN	Sent if the operator clicks on the down arrow of the scroll bar, or presses the VK_DOWN key.
SB_PAGEUP	Sent if the operator clicks on the area above the slider, or presses the VK_PAGEUP key.
SB_PAGEDOWN	Sent if the operator clicks on the area below the slider, or presses the VK_PAGEDOWN key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider.
SB_SLIDERTRACK	If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_WINDOWPOSCHANGED

This message is sent to the window procedure of the window whose position is changed, that is has any of the values of the *fl* parameter of the SWP structure set, with the exception of the SWP_NOADJUST and SWP_NOREDRAW values.

This message is also sent if the return value from the WM_ADJUSTWINDOWPOS is not 0.

Parameters

param1

pswp *(PSWP)*
SWP structures.

This points to two SWP structures. The first SWP structure describes the entire new window state, whereas the second structure describes the entire old window state. The *fl* parameter of the first structure contains only those indicators corresponding to the state changes that occurred.

param2

flAwp *(ULONG)*
Adjust window position status indicators.

The return value from the WM_ADJUSTWINDOWPOS message:

0	The SWP_NOADJUST option has been specified.
Other	Adjust window position status indicators.

The AWF_* flags specify the state change of the frame window.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure sets *flreply* to 0 and sends the following messages, based on the values of the *fl* parameter of the first SWP data structure:

SWP_SIZE	A WM_SIZE with the new window size from the first SWP structure
SWP_HIDE	A WM_SHOW to hide the new window
SWP_SHOW	A WM_SHOW to show the new window.

Default Dialog Processing

This section describes how messages are processed by the default dialog procedure. The default dialog procedure can be called using WinDefDlgProc. A user dialog procedure should make this call for all messages that it does not want to process.

For WM_* messages other than those specified in this section the Default Dialog Procedure takes the same action and sets **result** to the same value as in Chapter 15, "Frame Control Window Processing." In the instance of messages that would be sent to FID_CLIENT, they are passed to the default window procedure.

For any other messages the default window procedure takes no action, other than to set **reply** to NULL.

WM_CHAR (Default Dialogs)

For the cause of this message, see "WM_CHAR" on page 12-24.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-24.

Default Processing

If KC_CHAR is the mnemonic for a button that already has the focus, a BM_CLICK is sent to that button and *fresult* is set to TRUE. If the button does not have the focus, it receives the focus and *fresult* is set to TRUE.

If *usvk* contains the value VK_TAB, the focus is set to the next tab item in the dialog. *fresult* is set to TRUE.

If *usvk* contains the value VK_BACKTAB, the focus is set to the previous tab item in the dialog. *fresult* is set to TRUE.

If *usvk* contains the value VK_LEFT or VK_UP, the focus is set to the previous item in the group. *fresult* is set to TRUE.

If *usvk* contains the value VK_RIGHT or VK_BOTTOM, the focus is set to the next item in the group. *fresult* is set to TRUE.

If *usvk* contains the value VK_ENTER or VK_NEWLINE, if a pushbutton has the focus a BM_CLICK is sent to that button. *fresult* is set to TRUE. If another control in the dialog has the focus the dialog is searched for a pushbutton with style BS_DEFAULT. If a pushbutton of this style is found, a BM_CLICK is sent to that button and *fresult* is set to TRUE.

If *usvk* contains the value VK_ESC, WM_COMMAND is posted, with *ussource* is set to CMDSRC_PUSHBUTTON and *uscmd* is set to DID_CANCEL. *fresult* is set to TRUE.

In other instances, if an owner exists the message is sent to the owner, otherwise *fresult* is set to FALSE.

WM_CLOSE (Default Dialogs)

For the cause of this message, see "WM_CLOSE" on page 12-26.

Parameters

For a description of the parameters, see "WM_CLOSE" on page 12-26.

Default Processing

The default dialog procedure responds to this message by dismissing the dialog by issuing the WinDismissDlg function with its *ulResult* parameter set to DID_CANCEL.

WM_COMMAND (Default Dialogs)

For the cause of this message, see "WM_COMMAND" on page 12-27.

Parameters

For a description of the parameters, see "WM_COMMAND" on page 12-27.

Default Processing

The default dialog procedure responds to this message by dismissing the dialog and passing *uscmd* (the control item identifier) as *ulResult* of the WinProcessDlg or the WinDlgBox function that initiated the dialog. It sets *flreply* to 0.

WM_INITDLG (Default Dialogs)

For the cause of this message, see "WM_INITDLG" on page 12-38.

Parameters

For a description of the parameters, see "WM_INITDLG" on page 12-38.

Remarks

This message is sent to the dialog procedure, before the dialog box is shown, thereby offering the dialog procedure the opportunity to perform the initialization of the dialog box.

If any string substitutions are made by the WinSubstituteStrings call when the dialog is created, the WM_SUBSTITUTESTRING message may have been sent before the WM_INITDLG message is sent.

Default Processing

The default dialog procedure passes this message to the default window procedure, which sets *fresult* to FALSE.

WM_MATCHMNEMONIC (Default Dialogs)

For the cause of this message, see "WM_MATCHMNEMONIC" on page 12-40.

Parameters

For a description of the parameters, see "WM_MATCHMNEMONIC" on page 12-40.

Remarks

This message is only processed by Button and Static Controls; all other controls return FALSE.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_QUERYDLGCODE

This message is sent by the dialog manager to identify the type of control, to determine what kinds of messages the control understands, and also to determine whether an input message may be processed by the dialog manager or passed down to the control.

Parameters

param1

ppQmsg *(PQMSG)*
Message queue structure.

This points to a QMSG structure.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

ulDialogCode *(ULONG)*
Dialog code information flags.

These identify the type of control:

DLGC_ENTRYFIELD	Identifies an entry field control. Assumed to understand the EM_SETSEL message.
DLGC_BUTTON	Identifies a button item. Assumed to understand the BM_CLICK message.
DLGC_CHECKBOX	Identifies a check-box item. Used with the DLGC_BUTTON code.
DLGC_RADIOBUTTON	Identifies a radio button control. Used with the DLGC_BUTTON code.
DLGC_STATIC	Identifies a static control. Static controls are not included in arrow key enumeration.
DLGC_DEFAULT	Identifies a default pushbutton control.
DLGC_PUSHBUTTON	Identifies a nondefault pushbutton.
DLGC_SCROLLBAR	Identifies a scroll bar control.
DLGC_MENU	Identifies a menu control.
DLGC_MLE	Identifies a multiline entry field control.

Remarks

When processing user input, the dialog manager makes some assumptions about the operation of specific controls. The dialog manager sends the WM_QUERYDLGCODE message to obtain a code that governs what assumptions can be made.

If the window receiving this message is *not* a control as defined above, this message returns 0.

Default Processing

The default dialog procedure takes no action on this message, other than to set *ulDialogCode* to NULL.

Default File Dialog Processing

This section describes how messages are processed by the default dialog procedure of the file dialog. This standard dialog can be used to provide a common, consistent file selection function.

The file dialog's default procedure can be called using the WinDefFileDlgProc function. A user-provided subclassing dialog procedure should make this call for all messages that it does not process when using the file dialog.

The default dialog procedure of the file dialog sends the messages listed in this section to itself to perform the requested action. This design allows a user-provided dialog procedure to customize the file dialog to its own needs.

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

This is the ID of the message that is displayed by the file dialog if the default file dialog procedure processes the message.

usErrorId *(USHORT)*
Error message ID.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

usUserReply *(USHORT)*
User's reply.

Specifies the user's reply to the error message presented. Return values are as follows:

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.

Remarks

The application uses this message to provide application-specific error messages in response to file dialog errors that are detected during file dialog processing. The application can choose whether to allow the dialog to present its message or whether to provide its own message and return the response from that message window to the dialog for processing.

Default Processing

The WinDefDlgProc function does not expect to receive this message and takes no action on it other than to return NULL.

FDM_FILTER

This message is sent before a file that meets the current filter criteria is added to the File list box.

Parameters

param1

pszFilename *(PSZ)*
Pointer.

Pointer to the file name.

param2

pszEAType *(PSZ)*
Pointer.

Pointer to the .TYPE EA extended attribute.

Returns

reply

bFilterAction *(BOOL)*
Success indicator.

TRUE Add the file.
FALSE Do not add the file.

Remarks

The application checks this message to obtain the name and the .TYPE EA extended attribute of the file to be added. The application then determines whether or not the file will be added.

When FALSE is returned, the file is not added to the dialog's list box.

Default Processing

The WinDefDlgProc function does not expect to receive this message and takes no action on it other than to return FALSE.

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

pszFileName *(PSZ)*
Pointer.

Pointer to the fully-qualified file name.

param2

usSeltype *(USHORT)*
Selection type.

Returns

reply

bValidity *(BOOL)*
Validity indicator.

TRUE File name is valid.
FALSE File name is not valid.

Remarks

This message is only sent just before the dialog returns to the caller with the user-selected file name. Before this message is sent, *pszFileName* is updated with the user-selected file name. The application can determine if this file name is acceptable. For instance, if the file dialog is being used to pick a "SaveAs" file name, the application can check to see if the file is read-only. If it is, a warning dialog should be brought up to notify the user.

When FALSE is returned from a FDM_VALIDATE message, the dialog will not be dismissed and the user can continue to use the File Dialog to select an alternate file.

In multiple file selection dialogs this message is sent for each selected entry within the file list box. When the name of the file being validated comes from a selected entry in the list box, *param2* will contain FDS_LBSELECTION. When the name of the file comes from the file name entry field, *param2* will contain FDS_EFSELECTION. Single file selection dialogs will always return FDS_EFSELECTION in *param2* since the returned file name always comes from the single line entry field.

Default Processing

The WinDefDlgProc function does not expect to receive this message and takes no action on it other than to return FALSE.

Default Font Dialog Processing

This section describes how messages are processed by the default dialog procedure of the font dialog. This standard dialog can be used to provide a common, consistent font selection function.

The font dialog's default procedure can be called using the WinDefFontDlgProc function. A user-provided subclassing dialog procedure should make this call for all messages that it does not process when using the font dialog.

The default dialog procedure of the font dialog sends the messages listed in this section to itself to perform the requested action. This design allows a user-provided dialog procedure to customize the font dialog to its own needs.

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.

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 A-76 for the default field values.

hwnd *(HWND)*
Window handle of the sample area.

hps *(HPS)*
Presentation-space handle.

fsState *(USHORT)*
Reserved.

fsAttribute *(USHORT)*
Reserved.

fsStateOld *(USHORT)*
Reserved.

fsAttributeOld *(USHORT)*
Reserved.

rclItem *(RECTL)*
Item rectangle to be drawn in window coordinates.

idItem *(SHORT)*
Reserved.

hItem *(PCNRDRAWITEMINFO)*
Reserved.

Returns

reply

drawn *(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.

Remarks

The font dialog provides this message to give the application the opportunity to provide a custom drawn preview area.

The font dialog default dialog procedure generates this message and sends it to its owner, informing the owner that the preview area is to be drawn. The owner is then given the opportunity to draw that area and to indicate that the area has been drawn or that the font dialog is to draw it.

Default Processing

For a description of the default processing, see "WM_DRAWITEM" on page 12-31.

FNTM_FACENAMECHANGED

This message notifies the subclassing application whenever the font family name is changed by the user.

Parameters

param1

pszFamilyname *(PSZ)*
Pointer.

Pointer to the currently-selected face name.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

pszFamilyname is the currently selected family name. The application can modify this string if it desires. The buffer set aside is the maximum size a face name string can be (FACESIZE).

Default Processing

The WinDefDlgProc function does not expect to receive this message and takes no action on it other than to return 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

pszFontname *(PSZ)*
Pointer.

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

reply

fFilterAction *(BOOL)*
Filter indicator.

TRUE	Add the text string to the combination box.
FALSE	Do not add the text string to the combination box.

Remarks

The application checks this message to obtain the name and the .TYPE EA extended attribute of the file being added. The application then determines whether or not the file will be added.

When FALSE is returned, the file is not added to the dialog's list box.

Default Processing

The WinDefDlgProc function does not expect to receive this message and takes no action on it other than to return FALSE.

FNTM_POINTSIZECHANGED

This message notifies subclassing applications when the point size of the font is changed by the user.

Parameters

param1

pszPointSize *(PSZ)*
Pointer.

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

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

When the application wants to limit the point sizes the user can select, it should process this message by changing the *pszPointSize* value and putting up a message box explaining the limitation to the user.

Default Processing

The WinDefDlgProc function does not expect to receive this message and takes no action on it other than to return 0.

FNTM_STYLECHANGED

This message notifies subclassing applications when the user changes any of the attributes in the STYLECHANGE structure.

Parameters

param1

stycstyc *(STYLECHANGE)*
Style changes.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The "Old" fields show the style attributes before the user made the change. The other parameters show what the state will be after the application passes this message to WinDefFontDlgProc. When the "Old" field and the "New" field are the same, no change is made for that attribute.

Default Processing

The WinDefDlgProc function does not expect to receive this message and takes no action on it other than to return 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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message notifies an application that the dialog is about to update the preview area.

Default Processing

The WinDefDlgProc function does not expect to receive this message and takes no action on it other than to return 0.

Language Support Window Processing

This system-provided window procedure processes messages for a window that has been created with a window class specifying a "NULL" window procedure.

The following describes the WM_* messages and the language support window procedure action.

For any other messages the Language Support Window Procedure performs the same actions as the Default Window Procedure.

WM_ACTIVATE (Language Support Window)

For the cause of this message, see "WM_ACTIVATE" on page 12-3.

Parameters

For a description of the parameters, see "WM_ACTIVATE" on page 12-3.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PACTIVATE message to the application queue and setting *fIreply* to 0.

Default Processing

The default window procedure takes no action on this message, other than to set *fIreply* to 0.

WM_CONTROL (Language Support Window)

For the cause of this message, see "WM_CONTROL" on page 12-28.

Parameters

For a description of the parameters, see "WM_CONTROL" on page 12-28.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PCONTROL message to the application queue and setting *fIreply* to 0.

Default Processing

The default window procedure takes no action on this message, other than to set *fIreply* to 0.

WM_PAINT (Langauge Support Window)

For the cause of this message, see "WM_PAINT" on page 12-47.

Parameters

For a description of the parameters, see "WM_PAINT" on page 12-47.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PPAINT message to the application queue and setting *fIreply* to 0.

The WinBeginPaint and WinEndPaint functions are issued by the Language Support Window Procedure, during the processing of the WM_PPAINT message.

Default Processing

The default window procedure issues the WinBeginPaint and WinEndPaint functions, and then sets *fIreply* to 0.

WM_PPAINT (Language Support Window)

For the cause of this message, see "WM_PPAINT" on page 12-48.

Parameters

For a description of the parameters, see "WM_PPAINT" on page 12-48.

Remarks

The Language Support Window Procedure issues the WinBeginPaint and WinEndPaint functions, and then sets *flreply* to 0.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SETFOCUS (Language Support Window)

For the cause of this message, see "WM_SETFOCUS" on page 12-58.

Parameters

For a description of the parameters, see "WM_SETFOCUS" on page 12-58.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PSETFOCUS message to the application queue and setting *flreply* to 0.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SIZE (Language Support Window)

For the cause of this message, see "WM_SIZE" on page 12-61.

Parameters

For a description of the parameters, see "WM_SIZE" on page 12-61.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PSIZE message to the application queue and setting *flreply* to 0.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SYSCOLORCHANGE (Language Support Window)

For the cause of this message, see "WM_SYSCOLORCHANGE" on page 12-63.

Parameters

For a description of the parameters, see "WM_SYSCOLORCHANGE" on page 12-63.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PSYSCOLORCHANGE message to the application queue and setting *flreply* to 0.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

Language Support Dialog Processing

This system-provided window procedure processes messages for a dialog that has been created or loaded specifying a 'NULL' dialog procedure.

For any other messages the Language Support Dialog Procedure issues and returns the result of the WinDefDlgProc function.

WM_ACTIVATE (Language Support Dialog)

For the cause of this message, see "WM_ACTIVATE" on page 12-3.

Parameters

For a description of the parameters, see "WM_ACTIVATE" on page 12-3.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc function, then posting a WM_PACTIVATE message to the application queue and setting *flreply* to the result of the WinDefDlgProc function.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_CONTROL (Language Support Dialog)

For the cause of this message, see "WM_CONTROL" on page 12-28.

Parameters

For a description of the parameters, see "WM_CONTROL" on page 12-28.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc function, then posting a WM_PCONTROL message to the application queue and setting *flreply* to the result of the WinDefDlgProc function.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_PAINT (Language Support Dialog)

For the cause of this message, see "WM_PAINT" on page 12-47.

Parameters

For a description of the parameters, see "WM_PAINT" on page 12-47.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc function, then posting a WM_PPAINT message to the application queue and setting *flreply* to the result of the WinDefDlgProc function.

The WinBeginPaint and WinEndPaint functions are issued by the Language Support Dialog Procedure, during the processing of the WM_PPAINT message.

Default Processing

The default window procedure issues the WinBeginPaint and WinEndPaint functions, and then sets *flreply* to 0.

WM_PPAINT (Language Support Dialog)

For the cause of this message, see "WM_PPAINT" on page 12-48.

Parameters

For a description of the parameters, see "WM_PPAINT" on page 12-48.

Remarks

The Language Support Dialog Procedure issuing the WinDefDlgProc function, then issues the WinBeginPaint and WinEndPaint functions, and then setting *flreply* to the result of the WinDefDlgProc function.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SETFOCUS (Language Support Dialog)

For the cause of this message, see "WM_SETFOCUS" on page 12-58.

Parameters

For a description of the parameters, see "WM_SETFOCUS" on page 12-58.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc function, then posting a WM_PSETFOCUS message to the application queue and setting *flreply* to the result of the WinDefDlgProc function.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SIZE (Language Support Dialog)

For the cause of this message, see "WM_SIZE" on page 12-61.

Parameters

For a description of the parameters, see "WM_SIZE" on page 12-61.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc function, then posting a WM_PSIZE message to the application queue and setting *flreply* to the result of the WinDefDlgProc function.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_SYSCOLORCHANGE (Language Support Dialog)

For the cause of this message, see "WM_SYSCOLORCHANGE" on page 12-63.

Parameters

For a description of the parameters, see "WM_SYSCOLORCHANGE" on page 12-63.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc function, then posting a WM_PSYSCOLORCHANGE message to the application queue and setting *flreply* to the result of the WinDefDlgProc function.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

Chapter 13. Button Control Window Processing

This system-provided window procedure processes the actions on a button control (WC_BUTTON).

Purpose

A button control is a small rectangular child window representing a button that the operator can "switch" on or off. Button controls can be used alone or in groups, and can either be labeled or appear without text. Button controls typically change appearance when the operator clicks a pointing device on them or pressing the space bar when the button has the keyboard focus.

Buttons can be disabled to prevent them from responding when the operator clicks on them. Disabled buttons are displayed using a different emphasis technique (for example, color or half-toning).

Button Control Styles

These button control styles are available:

BS_PUSHBUTTON A pushbutton is a box that contains a string. When a button is pushed, by clicking the pointing device on it or pressing the spacebar when it is active, the parent window is notified.

BS_CHECKBOX A check box is a small square with a character string to the right. If it is checked, a small black box appears inside the small square. When the box or string is clicked, by clicking on it with the pointing device or pressing the keyboard spacebar when it is active, the check box changes state and the parent window is notified.

BS_AUTOCHECKBOX An automatic check box automatically toggles its state whenever the user clicks on it.

BS_RADIOBUTTON A radio button is similar to a check box, but is typically used in groups in which only one button at a time is checked. When a radio button is clicked or a cursor key is pressed to move within the group, it notifies its owner window. It is then up to the owner window to check the clicked radio button and uncheck all the rest, if necessary.

BS_AUTORADIOBUTTON When clicked, an automatic radio button automatically checks itself and unchecks all other radio buttons in the same group.

BS_3STATE A three-state check box is identical to a check box control except that its check box can be half-toned as well as the box being checked or unchecked.

BS_AUTO3STATE An automatic three-state check box automatically toggles its state when the user clicks on it.

BS_USERBUTTON This is an application-definable button. The owner window of this style control receives the additional button style BN_PAINT.

This style can be ORed with any of the basic button styles:

BS_NOPOINTERFOCUS Buttons with this style do not set the focus to themselves when clicked with the pointing device. This enables the cursor to stay on a control for which information is required, rather than moving to the button. This style has no effect on keyboard interaction. The tab key can still be used as usual to move the focus to the button.

BS_ICON Places an icon instead of text on the push button control.

BS_AUTOSIZE Buttons with this style will be sized to make sure the contents fit.

This style can be ORed with the BS_AUTORADIOBUTTON style:

BS_NOCURSORSELECT The radio button does not select itself when given the focus as the result of an arrow key or tab key.

These styles can be ORed with the BS_PUSHBUTTON style:

BS_HELP The button posts a WM_HELP message rather than a WM_COMMAND message.

BS_SYSCOMMAND The button posts a WM_SYSCOMMAND message rather than a WM_COMMAND message.

BS_NOBORDER The pushbutton is displayed without a border drawn around it. There is no other change in the pushbutton's operation.

If both BS_HELP and BS_SYSCOMMAND are set, BS_HELP takes precedence.

This style can be ORed with the BS_PUSHBUTTON and BS_USERBUTTON styles:

BS_DEFAULT A BS_DEFAULT pushbutton is one with a thick border box. It has the same properties as a pushbutton. In addition, the user may press a BS_DEFAULT pushbutton by pressing the RETURN or ENTER key. The intention is the same for user-buttons, but the appearance of a BS_DEFAULT userbutton is application defined.

Button Control Data

See BTNCDATA on page A-9.

Default Colors

The following system colors are used when the system draws button controls:

SYSCLR_BUTTONLIGHT
SYSCLR_WINDOW
SYSCLR_MENUTEXT
SYSCLR_BUTTONDEFAULT
SYSCLR_BUTTONMIDDLE
SYSCLR_WINDOW
SYSCLR_WINDOWFRAME.

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_HILITEFOREGROUNDCOLOR
PP_FOREGROUNDCOLOR
PP_DISABLEDFOREGROUNDCOLOR
PP_HILITEFOREGROUNDCOLOR
PP_BACKGROUNDCOLOR
PP_BORDERCOLOR.

Button Control Notification Messages

These messages are initiated by the button control window to notify its owner of significant events.

WM_COMMAND (in Button Controls)

For the cause of this message, see "WM_COMMAND" on page 12-27.

Parameters

For a description of the parameters, see "WM_COMMAND" on page 12-27.

Button control sets *uscmd* to the button identity and *ussource* to CMDSRC_PUSHBUTTON.

Remarks

The button control generates this message when a pushbutton of style BS_PUSHBUTTON is pressed or when it receives a BM_CLICK message. The button control posts the message to the queue of the control owner.

Default Processing

The default window procedure takes no action on this message, other than to set *reply* to 0.

WM_CONTROL (in Button Controls)

For the cause of this message, see "WM_CONTROL" on page 12-28.

Parameters

param1

idid *(USHORT)*
Button control identity.

usnotifycode *(USHORT)*
Notification code.

The notification code BN_PAINT is only generated when the button control has a style of BS_USERBUTTON.

The button control uses these notification codes:

BN_CLICKED	The button has been pressed.
BN_DBLCLICKED	The button has been double-clicked.
BN_PAINT	The button requires painting, using one of the following draw states:
	BDS_DISABLED The disabled state of the button requires painting.
	BDS_HILITED The highlighted state of the button requires painting.
	BDS_DEFAULT The default state of the button requires painting.

param2

flcontrolspec *(ULONG)*
Control-specific information.

When *usnotifycode* is BN_PAINT this parameter is a pointer to a USERBUTTON structure, otherwise this parameter is the window handle of the button control.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The button control generates this message and sends it to its owner, informing the owner of this event, when:

- Its style is not BS_PUSHBUTTON and the button is pressed.
- It receives a BM_CLICK message.
- Its style is BS_USERBUTTON and the button is clicked or double clicked.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_HELP (in Button Controls)

For the cause of this message, see "WM_HELP" on page 12-36.

Parameters

For a description of the parameters, see "WM_HELP" on page 12-36.

Button control sets *uscmd* to the button identity.

Remarks

This message is identical to a WM_COMMAND message, but implies that the application should respond to this message by displaying help information.

The button control generates this message and posts it to the queue of its owner, if it has the style of BS_HELP and a pushbutton is pressed, or when it receives a BM_CLICK message.

Default Processing

The default window procedure sends this message to the parent window, if it exists and is not the desktop. Otherwise, it sets *flreply* to 0.

WM_SYSCOMMAND

For the cause of this message, see "WM_SYSCOMMAND" on page 12-63.

Parameters

For a description of the parameters, see "WM_SYSCOMMAND" on page 12-63.

Button control sets *uscmd* to the button identity.

Remarks

If the button control is specified with a style of BS_SYSCOMMAND but not with BS_HELP, the button control generates this message and posts it to the queue of its owner when a pushbutton is pressed, or when it receives a BM_CLICK message.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

Button Control Window Messages

This section describes the Button Control Window Procedure actions on receiving the following messages.

BM_CLICK

An application sends this message to cause the effect of the operator clicking a pushbutton.

Parameters

param1

usUp *(USHORT)*
Up and down indicator:

TRUE	Perform the default upclick action
FALSE	Perform the default downclick action.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The button control responds to this message by taking the action that occurs if the button is clicked by the operator. This causes the following messages to be generated:

- A WM_HELP (in Button Controls) message, if the button has a style of BS_HELP.
- A WM_SYSCOMMAND message, if the button has a style of BS_PUSHBUTTON and a style of BS_SYSCOMMAND and not a style of BS_HELP.
- A WM_COMMAND (in Button Controls) message, if the button has a style of BS_PUSHBUTTON but not a style of BS_SYSCOMMAND and not a style of BS_HELP.
- A WM_CONTROL (in Button Controls) message, whose *usnotifycode* is set to BN_CLICKED, if the button has a style of BS_USERBUTTON, BS_PUSHBUTTON, BS_CHECKBOX, or BS_3STATE, and not a style of BS_SYSCOMMAND or BS_HELP.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fireply* to the default value of 0.

BM_QUERYCHECK

This message returns the checked state of a button control.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

usresult *(USHORT)*
Check indicator:

0 The button control is in unchecked state.
1 The button control is in checked state.
2 The button control is in indeterminate state.

Remarks

The button control responds to this message, if it has a style of BS_CHECKBOX, BS_AUTOCHECKBOX, BS_RADIOBUTTON, BS_AUTORADIOBUTTON, BS_3STATE, or BS_AUTO3STATE, by setting *usresult* as appropriate.

If the button has any other style, the button control takes no action other than to set *usresult* to 0.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *usresult* to the default value of 0.

BM_QUERYCHECKINDEX

This message returns the zero-based index of a checked radio button.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sresult *(SHORT)*
Radio-button index:

–1 No radio button of the group is checked, or this button control does not have the style BS_RADIOBUTTON or BS_AUTORADIOBUTTON.
Other Zero-based index of the checked radio button of the group.

Remarks

The button control responds to this message by setting *sresult* as appropriate.

This message may be sent to any radio button or autoradio button in a group of buttons. For details of the WS_GROUP style, see "Window Styles" on page 12-2.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sresult* to the default value of 0.

BM_QUERYHILITE

This message returns the highlighting state of a button control.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fresult *(BOOL)*
Highlight indicator:

TRUE The button control is displayed in highlighted state.
FALSE The button control is displayed in unhighlighted state.

Remarks

The button control responds to this message, if it has a style of BS_PUSHBUTTON, by setting *fresult* as appropriate.

If the button has any other style, the button control takes no action other than to set *fresult* to FALSE.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, except to set *fresult* to the default value of FALSE.

BM_SETCHECK

This message sets the checked state of a button control.

Parameters

param1

uscheck *(USHORT)*
Check state:

0 Display the button control in the unchecked state
1 Display the button control in the checked state
2 Display a 3-state button control in the indeterminate state.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

usoldstate *(USHORT)*
Old check state of the button control:

0 Unchecked
1 Checked
2 Indeterminate.

Remarks

The button control responds to this message by displaying it in the appropriate state and returning the old state.

If the button control has the style of BS_CHECKBOX, BS_AUTOCHECKBOX, BS_RADIOBUTTON, or BS_AUTORADIOBUTTON, it is displayed in the checked state if *uscheck* is set to 1, or in the unchecked state if it is set to 0 and *usoldstate* is set as appropriate.

If the button control has the style of BS_RADIOBUTTON or BS_AUTORADIOBUTTON, the WS_TABSTOP style is modified. If the resulting state of the button is checked, the WS_TABSTOP style is set, otherwise it is reset.

If the button control has the style of BS_3STATE or BS_AUTO3STATE, it is displayed in the unchecked state if *uscheck* is set to 0, in the checked state if it is set to 1, and in the indeterminate state if it is set to 2 and *usoldstate* is set as appropriate.

If the button control has the style of BS_USERBUTTON, a WM_CONTROL (in Button Controls) message is sent to its owner with *usnotifycode* set to BN_PAINT and *usoldstate* is set as appropriate.

If the button control has any other style, the button control takes no action other than to set *usoldstate* to 0.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, except to set *usoldstate* to the default value of 0.

BM_SETDEFAULT

This message sets the default state of a button control.

Parameters

param1

usdefault *(USHORT)*
Default state:

TRUE Display the button control in the default state
FALSE Display the button control in the nondefault state.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful operation
FALSE Error occurred.

Remarks

The button control responds to this message, if it has a style of BS_USERBUTTON or BS_PUSHBUTTON, by displaying the button control in the default or nondefault state as appropriate, and setting *fSuccess* to TRUE.

If the button control has any other style, the button control takes no action other than to set *fSuccess* to FALSE.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

BM_SETHILITE

This message sets the highlight state of a button control.

Parameters

param1

ushilite *(USHORT)*
Highlight indicator:

TRUE	Display the button control in the highlighted state
FALSE	Display the button control in the unhighlighted state.

param2 *(ULONG)*
Reserved.

0	Reserved value, 0.

Returns

reply

foldstate *(BOOL)*
Old highlight state:

TRUE	The button control was in highlighted state
FALSE	The button control was in unhighlighted state.

Remarks

The button control responds to this message, if it has a style of BS_PUSHBUTTON, BS_CHECKBOX, BS_AUTOCHECKBOX, BS_RADIOBUTTON, BS_AUTORADIOBUTTON, BS_3STATE, or BS_AUTO3STATE, by displaying the button control in the appropriate highlight state and setting *foldstate* as appropriate.

If the style of the Button Control is BS_USERBUTTON, a WM_CONTROL (in Button Controls) message is sent to its owner with *usnotifycode* set to BN_PAINT and with *flcontrolspec* pointing to a USERBUTTON structure and sets *foldstate* as appropriate.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *foldstate* to the default value of FALSE.

WM_ENABLE (in Button Controls)

For the cause of this message, see "WM_ENABLE" on page 12-31.

Parameters

For a description of the parameters, see "WM_ENABLE" on page 12-31.

Remarks

The button control window procedure responds to this message by setting the enable state and by setting *fireply* to 0.

Default Processing

The default window procedure takes no action on this message, other than to set *fireply* to 0.

WM_MATCHMNEMONIC (in Button Controls)

For the cause of this message, see "WM_MATCHMNEMONIC" on page 12-40.

Parameters

For a description of the parameters, see "WM_MATCHMNEMONIC" on page 12-40.

Remarks

The button control window procedure responds to this message by setting *fresult* as appropriate. If MP1 matches the button mnemonic, return *fresult* to TRUE.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_QUERYCONVERTPOS (in Button Controls)

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-51.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-51.

Remarks

The button control window procedure returns QCP_NOCONVERT.,

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-51.

WM_QUERYWINDOWPARAMS (in Button Controls)

Occurs when an application queries the button control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Remarks

The button control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure sets the *ulText*, *ulPresParams*, and *ulCtlData* parameters of the WNDPARAMS data structure, identified by *pwndparams*, to zero and sets *fresult* to FALSE.

WM_SETWINDOWPARAMS (in Button Controls)

Occurs when an application sets or changes the button control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-60.

Remarks

The button control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 14. Entry Field Control Window Processing

This system-provided window procedure processes the actions on an entry field control (WC_ENTRYFIELD).

Purpose

An entry field control is a rectangular window that displays a single line of text that the operator can edit. When it has the focus, the cursor marks the current **insertion** or **replacement** point.

When working with entry fields, the WM_CONTROL message is of major concern. An entry-field control communicates with its owner by sending WM_CONTROL messages. It contains a notification code in MP1 and a handle to the current entry field in MP2. The return value for WM_CONTROL is 0. Notification codes are denoted by an EN prefix.

Entry Field Control Styles

These entry field control styles are available:

ES_LEFT The text in the control is left-justified. This is the default style if neither ES_RIGHT nor ES_CENTER is specified.

ES_RIGHT The text in the control is right-justified.

ES_CENTER The text in the control is centered.

ES_AUTOSIZE The text will be sized to make sure the contents fit.

ES_AUTOSCROLL If the user tries to move off the end of a line, the control automatically scrolls one-third the width of the window in the appropriate direction.

ES_MARGIN This style can be used to cause a border to be drawn around the control, with a margin around the editable text. The margin is half a character-width wide and half a character-height high.

When an entry field control with this style is positioned, it adjusts the position so that the text is placed at the position specified. This position differs from the original position by the width of the border and the margin.

ES_READONLY This style causes a single line entry field to be created in read only state.

When an entry field is in read only state, characters do not get inserted into the text. However the insertion interface is still functional.

The entry field read only state can be altered by use of the EM_SETREADONLY message.

ES_UNREADABLE This style causes the text to be displayed as an asterisk for each character. It can be used for passwords.

ES_COMMAND This style identifies the entry field as a command entry field. This information is used by the Help Manager to provide command help if the end user requests help for this field.

Not more than one entry field on each dialog should be given this style.

ES_AUTOTAB This style indicates that when the field is filled by adding a character to the end of the entry field text, the effect of a tab key will be generated. Inserting or replacing a character in the middle of the text, however, does not result in an autotab.

This style is recommended for use with fixed-length, non-scrollable fields that are filled completely. The maximum length of the entry field text is held in the control data, see "Entry Field Control Data" on page 14-2

These entry field controls are intended for countries that use a double-byte character encoding scheme:

ES_SBCS The text is purely single-byte.

If the number of characters entered exceeds EM_SETTEXTLIMIT, or a DBCS character is entered, the alarm sounds and the last character entered is ignored.

ES_DBCS The text is purely double byte.

If the number of bytes in the entry field exceeds EM_SETTEXTLIMIT, or an SBCS character is entered, the alarm sounds and the last character entered is ignored.

ES_ANY The text is a mixture of SBCS and DBCS characters.

If the number of bytes in the input field exceeds EM_SETTEXTLIMIT, the alarm sounds and the last character entered is ignored.

ES_ANY is the default.

Note: If the queue code page is an ASCII code page and the data in the entry field is to be converted to an EBCDIC code page, there is a possibility that shift-in and shift-out characters introduced by the conversion process can cause the converted data to overrun the target field. Coding ES_MIXED protects the target field from overrun in this situation.

ES_MIXED The text is a mixture of SBCS and DBCS characters which may subsequently be converted from an ASCII DBCS code page to an EBCDIC DBCS code page with a consequent possible increase in the length of the data.

If

```
DBCSchars*2 + SBCSchars + N > EM_SETTEXTLIMIT
```

where N starts at 0 and is incremented whenever the string goes from SBCS to DBCS or DBCS to SBCS, the alarm sounds and the last character entered is ignored.

Note: For every conversion from SBCS to DBCS there must be a corresponding return to SBCS (N must be an even number).

Entry Field Control Data

See ENTRYFDATA on page A-34.

Default Colors

The following system colors are used when the system draws button controls:

SYSCLR_ENTRYFIELD
SYSCLR_BUTTONDARK
SYSCLR_BUTTONLIGHT
SYSCLR_OUTPUTTEXT
SYSCLR_WINDOWTEXT
SYSCLR_HIGHLITEFOREGROUND
SYSCLR_HIGHLITEBACKGROUND

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_FOREGROUNDCOLOR
PP_DISABLEDFOREGROUNDCOLOR
PP_HIGHLIGHTFOREGROUNDCOLOR
PP_FONTNAMESIZE

Entry Field Control Notification Messages

This message is initiated by the entry field control window to notify its owner of significant events.

WM_CONTROL (in Entry Fields)

For the cause of this message, see "WM_CONTROL" on page 12-28.

Parameters

param1

idId *(USHORT)*
Control window identity.

usnotifycode *(USHORT)*
Notify code:

EN_CHANGE	The content of the entry field control has changed, and the change has been displayed on the screen.
EN_KILLFOCUS	The entry field control is losing the focus.
EN_MEMERROR	The entry field control cannot allocate the storage necessary to accommodate window text of the length implied by the EM_SETTEXTLIMIT message.
EN_OVERFLOW	The entry field control cannot insert more text than the current text limit. The text limit may be changed with the EM_SETTEXTLIMIT message. If the recipient of this message returns TRUE, then the entry field control retries the operation, otherwise it terminates the operation.
EN_SCROLL	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.
EN_SETFOCUS	The entry field control is receiving the focus.

param2

hwndcontrolspec *(HWND)*
Entry field control window handle.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The entry field control window procedure generates this message and sends it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set *fireply* to 0.

Entry Field Control Window Messages

This section describes the entry field control window procedure actions on receiving these messages:

EM_CLEAR

This message deletes the text that forms the current selection.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The entry field control window procedure responds to this message by deleting the text that forms the current selection and setting **maxsel** equal to **minsel**.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

EM_COPY

This message pastes the current selection to the clipboard.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The entry field control window procedure responds to this message by pasting the text that forms the current selection to the clipboard in CF_TEXT format.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

EM_CUT

This message pastes the text that forms the current selection to the clipboard, and then deletes it from the entry field control.

Parameters

param1 *(ULONG)*

Reserved.

0 Reserved value, 0.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*

Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The entry field control window procedure responds to this message by pasting the text that forms the current selection to the clipboard in CF_TEXT format, and then deleting it from the entry field control and setting **maxsel** equal to **minsel**.

This message is the combination of a EM_COPY message followed by a EM_CLEAR message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

EM_PASTE

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

Parameters

param1 *(ULONG)*

Reserved.

0 Reserved value, 0.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

For example, if the text to be inserted does not fit in the entry field control without overflowing the text limit set by the EM_SETTEXTLIMIT message, in which instance no text is inserted.

Remarks

The entry field control window procedure responds to this message by replacing the text that forms the current selection with text from the clipboard, if the data is in CF_TEXT format.

Only characters from the clipboard up to the first carriage return are used in the replacement.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

EM_QUERYCHANGED

This message enquires if the text of the entry field control has been changed since the last enquiry.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fchanged *(BOOL)*
Changed indicator:

TRUE The text in the entry field control has been changed since the last time it received this message or a WM_QUERYWINDOWPARAMS message.
FALSE All other situations.

Remarks

The entry field control window procedure responds to this message by setting *fchanged* to indicate whether the text of the entry field has been changed since the last time either this message or a WM_QUERYWINDOWPARAMS (in Entry Fields) message has been received.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fchanged* to the default value of FALSE.

EM_QUERYFIRSTCHAR

This message returns the zero-based offset of the first character visible at the left edge of an entry-field control.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sOffset *(SHORT)*

Zero-based offset of the first character visible at the left edge of an entry-field control.

Remarks

The entry field control window procedure responds to this message by returning the zero-based offset into the text that corresponds to the first character displayed in the entry field control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sOffset* to the default value of 0.

EM_QUERYREADONLY

This message returns the read only state of an entry field control.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fReadOnly *(BOOL)*
Read only state indicator:

TRUE Read only state is enabled.
FALSE Read only state is disabled.

Remarks

The entry field control window procedure responds to this message by returning the read only state of the entry field control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fReadOnly* to the default value of FALSE.

EM_QUERYSEL

This message gets the zero-based offsets of the bounds of the text that forms the current selection.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sMinSel *(SHORT)*
Offset of the first character in the selection.

sMaxSel *(SHORT)*
Offset of the first character after the selection.

Remarks

The entry field control window procedure responds to this message by returning the zero-based offsets of the bounds of the text that forms the current selection.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *reply* to the default value of 0, which is equivalent to setting both *sMinSel* and *sMaxSel* to 0.

EM_SETFIRSTCHAR

This message specifies the offset of the character to be displayed in the first position of the entry field control.

Parameters

param1

sOffset *(SHORT)*
Zero-based offset of the first character to be displayed.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred. For example, because *sOffset* is not valid.

Remarks

The entry field control window procedure responds to this message by setting the text displayed in the edit control so that the first character displayed on the left of the window has the zero-based index specified by *sOffset*.

An EN_SCROLL notification message occurs, if the entry field control scrolls. This message returns FALSE if the edit control does not have the ES_AUTOSCROLL style or it is center of right justified.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

EM_SETINSERTMODE

This message sets the insert mode of an entry field.

Parameters

param1

usInsert *(USHORT)*
Insert mode indicator:

TRUE	Enable insert mode.
FALSE	Enable overtype mode.

param2 *(ULONG)*
Reserved.

0	Reserved value, 0.

Returns

reply

fOldInsertMode *(BOOL)*
Previous insert mode indicator:

TRUE	Insert mode was previously enabled.
FALSE	Overtype mode was previously enabled.

Remarks

The entry field control window procedure responds to this message by setting the insert mode of the entry field, updating the SV_INSERTMODE system constant and redrawing the entry field.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fOldInsertMode* to the default value of FALSE.

EM_SETREADONLY

This message sets the read only state of an entry field control.

Parameters

param1

usReadOnly *(USHORT)*
Read only state indicator:

TRUE Enable read only state
FALSE Disable read only state.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fOldReadOnly *(BOOL)*
Previous read only state indicator:

TRUE Read only state was previously enabled.
FALSE Read only state was previously disabled.

Remarks

The entry field control window procedure responds to this message by setting the read only state of the entry field control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fOldReadOnly* to the default value of FALSE.

EM_SETSEL

This message sets the zero-based offsets of the bounds of the text that forms the current selection.

Parameters

param1

usminsel *(USHORT)*
Offset of the first character in the selection.

usmaxsel *(USHORT)*
Offset of the first character after the selection.

If *usminsel* equals *usmaxsel*, the current selection becomes an insertion point.

If *usminsel* equals 0 and *usmaxsel* is equal to or greater than the text limit set by the EM_SETTEXTLIMIT message, the entire text is selected. Selected text is displayed in reverse color.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The entry field control window procedure responds to this message by setting the zero-based offsets of the bounds of the text that forms the current selection.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

EM_SETTEXTLIMIT

This message sets the maximum number of bytes that an entry field control can contain.

Parameters

param1

sTextLimit *(SHORT)*
Maximum number of characters in the entry field control.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred. For example, because not enough storage can be allocated.

Remarks

The entry field control window procedure responds to this message by setting the maximum number of characters that can be contained.

This message is intended only to limit the length of lines that result from the user interacting with the entry field control. It also limits the length of text that can result from sending a EM_PASTE or WM_SETWINDOWPARAMS message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

WM_CHAR (in Entry Fields)

For the cause of this message, see "WM_CHAR" on page 12-24.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-24.

Remarks

The entry field control window procedure responds to this message by sending it to its owner if it has not processed the keystroke. This is the most common means by which the input focus is switched around the various controls in a dialog box.

Unlike other controls, the *usvk* field of the message "WM_CHAR" on page 12-24. takes precedence over other fields only when the Shift key is pressed.

If this message contains a valid *usch* field of the message "WM_CHAR" on page 12-24. that character is entered into the text in insert or overtype mode.

The keystrokes processed by an entry field control are:

Left arrow	Move the cursor one character to the left.
Right arrow	Move the cursor one character to the right.
Shift + Left arrow	Extend the selection by one character to the left.
Shift + Right arrow	Extend the selection by one character to the right.
Home	Move the cursor to the beginning of the text.
End	Move the cursor to the end of the text.
Backspace	Delete the character to the left of the cursor.
Delete	When the selection is an insertion point, delete the character to the right of the cursor, otherwise delete the current selection, but do not put it in the clipboard.
Shift + Del	Cut the current selection to the clipboard.
Shift + Ins	Replace the current selection with the text contents from the clipboard.
Ctrl + Del	Delete to the end of the field.
Ctrl + Ins	Copy the current selection to the clipboard.

If the control contains more text than can be shown, the actions defined above that move the cursor cause the text to be scrolled. The amount of scrolling varies from key to key, and the position of the text within the control varies for the same cursor position.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set *fresult* to FALSE.

WM_QUERYCONVERTPOS (in Entry Fields)

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-51.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-51.

Remarks

The entry field control window procedure updates *pCursorPos* to the position of the cursor and returns QCP_CONVERT.

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-51.

WM_QUERYWINDOWPARAMS (in Entry Fields)

This message occurs when an application queries the entry field control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Remarks

The entry field control window procedure responds to this message by returning the window parameters indicated by the *ulStatus* parameter of the WNDPARAMS data structure identified by the *pwndparams* parameter.

Default Processing

The default window procedure sets the *ulText*, *ulPresParams*, and *ulCtlData* parameters of the WNDPARAMS data structure, identified by *pwndparams*, to 0 and sets *fresult* to FALSE.

WM_SETWINDOWPARAMS (in Entry Fields)

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

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-60.

Remarks

The entry field control window procedure responds to this message by setting the window parameters indicated by the *ulStatus* parameter of the WNDPARAMS data structure, identified by the *pwndparams* parameter.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 15. Frame Control Window Processing

This system-provided window procedure processes the actions on a frame window (WC_FRAME). The frame control window procedure sends all messages not processed to FID_CLIENT and sets **reply** to 0.

Purpose

The window that contains all of the parts listed below is called the *frame window*. Each of the parts that make up a window, such as the title bar and menu, are separate child windows of the frame window. All of these child windows, except the client window (FID_CLIENT), are called *frame controls*.

FID_CLIENT is not a frame control, it is an instance of a window class implemented by the application.

The frame window and all of the frame controls are implemented with system-provided preregistered window classes.

The frame window holds together all of the frame controls and FID_CLIENT that make up an application window. The frame window is responsible for arranging the frame controls and the FID_CLIENT as the frame window is sized and moved. It is also responsible for routing specific messages to its frame controls and the FID_CLIENT.

Each of the frame controls and FID_CLIENT are known to the frame window by a system-provided window-identifier value as listed below:

FID_CLIENT	Client window
FID_HORZSCROLL	Horizontal scroll bar
FID_MENU	Application menu
FID_MINMAX	Minimize/Maximize box
FID_SYSMENU	System menu
FID_TITLEBAR	Title bar
FID_VERTSCROLL	Vertical scroll bar.

For correct operation, only one window per frame must be defined with each of the above FID_* values.

Frame Creation Flags

These frame creation flags are available:

FCF_TITLEBAR	Title bar.
FCF_SYSMENU	System menu.
FCF_MENU	Application menu.
FCF_MINMAX	Minimize and Maximize buttons.
FCF_MINBUTTON	Minimize button.
FCF_MAXBUTTON	Maximize button.
FCF_VERTSCROLL	Vertical scroll bar.
FCF_HORZSCROLL	Horizontal scroll bar.
FCF_SIZEBORDER	Sizing border.
FCF_BORDER	Window is drawn with a thin border.
FCF_DLGBORDER	Window is drawn with a standard dialog border.

FCF_ACCELTABLE

Causes an accelerator table to be loaded, for this frame window, from the resource file identified on the WinCreateStdWindow function.

FCF_ICON

Window is created with an icon associated with it that is used to represent the window when it is minimized.

If present, the *Resource* parameter of the WinCreateStdWindow function must be the identity of an icon. This icon is loaded and associated with the window. When the window is minimized, the icon is shown if the screen is capable of showing it. When the window is destroyed, the icon is also destroyed.

FCF_SHELLPOSITION

The window is created with a size and position determined by the shell, rather than explicitly by the application.

FCF_SYSMODAL

The frame window is System Modal.

FCF_NOBYTEALIGN

When this flag is **not** set, the frame window is adjusted so that window operations, such as moving, can be performed in an optimized manner. For example, some displays can move a window more quickly if the movement is by a multiple of eight pels.

If this flag is set, such optimizations are not performed and size and position values are honored.

FCF_TASKLIST

When this flag is set, the program title is added to the front of the frame window text, the resulting string is used as the window title and is also entered on the task list.

In this context, the program title is the text string used by the Desktop Manager to identify the program, or the text string specified as a parameter in the START command. If neither string has been defined, the filename and extension of the .EXE file are used as the program title.

Note that a WinSetWindowText will not change the entry in the switch list, a WinChangeSwitchEntry must be done to affect this.

FCF_NOMOVEWITHOWNER

The window should not be moved when its owner is moved.

FCF_STANDARD

Same as (FCF_TITLEBAR | FCF_SYSMENU | FCF_MINBUTTON | FCF_MAXBUTTON | FCF_SIZEBORDER | FCF_ICON | FCF_MENU | FCF_ACCELTABLE | FCF_SHELLPOSITION | FCF_TASKLIST).

This value is assumed if any Frame Window is created with no Control Data.

FCF_SCREENALIGN

See FS_SCREENALIGN.

FCF_MOUSEALIGN

See FS_MOUSEALIGN.

FCF_AUTOICON

Performance optimization. When repainting iconized frames, the system will redraw the icon and will not send a WM_PAINT message to the application.

FCF_HIDEBUTTON

Hide button.

FCF_HIDEMAX

Hide and maximize buttons.

Frame Control Styles

These frame control styles are available. Frame styles may only be used when the frame is created from a dialog template.

FS_SCREENALIGN	The coordinates specifying the location of the dialog box are relative to the top left corner of the screen, rather than being relative to the owner window's origin.
FS_MOUSEALIGN	The coordinates specifying the location of the dialog box are relative to the position of the pointing device pointer at the time the window was created. The operating system tries to keep the dialog box on the screen, if possible.
FS_SIZEBORDER	See FCF_SIZEBORDER.
FS_BORDER	See FCF_BORDER.
FS_DLGBORDER	See FCF_DLGBORDER.
FS_SYSMODAL	See FCF_SYSMODAL.
FS_NOBYTEALIGN	See FCF_NOBYTEALIGN.
FS_TASKLIST	See FCF_TASKLIST.
FS_NOMOVEWITHOWNER	See FCF_NOMOVEWITHOWNER.
FS_AUTOICON	See FCF_AUTOICON.

Frame Control Data

See FRAMECDATA on page A-60.

Default Colors

The following system colors are used when the system draws button controls:

SYSCLR_DIALOGBACKGROUND
SYSCLR_ACTIVETITLE
SYSCLR_INACTIVETITLE
SYSCLR_APPWORKSPACE
SYSCLR_ACTIVEBORDER
SYSCLR_WINDOW
SYSCLR_SHADOW
SYSCLR_WINDOWFRAME
SYSCLR_FIRST.

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_BACKGROUNDCOLOR
PP_SHADOW
PP_FOREGROUNDCOLOR
PP_BORDERCOLOR
PP_DISABLEDBACKGROUNDCOLOR.

Frame Control Notification Messages

These messages are initiated by the frame control window to notify the FID_CLIENT window.

WM_MINMAXFRAME (in Frame Controls)

For the cause of this message, see "WM_MINMAXFRAME" on page 12-42.

Parameters

For a description of the parameters, see "WM_MINMAXFRAME" on page 12-42.

Remarks

The window words QWS_XRESTORE, QWS_YRESTORE, QWS_CXRESTORE, and QWS_CYRESTORE for *hwnd* are initialized before this message is sent. The window state has not been changed when this message is sent, and so the WinQueryWindowPos function can be used.

This message is sent by default to the FID_CLIENT window.

The system default actions, if FALSE is returned to this message, are based on the operation specified by the *pswp* parameter.

These actions affect the status of the frame window, and the title button windows and system menu windows contained within it, as follows:

- Window is maximized from a minimized state.
  - Title button windows:

    The RESTORE button window is replaced by a MIN button window and the MAX button window is replaced by a RESTORE button window.
  - System menu window:

    The MINIMIZE menu entry is enabled and the MAXIMIZE menu entry is disabled.
  - Other changes:

    The frame window has the WS_MAXIMIZED style bit set and the WS_MINIMIZED style bit reset. Also the MS_VERTICALFLIP style bit of the system menu window is reset.
- Window is restored from a minimized state.
  - Title button windows:

    The RESTORE button window is replaced by a MIN button window (the MAX button window is unaltered).
  - System menu window:

    The MINIMIZE menu entry is enabled, the RESTORE menu entry is disabled and the SIZE menu entry is enabled.
  - Other changes:

    The frame window has the WS_MINIMIZED style bit and the MS_VERTICALFLIP style bit of the system menu window reset.
- Window is minimized from a maximized state.
  - Title button windows:

    The RESTORE button window is replaced by a MAX button window and the MIN button window is replaced by a RESTORE button window.
  - System menu window:

    The MAXIMIZE menu entry is enabled and the MINIMIZE menu entry is disabled.
  - Other changes:

    The frame window has the WS_MINIMIZED style bit set and the WS_MAXIMIZED style bit reset. Also the MS_VERTICALFLIP style bit of the system menu window is set.

- Window is restored from a maximized state.
  - Title button windows:

    The RESTORE button window is replaced by a MAX button window (the MIN button window is unaltered).
  - System menu window:

    The MAXIMIZE menu entry is enabled, the RESTORE menu entry is disabled and the SIZE menu entry is enabled.
  - Other changes:

    The frame window has the WS_MAXIMIZED style bit reset.
- Window is minimized from a restored state.
  - Title-button windows:

    The MIN button window is replaced by a RESTORE button window (the MAX button window is unaltered).
  - System menu window:

    The RESTORE menu entry is enabled, the MINIMIZE menu entry is disabled and the SIZE menu entry is disabled.
  - Other changes:

    The frame window has the WS_MINIMIZED style bit set, and the MS_VERTICALFLIP style bit of the system menu window is set.
- Window is maximized from a restored state.
  - Title-button windows:

    The MAX button window is replaced with a RESTORE button window (the MIN button window is unaltered).
  - System menu window:

    The RESTORE menu entry is enabled, the MAXIMIZE menu entry is disabled.
  - Other changes:

    The frame window has the WS_MAXIMIZED style bit set.

Default Processing

The default window procedure takes no action on this message, other than to set *fOverrideDefault* to FALSE.

Frame Control Window Messages

This section describes the frame control window procedure actions on receiving the following messages.

WM_ACTIVATE (in Frame Controls)

For the cause of this message, see "WM_ACTIVATE" on page 12-3.

Parameters

For a description of the parameters, see "WM_ACTIVATE" on page 12-3.

Remarks

The frame control window procedure responds to this message by first sending a TBM_SETHILITE message to the FID_TITLEBAR control, if it exists, to highlight or unhighlight the title bar. If the style is FCF_DLGBORDER, the border is redrawn in either highlighted or unhighlighted state, as necessary.

It then sends the WM_ACTIVATE message to the FID_CLIENT window.

Then it sets *fIreply* to 0.

Default Processing

The default window procedure takes no action on this message, other than to set *fIreply* to 0.

WM_ADJUSTFRAMEPOS

This message is sent to a frame window whose position or size is to be adjusted.

Parameters

param1

pswp *(PSWP)*
New frame window state.

This points to a SWP structure.

The structure has been filled in by the WinSetWindowPos or WinSetMultWindowPos functions with the proposed move or size data for the frame window.

param2

hsvwphsvwp *(HSVWP)*
Identifier of the frame window repositioning process.

Returns

fIreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

When a WinSetWindowPos or WinSetMultWindowPos function involves adjusting the position or size of a frame window, a WM_ADJUSTFRAMEPOS message is sent to the frame window.

The frame control processes the message by informing all the windows in its owner hierarchy, that is all the windows owned by the frame and all the windows owned by them and so on, by sending each a WM_OWNERPOSCHANGE message. Each window receiving the a WM_OWNERPOSCHANGE message is expected to modify the SWP structure provided as the first parameter in the message to the appropriate values relative to the new position and/or size of its owner, whose new position and size is specified in a SWP structure provided as the second parameter in the message.

In this way the frame control can determine the state changes to be made to all the windows in its owner hierarchy, in accordance with the values specified in the SWP structure referenced by the *pswp* parameter. The rules for changing the state of these owned windows are:

SWP_SIZE and SWP_MOVE
The owned window is moved relative to the top left corner of its owner.

SWP_SHOW
The visibility state of an owned window is changed to agree with that of their owner.

SWP_MINIMIZE
An owned window is made invisible when the owner is minimized.

SWP_MAXIMIZE and SWP_RESTORE
An owned window that was previously made invisible when the owner was minimized is made visible.

The frame window coordinates the repositioning of the frame window and all its owned windows, by using the WinSaveWindowPos function to associate those windows whose states are to change with the identifier of the frame window repositioning process, that is the *hsvwphsvwp* parameter. Eventually, the state changes to be made to the owned windows are contained in the array of SWP structures identified by the *aswpaswp* parameter.

If the frame window is subclassed, this message must then be passed to the superclass window procedure for processing. The superclass window procedure is the window procedure of the window before it was subclassed. This message is passed along the chain of window procedures and is eventually processed by the system frame window procedure.

Default Processing
The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_BUTTON1DBLCLK (in Frame Controls)
For the cause of this message, see "WM_BUTTON1DBLCLK" on page 12-10.

Parameters
For a description of the parameters, see "WM_BUTTON1DBLCLK" on page 12-10.

Default Processing
If the frame is minimized, the frame control window procedure causes the frame window to return to its previous state. Otherwise, the message is handled like a WM_BUTTON1DOWN message.

WM_BUTTON2DBLCLK (in Frame Controls)
For the cause of this message, see "WM_BUTTON2DBLCLK" on page 12-11.

Parameters
For a description of the parameters, see "WM_BUTTON2DBLCLK" on page 12-11.

Default Processing
The frame control window procedure processes this message identically to WM_BUTTON1DBLCLK (in Frame Controls).

WM_BUTTON1DOWN (in Frame Controls)

For the cause of this message, see "WM_BUTTON1DOWN" on page 12-13.

Parameters

For a description of the parameters, see "WM_BUTTON1DOWN" on page 12-13.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

Default Processing

The frame control window procedure responds to this message by issuing the WinSetActiveWindow function and sets *fresult* to TRUE. If this is over a part of the window that does not have a frame control, it issues a WinSetActiveWindow function. If the click is over the size border, this window begins tracking by sending a WM_TRACKFRAME message to itself. If the click is not over the size border, this message is passed on.

WM_BUTTON2DOWN (in Frame Controls)

For the cause of this message, see "WM_BUTTON2DOWN" on page 12-15.

Parameters

For a description of the parameters, see "WM_BUTTON2DOWN" on page 12-15.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

Default Processing

The frame control window procedure processes this message identically to "WM_BUTTON1DOWN (in Frame Controls)."

WM_BUTTON1UP (in Frame Controls)

For the cause of this message, see "WM_BUTTON1UP" on page 12-19.

Parameters

For a description of the parameters, see "WM_BUTTON1UP" on page 12-19.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

Default Processing

The frame control window procedure responds to this message by issuing the WinSetActiveWindow function and sets *fresult* to TRUE. If the window is not minimized, this message is not processed. If the frame is minimized, this message causes the system menu to pop up.

WM_BUTTON2UP (in Frame Controls)

For the cause of this message, see "WM_BUTTON2UP" on page 12-20.

Parameters

For a description of the parameters, see "WM_BUTTON2UP" on page 12-20.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

Default Processing

The frame control window procedure processes this message identically to "WM_BUTTON1UP (in Frame Controls)" on page 15-8.

WM_CALCFRAMERECT (in Frame Controls)

For the cause of this message, see "WM_CALCFRAMERECT" on page 12-22.

Parameters

For a description of the parameters, see "WM_CALCFRAMERECT" on page 12-22.

Remarks

Frame control calculates the appropriate rectangle, taking into account byte alignment, or nonbyte alignment if FCF_NOBYTEALIGN is specified.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

WM_CHAR (in Frame Controls)

This message is sent by controls to their owner window if they do not process the key stroke themselves. It is the most common means by which the input focus is switched around the various controls in a dialog box.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-24.

Default Processing

The frame control window procedure responds to this message as follows:

- If the message contains a valid VK_ value, that value is processed before any valid character in the message.
- If the character matches a mnemonic in the text of a button or static control child window, the focus is set to that window.
- If the character is Tab or Backtab, the focus is set to the next or previous tabstop window.
- If the character is Up or Left Arrow, the focus is set to the previous item in the group.
- If the character is Down or Right Arrow, the focus is set to the next item in the group.
- If the Enter key is pressed, a WM_COMMAND message is posted to itself, containing the identity of the button with the focus, or, if none, the identity of the default pushbutton.
- If the Escape key is pressed, a WM_COMMAND message is posted to itself with the command value DID_CANCEL.

WM_CLOSE (in Frame Controls)

For the cause of this message, see "WM_CLOSE" on page 12-26.

Parameters

For a description of the parameters, see "WM_CLOSE" on page 12-26.

Remarks

Frame control sends this message to the client window (FID_CLIENT) if it exists, otherwise it calls the WinDefWindowProc function.

Default Processing

The default window procedure posts a WM_QUIT message to the appropriate queue and sets *flreply* to 0.

WM_COMMAND

For the cause of this message, see "WM_COMMAND" on page 12-27.

Parameters

For a description of the parameters, see "WM_COMMAND" on page 12-27.

Default Processing

The Frame Control window procedure responds to this message by sending it the client window if it exists, otherwise the message is thrown away.

WM_DRAWITEM (in Frame Controls)

For the cause of this message, see "WM_DRAWITEM" on page 12-31.

Parameters

For a description of the parameters, see "WM_DRAWITEM" on page 12-31.

Remarks

The identity of the top-level action-bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_ERASEBACKGROUND

This message causes a client window to be filled with the background, should this be appropriate.

Parameters

param1

hpshpsFrame *(HPS)*
Presentation-space handle for the frame window.

param2

pprcPaint *(PRECTL)*
Rectangle structure of rectangle to be painted.

This points to a RECTL structure.

Returns

reply

fresult *(BOOL)*
Processed indicator:

TRUE If a FID_CLIENT window exists, the area of the frame covered by the FID_CLIENT window is erased in the system-window background color.

If no FID_CLIENT window exists, the entire frame window is erased in the system-window background color.

FALSE The client window did process the message.

Remarks

The frame window procedure processes this message in the following manner:

1. The frame window sends this message to the client in response to the frame WM_PAINT message, with the presentation-space handle of the frame window (obtained from WinBeginPaint).
2. If the client window returns TRUE, the frame window procedure erases the rectangle of the frame window covered by the client window, by filling it with the system color SCLR_WINDOW.
3. If the client window returns FALSE, no action is taken. This is the default behavior, as WinDefWindowProc returns FALSE if passed this message.
4. Also, the client window can use the presentation-space handle passed in this message to selectively erase parts of the screen. If the client window processes the message in this way, FALSE should be returned to avoid the erasure being done automatically by the frame window procedure.

   It should be noted again that the presentation space is *not* a client window presentation space; it is a presentation space for the frame window returned by WinBeginPaint, that is, a cached presentation space in frame (not client) window coordinates, clipped to the area of the frame that needs to be updated (possibly including areas outside the client window).

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_FLASHWINDOW

An application has issued a WinFlashWindow function.

Parameters

param1

usFlash *(USHORT)*
Flash indicator:

TRUE Start the window border flashing
FALSE Stop the window border flashing.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fresult *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Default Processing

The frame control window procedure responds to this message from an application by starting or stopping the flashing of the window border, and by setting *fresult* as appropriate.

WM_FOCUSCHANGE (in Frame Controls)

For the cause of this message, see "WM_FOCUSCHANGE" on page 12-34.

Parameters

For a description of the parameters, see "WM_FOCUSCHANGE" on page 12-34.

Remarks

The frame control responds to this message by sending the other messages depending on the value of the *fsFocusChange* parameter. These messages, if sent, are sent in the following order:

1. WM_SETFOCUS to the window losing the focus.
2. WM_SETSELECTION to the windows losing their selection.
3. WM_ACTIVATE to the windows being deactivated.
4. WM_ACTIVATE to the windows being activated.
5. WM_SETSELECTION to the windows being selected.
6. WM_SETFOCUS to the window receiving the focus.

Default Processing

The default window procedure sends this message to either the owner, if one exists, or to the parent of the window, if it is not the desktop window, otherwise it sets *flReply* to 0.

WM_FORMATFRAME (in Frame Controls)

For the cause of this message, see "WM_FORMATFRAME" on page 12-35.

Parameters

For a description of the parameters, see "WM_FORMATFRAME" on page 12-35.

Remarks

Applications that subclass frame controls may find that the frame is already subclassed; the number of frame controls is variable.

The WM_FORMATFRAME and WM_QUERYFRAMECTLCOUNT messages must always be subclassed by calling the previous window procedure and modifying its result.

Default Processing

The SWP structure for the FID_CLIENT frame control, if present, is the last element of the *pswp* parameter, unless additional frame controls are added by subclassing; the SWP structures for these follow that for FID_CLIENT if present. The frame control window procedure first sends the message to the FID_CLIENT window. If FID_CLIENT returns *ccount* to indicate that the message has been processed, no additional processing is performed.

If not processed by the client, the frame control window procedure calculates the size and position of all the standard frame controls.

WM_INITMENU (in Frame Controls)

For the cause of this message, see "WM_INITMENU" on page 12-39.

Parameters

For a description of the parameters, see "WM_INITMENU" on page 12-39.

Remarks

The identity of the top-level action-bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT. If the identity is FID_SYSMENU the system menu state is initialized according to the current state of the window.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_MEASUREITEM (in Frame Controls)

For the cause of this message, see "WM_MEASUREITEM" on page 12-41.

Parameters

For a description of the parameters, see "WM_MEASUREITEM" on page 12-41.

Remarks

The identity of the top-level action bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sHeight* to the default value of 0.

WM_MENUSELECT (in Frame Controls)

For the cause of this message, see "WM_MENUSELECT (in Frame Controls)."

Parameters

For a description of the parameters, see "WM_MENUSELECT (in Frame Controls)."

Remarks

The identity of the top-level action-bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to TRUE.

WM_NEXTMENU (in Frame Controls)

For the cause of this message, see "WM_NEXTMENU" on page 12-44.

Parameters

For a description of the parameters, see "WM_NEXTMENU" on page 12-44.

Remarks

The frame control window procedure processes the message by returning the handle of the system menu window if *hwndMenu* is the handle of the main action bar window, or by returning the handle of the main action bar window if *hwndMenu* is the handle of the system menu window.

Default Processing

The default window procedure takes no action on this message, other than to set *hwndNewMenu* to NULLHANDLE.

WM_OWNERPOSCHANGE

This message is sent by a frame window processing the WM_ADJUSTFRAMEPOS message.

Parameters

param1

ppswp *(PSWP)*
Owned window state.

This points to a SWP structure.

The receiver of this message is expected to alter this SWP parameter to the appropriate values relative to the new position and/or size of its owner, whose new position and size is specified in a SWP structure in the *ppswpOwner* parameter.

param2

ppswpOwner *(PSWP)*
Owner window state.

This points to a SWP structure.

This represents the new position and size of the owner window.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_PAINT (in Frame Controls)

For the cause of this message, see "WM_PAINT" on page 12-47.

Parameters

For a description of the parameters, see "WM_PAINT" on page 12-47.

Default Processing

The frame is redrawn as governed by the FCF_BORDER or FCF_DLGBORDER style. A WM_ERASEBACKGROUND message is sent to FID_CLIENT window, and if it returns FALSE, then the FID_CLIENT window is erased to the system-provided window background color and sets *flreply* to 0.

WM_QUERYBORDERSIZE

This message is sent to the frame window to determine the width and height of the border of the window.

Parameters

param1

pSize *(PWPOINT)*
Width and height of size border control.

This points to a WPOINT structure, that is used to hold the width in the *x* parameter and the height in the *y* parameter.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fresult *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The frame window responds to this message by returning the width and height of its border in the *pSize* parameter, as follows:

- SV_CX/CYSIZEBORDER if FCF_SIZEBORDER is specified
- SV_CX/CYDLGFRAME if FCF_DLGBORDER is specified
- SV_CX/CYBORDER if FS_BORDER is specified.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fresult* to the default value of FALSE.

WM_QUERYCONVERTPOS (in Frame Controls)

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-51.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-51.

Remarks

The frame control window procedure returns QCP_NOCONVERT.,

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-51.

WM_QUERYFOCUSCHAIN

This message is used to request the handle of a window in the focus chain.

Parameters

param1

fsCmd *(USHORT)*
Command to be performed.

This field contains a flag to indicate what action is to be performed:

QFC_NEXTINCHAIN	Return the next window in the focus chain.
QFC_ACTIVE	The *hwndParent* parameter is not used. Return the handle of the frame window that would be activated or deactivated, if this window gains or loses the focus.
QFC_FRAME	The window handle returned is a child of the window specified by the *hwndParent* parameter. Return the handle of the first frame window associated with this window.
QFC_SELECTACTIVE	The *hwndParent* parameter is not used. Return the handle of the window from the group of owned windows to which this window belongs which either currently has the focus or, if no window has the focus, previously had the focus. Return NULL, if no window in the owner group has had the focus.
QFC_PARTOFCHAIN	The *hwndParent* parameter is not used. Return TRUE if the handle of the window identified by the *hwndParent* parameter is in the focus chain, otherwise return FALSE. Because this message is passed along the focus chain, this is equivalent to returning TRUE, if the handle of the window receiving this message is *hwndParent* or to returning FALSE, if it is not.

param2

hwndParent *(HWND)*
Parent window.

Returns

Reply

hwndResult *(HWND)*
Handle of the window requested.

0 No window handle exists for this case of the *fsCmd* parameter

This value is also to be interpreted as FALSE for the case when the *fsCmd* is set to QFC_PARTOFCHAIN.

Other Handle of the window requested.

This value is also to be interpreted as TRUE for the cases when the *fsCmd* is set to QFC_PARTOFCHAIN.

Remarks

The frame control window procedure responds to this message by returning the appropriate window handle, as described under the *fsCmd* field.

Default Processing

The default window procedure takes the same action as the frame control window procedure.

WM_QUERYFRAMECTLCOUNT

This message is sent to the frame window in response to the receipt of a WM_SIZE or a WM_UPDATEFRAME (in Frame Controls) message.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sControlCount *(SHORT)*
Count of frame controls.

Remarks

By sending this message to itself, any procedures that subclass the frame window become aware that the number of frame controls is being calculated and include any special frame controls of the subclass in the count.

This count is used to allocate the appropriate number of SWP structures that are passed in the WM_FORMATFRAME (in Frame Controls) message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sControlCount* to the default value of 0 which is equivalent to 0.

WM_QUERYFRAMEINFO

This message enables an application to query information about frame windows.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

flFlags *(ULONG)*
Frame information flags.

One or more of the following are returned:

FI_FRAME	Identifies a frame window.
FI_OWNERHIDE	The frame window is hidden when its owner is hidden.
FI_NOMOVEWITHOWNER	The frame window does not move with its owner.
FI_ACTIVATEOK	The frame window may be activated. This means, for example, that the frame window is not disabled.

Remarks

This message can be used to query whether or not a particular window is a frame window.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_QUERYICON

This message is sent to a frame window to query its associated icon.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

hptrIcon *(HPOINTER)*
Handle to the icon.

Default Processing

The icon for the frame is returned.

WM_QUERYWINDOWPARAMS (in Frame Controls)

This message occurs when an application queries the frame control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Default Processing

The frame control window procedure queries the appropriate window parameters in accordance with *pwndparams* and sets *fresult* to TRUE if the operation is successful, otherwise to FALSE.

The window text of a frame control is obtained by sending this message to its FID_TITLEBAR.

WM_SETBORDERSIZE

This message is sent to the frame window to change the width and height of the border.

Parameters

param1

uscx *(USHORT)*
Width of border.

param2

uscy *(USHORT)*
Height of border.

Returns

reply

fresult *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The frame control sets the width and height to *uscx* and *uscy* respectively.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_SETICON

This message is sent to a frame window to set its associated icon.

Parameters

param1

hptrIcon *(HPOINTER)*
New icon handle.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fresult *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Default Processing

The icon for the frame is set.

WM_SETWINDOWPARAMS (in Frame Controls)

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

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-60.

Default Processing

The frame control window procedure sets the appropriate window parameters in accordance with *pwndparams* and sets *fresult* to TRUE if the operation is successful, otherwise to FALSE.

The window text of a frame control is set by sending this message to its FID_TITLEBAR.

WM_SIZE (in Frame Controls)

For the cause of this message, see "WM_SIZE" on page 12-61.

Parameters

For a description of the parameters, see "WM_SIZE" on page 12-61.

Default Processing

The frame control window procedure responds to this message by sending a WM_FORMATFRAME (in Frame Controls) message to itself and by setting *flreply* to 0.

WM_SYSCOMMAND

This message occurs when a control window has a significant event to notify to its owner, or when a key stroke has been translated by an accelerator table into a WM_SYSCOMMAND.

Parameters

param1

uscmd *(USHORT)*
Command value.

The frame control takes the action described on these *uscmd* values:

SC_SIZE	Sends a WM_TRACKFRAME (in Frame Controls) to the frame window.
SC_MOVE	Sends a WM_TRACKFRAME (in Frame Controls) to the frame window.
SC_MINIMIZE	If a control with the identifier FID_MINMAX is present, minimizes the frame window, or restores it to a remembered size and position.
SC_MAXIMIZE	If a control with the identifier FID_MINMAX is present, maximizes the frame window, or restores it to a remembered size and position. When a window is moved or sized in the normal way at least one border should remain on the screen. When a window is maximized and the maximum size is as large as the screen, all borders should be positioned just outside the screen.
SC_RESTORE	If a control with the identifier FID_MINMAX is present, restores a maximized frame window to its previous size and position.
SC_NEXT	Cycles the active window status to the next main window.
SC_APPMENU	Sends a MM_STARTMENUMODE message to the control with the identifier FID_MENU.
SC_SYSMENU	Sends a MM_STARTMENUMODE message to the control with the identifier FID_SYSMENU.
SC_CLOSE	If Close is not enabled in the system menu, this message is ignored. Otherwise the frame posts a WM_CLOSE message to the client if it exists or to itself, if not.
SC_NEXTFRAME	The next frame window that is a child of the desktop window is activated.
SC_NEXTWINDOW	The next window with the same owner window is activated.
SC_TASKMANAGER	The Task List is activated.
SC_HELPEXTENDED	The frame manager sends HM_EXT_HELP to the associated Help Manager Object Window. If there is no such associated window, the original message is sent to the client.
SC_HELPKEYS	The frame manager sends HM_KEYS_HELP to the associated Help Manager Object Window. If there is no such associated window, the original message is sent to the client.
SC_HELPINDEX	The frame manager sends HM_HELP_INDEX to the associated Help Manager Object Window. If there is no such associated window, the original message is sent to the client.
SC_HIDE	Sets the visibility state of the frame window to off causing it to appear hidden or invisible.

param2

ussource *(USHORT)*
Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control. *uscmd* is the window identifier of the pushbutton.
CMDSRC_MENU	Posted by a menu control. *uscmd* is the identifier of the menu item.
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. *uscmd* is the accelerator command value.
CMDSRC_OTHER	Other source. *uscmd* gives further control-specific information defined for each control type.

fpointer *(BOOL)*
Pointing-device indicator:

TRUE The message is posted as a result of a pointing-device operation.
FALSE The message is posted as a result of a keyboard operation.

Returns

ulreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message is posted to the window procedure of the owner of the frame control. *ulreply* is set to 0.

Default Processing

The default window procedure takes no action on this message, other than to set *ulreply* to 0.

WM_TRACKFRAME (in Frame Controls)

This message is sent to a frame window whenever it is to be moved or sized.

Parameters

param1

fsTrackFlags *(USHORT)*
Tracking flags.

Contains a combination of one or more TF_* flags; for details, see the TRACKINFO data structure.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fresult *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred, or the operation is terminated.

Remarks

The frame control window procedure responds to this message by causing a tracking rectangle to be drawn to move or size the window. For information, see the WinTrackRect function.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to TRUE.

WM_TRANSLATEACCEL (in Frame Controls)

For the cause of this message, see "WM_TRANSLATEACCEL" on page 12-67.

Parameters

For a description of the parameters, see "WM_TRANSLATEACCEL" on page 12-67.

Remarks

The frame control window procedure processes the message by checking whether the character is in the accelerator table, by using the WinTranslateAccel function.

Default Processing

The default window procedure takes no action on this message, other than to set *fTranslated* to FALSE.

WM_TRANSLATEMNEMONIC (in Frame Controls)

For the cause of this message, see "WM_TRANSLATEMNEMONIC" on page 12-67.

Parameters

For a description of the parameters, see "WM_TRANSLATEMNEMONIC" on page 12-67.

Remarks

The frame control window procedure processes the message by sending it to the application menu window, that is, the window with the identity FID_MENU.

Default Processing

For the default window procedure processing of this message, see "WM_TRANSLATEMNEMONIC" on page 12-67.

WM_UPDATEFRAME (in Frame Controls)

For the cause of this message, see "WM_UPDATEFRAME" on page 12-68.

Parameters

For a description of the parameters, see "WM_UPDATEFRAME" on page 12-68.

Remarks

This message must be sent to the frame window whenever an application adds or removes one of the frame controls identified by the FCF_* flags. It must also be sent if the application adds or removes a submenu of the menu bar of the frame window.

The frame control window procedure first sends the message on to the FID_CLIENT window. The FID_CLIENT window might either reformat the frame window and set *fresult* to TRUE, in which case the frame control window procedure takes no further action, or it might set *fresult* to FALSE, in which case the frame control window procedure performs the reformatting.

If *flCreateFlags* contains FCF_SIZEBORDER, reformatting the frame window includes invalidating the area occupied by the size border.

The frame control window procedure sets *fresult* to TRUE.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to TRUE.

Chapter 16. List Box Control Window Processing

This system-provided window procedure processes the actions on a list box control (WC_LISTBOX).

Purpose

A list box control is a window containing a list of items. Each item in a list box contains a text string (0 or more characters) and a handle. The text string is displayed in the list box window. The handle can be used by the application to refer to other data associated with each item.

List Box Control Styles

These list box control styles are available:

LS_HORZSCROLL The list box control enables the operator to scroll the list box horizontally.

LS_MULTIPLESEL The list box control enables the operator to select more than one item at any one time. Lists that do not have this style allow only a single selection at any one time. If this style is specified, LS_EXTENDEDSEL should also be specified.

LS_EXTENDEDSEL If this style is specified, the extended selection user interface is enabled.

LS_OWNERDRAW The list box control has one or more items that can be drawn by the owner. Typically, these items are represented by bit maps rather than by text strings.

LS_NOADJUSTPOS If this style is included, the list box control is drawn at the size specified. This can cause parts of an item to be shown.

List Box Control Data

None.

Default Colors

The following system colors are used when the system draws button controls:

SYSCLR_FIELDBACKRGOUND
SYSCLR_BUTTONDARK
SYSCLR_WINDOW
SYSCLR_WINDOWTEXT
SYSCLR_ENTRYFIELD
SYSCLR_HILITEFOREGROUND
SYSCLR_HILITEBACKGROUND
SYSCLR_WINDOWFRAME

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_DISABLEDFOREGROUNDCOLOR
PP_FOREGROUNDCOLOR
PP_HILITEFOREGROUNDCOLOR
PP_BORDERCOLOR

List Box Control Notification Messages

These messages are initiated by the list box control window to notify its owner of significant events.

WM_CONTROL (in List Boxes)

For the cause of this message, see "WM_CONTROL" on page 12-28.

Parameters

param1

idId *(USHORT)*
Control-window identity.

usnotifycode *(USHORT)*
Notify code.

The list box control window procedure uses these notification codes:

LN_ENTER	Either the Enter or Return key has been pressed while the list box control has the focus, or the list box control has been double-clicked.
LN_KILLFOCUS	The list box control loses the focus.
LN_SCROLL	The list box control is about to scroll horizontally. This can happen when the application has issued a WinScrollWindow function.
LN_SETFOCUS	The list box control receives the focus.
LN_SELECT	An item is being selected (or deselected).

Note: To discover the index of the selected item, the application must use the LM_QUERYSELECTION message.

param2

hwndcontrolspec *(HWND)*
List box control window handle.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The list box control window procedure generates this message and sends it to its owner, informing the owner of this event.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_DRAWITEM (in List Boxes)

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

Parameters

param1

idListBox *(USHORT)*
Window identifier.

The window identity of the list box control sending this notification message.

param2

pOwnerItem *(POWNERITEM)*
Owner-item structure.

This points to an owner-item structure; see OWNERITEM on page A-76.

Returns

reply

fDrawn *(BOOL)*
Item-drawn indicator:

TRUE	The owner draws the item, so the list box 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 list box control draws the item.

Remarks

The list box control window procedure only draws items that are represented by text strings and emphasizes selected items by inverting them.

If an application uses list box controls containing items that are not represented by text strings, or requires that the emphasized state of an item is to be drawn in a special manner, the list box control must specify the style LS_OWNERDRAW and those items must be drawn by the owner.

The list box control window procedure generates this message and sends it to the owner of the list box control, informing the owner that an item is to be drawn, offering the owner the opportunity to draw that item, and indicating that either the item has been drawn, or that the list box control is to draw it.

The item text must not be changed during the processing of this message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fDrawn* to the default value of FALSE.

WM_MEASUREITEM (in List Boxes)

This notification is sent to the owner of a list box control to establish the height and width for an item in that control.

Parameters

param1

sListBox *(SHORT)*
List-box identifier.

param2

sItemIndex *(SHORT)*
Item index.

The zero-based index of the item which has changed.

Returns

reply

sHeight *(SHORT)*
Height of item.

sWidth *(SHORT)*
Width of item.

This value is required only if the list box control is scrollable horizontally, that is, it has a style of LS_HORZSCROLL.

Remarks

This message is sent to the owner of a list box that has a style of LS_OWNERDRAW, to offer the owner an opportunity to establish the height and width (for a horizontally scrollable list box control) of an item that accommodates any special requirements for the drawing of items in that list box. It is sent when items in the list box are inserted or deleted, and also when presentation parameters for the list box change.

All items in a list box must have the same height, which must be greater than or equal to the height of the current font.

In particular, this notification is sent to the owner of a list box that has a style of LS_OWNERDRAW, to offer the owner an opportunity to establish the height and width (for a horizontally scrollable list box control) of an item that accommodates any special requirements for the drawing of items in that list box.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sHeight* to the default value of 0.

List Box Control Window Messages

This section describes the list box control window procedure actions on receiving the following messages.

LM_DELETEALL

This message is sent to a list box control to delete all the items in the list box.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The list box control window procedure responds to this message by deleting all the items in the list box and by setting *fSuccess* to TRUE.

Default Processing

The default window procedure does not expect to receive this message and, therefore, takes no action on it, other than to set *fSuccess* to the default value of FALSE.

LM_DELETEITEM

This message deletes an item from the list box control.

Parameters

param1

sItemIndex *(SHORT)*
Item index.

The zero-based index of the item to be deleted.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sItemsLeft *(SHORT)*
Number remaining.

The number of items in the list after the item is deleted.

Remarks

The list box control window procedure responds to this message by deleting the indexed item of the list box and by setting *sItemsLeft* to the count of the items in the list after the item is deleted.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemsLeft* to the default value of 0.

LM_INSERTITEM

This message inserts an item into a list box control.

Parameters

param1

sItemIndex *(SHORT)*
Item index:

LIT_END	Add the item to the end of the list.
LIT_SORTASCENDING	Insert the item into the list sorted in ascending order.
LIT_SORTDESCENDING	Insert the item into the list sorted in descending order.
Other	Insert the item into the list at the offset specified by this zero-based index.

param2

pItemText *(PSTRL)*
Item text.

This points to the item text.

Returns

reply

sIndexInserted *(SHORT)*
Index of inserted item:

LIT_MEMERROR	The list box control cannot allocate space to insert the list item in the list.
LIT_ERROR	An error, other than LIT_MEMERROR, occurred.
Other	The zero-based index of the offset of the item within the list.

Remarks

The list box control window procedure responds to this message by inserting the item text identified by the *pItemText* parameter into the position in the list specified by the *sItemIndex* parameter.

The sorting sequence used is that defined by the WinCompareStrings function.

The list box control sets *sIndexInserted* to the zero-based index of the offset of the item within the list.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sIndexInserted* to the default value of 0.

LM_QUERYITEMCOUNT

This message returns a count of the number of items in the list box control.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sItemCount *(SHORT)*
Item count.

Remarks

The list box control window procedure responds to this message by setting *sItemCount* to the number of items in the list.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemCount* to the default value of 0.

LM_QUERYITEMHANDLE

This message returns the handle of the indexed item of the list box control.

Parameters

param1

sItemIndex *(SHORT)*
Item index.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

ulResult *(ULONG)*
Item handle.

Remarks

The meaning of the item handle is defined by the application. It may, for example, be a pointer to an application defined data structure.

Item handles are initialized to NULLHANDLE when an item is created. The list box control window procedure responds to this message by setting *ulResult* to the handle of the item whose index is specified by *sItemIndex*.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *ulresult* to the default value of NULLHANDLE.

The item handle is initialized to NULLHANDLE.

LM_QUERYITEMTEXT

This message returns the text of the specified list box item.

Parameters

param1

sItemIndex *(SHORT)*
Item index.

smaxcount *(SHORT)*
Maximum count:

0	No text is copied.
Other	Copy the item text as a null-terminated string, but limit the number of characters copied, including the null termination character, to this value.

param2

pItemText *(PSTRL)*
Buffer into which the item text is to be copied.

This points to a PSZ.

Returns

reply

sTextLength *(SHORT)*
Length of item text.

The length of the text string, excluding the null termination character.

Remarks

The list box control window procedure responds to this message by copying up to *smaxcount* characters, as a null-terminated string, from the text of the item specified by *sItemIndex* into the buffer identified by *pItemText*.

The length of the item text can be determined by using the LM_QUERYITEMTEXTLENGTH message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *reply* to the default value of 0.

LM_QUERYITEMTEXTLENGTH

This message returns the length of the text of the specified list box item.

Parameters

param1

sItemIndex *(SHORT)*
Item index.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sTextLength *(SHORT)*
Length of item text.

The length of the text string, excluding the null termination character.

LIT_ERROR	Error occurred. For example, the item specified by its index does not exist.
Other	Length of item text.

Remarks

The list box control window procedure responds to this message by setting *sTextLength* to the length in characters of the text of the item specified by *sItemIndex*.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than set *sTextLength* to the default value of 0.

LM_QUERYSELECTION

This message is used to enumerate the selected item, or items, in a list box.

Parameters

param1

sItemStart *(SHORT)*
Index of the start item.

If the list box allows multiple selected items, that is, it has a style of LS_MULTIPLESEL, then this parameter indicates the index of the item from which the search for the next selected item is to begin. Therefore, to get all the selected items of the list, this message is sent repeatedly, each time setting this parameter to the index of the item returned by the previous usage of this message.

If the list box only allows a single selection, this parameter is ignored.

LIT_FIRST	Start the search at the first item.
Other	Start the search after the item specified by this index.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sItemSelected *(SHORT)*

Index of the selected item:

LIT_NONE	No selected item. For a single selection list box, this implies that there is no selected item in the list box. For a multiple selection list box, this implies that there is no selected item in the list box whose index is higher than the index specified by the *sItemStart* parameter.
Other	Index of selected item. For a single selection list box, this is the index of the only selected item in the list box. For a multiple selection list box, this is the index of the next selected item in the list box whose index is higher than the index specified by the *sItemStart* parameter.

Remarks

The list box control window procedure responds to this message by returning in *sItemSelected* the zero-based index of the selected item or next selected item after *sItemStart*, if any.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than set *sItemSelected* to the default value of 0.

LM_QUERYTOPINDEX

This message obtains the index of the item currently at the top of the list box.

Parameters

param1 *(ULONG)*

Reserved.

0 Reserved value, 0.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

reply

sItemTop *(SHORT)*

Index of the item currently at the top of the list box:

LIT_NONE	No items in the list box
Other	Index of the item currently at the top of the list box.

Remarks

The list box control window procedure responds to this message by returning in *sItemTop* the zero-based index of the item currently at the top of the list box.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemTop* to the default value of 0.

LM_SEARCHSTRING

This message returns the index of the list box item whose text matches the string.

Parameters

param1

uscmd *(USHORT)*
Command.

Defines the criteria by which the string specified by the *pSearchString* parameter is to be compared with the text of the items, to determine the index of the first matching item.

These values can be combined using the logical-OR operator:

LSS_CASESENSITIVE	Matching occurs if the item contains the characters specified by the *pSearchString* parameter exactly. *This value is mandatory.*
LSS_PREFIX	Matching occurs if the leading characters of the item contain the characters specified by the *pSearchString* parameter. If this value is specified, LSS_SUBSTRING must not be specified.
LSS_SUBSTRING	Matching occurs if the item contains a substring of the characters specified by the *pSearchString* parameter. If this value is specified, LSS_PREFIX must not be specified.

sItemStart *(SHORT)*
Index of the start item:

LIT_FIRST	Start the search at the first item.
Other	Start the search after the item specified by this index.

param2

pSearchString *(PSTRL)*
Search string.

This points to a PSZ.

Returns

reply

sItemMatched *(SHORT)*
Index item whose text matches the string:

LIT_ERROR	Error occurred
LIT_NONE	No item found
Other	Index item whose text matches the string.

Remarks

The list box control window procedure responds to this message by setting *sItemMatched* to the index of the next item whose text matches the string specified by *pSearchString*.

All the items of the list are searched until a match is found, that is, the search wraps from the end to the start of the list.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemMatched* to the default value of 0.

LM_SELECTITEM

This message is used to set the selection state of an item in a list box.

Parameters

param1

sItemIndex *(SHORT)*
Index of the item to be selected or deselected:

LIT_NONE	All items are to be deselected
Other	Index of the item to be selected or deselected.

param2

usSelect *(USHORT)*
Select flag:

(Ignored if *sItemIndex* is set to LIT_NONE).

TRUE	The item is selected. If the control is a single selection list box (that is, it does not have the style of LS_MULTIPLESEL), any previously selected item is deselected.
FALSE	The item is deselected.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred. For example, when the item does not exist in the list box, or when an item that is not selected is deselected.

Remarks

The list box control window procedure responds to this message by setting the selection state, as indicated by *usSelect*, of the item whose index is specified in *sItemIndex*.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

LM_SETITEMHANDLE

This message sets the handle of the specified list box item.

Parameters

param1

sItemIndex *(SHORT)*
Item index.

param2

ulItemHandle *(ULONG)*
Item handle.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The meaning of the item handle is defined by the application. It may, for example, be a pointer to an application defined data structure.

Item handles are initialized to NULLHANDLE when an item is created.

The list box control window procedure responds to this message by setting the handle of the item whose index is specified by *sItemIndex* to the value specified by *ulItemHandle*.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fsuccess* to the default value of FALSE.

LM_SETITEMHEIGHT

This message sets the height of the items in a list box.

Parameters

param1

flNewHeight *(ULONG)*
Height of items in list box.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fsuccess *(BOOL)*
Success indicator:

TRUE	Successful operation
FALSE	Error occurred.

Remarks

The list box control window procedure responds to this message by setting the height of the items in a list box to that specified by *flNewHeight*.

This message does *not* send a WM_MEASUREITEM message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fsuccess* to the default value of FALSE.

LM_SETITEMTEXT

This message sets the text into the specified list box item.

Parameters

param1

sItemIndex *(SHORT)*
Item index.

param2

pItemText *(PSTRL)*
Item text.

This points to a PSZ.

Returns

reply

fsuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The list box control window procedure responds to this message by copying the text identified by the *pItemText* parameter into the item in the list specified by the *sItemIndex* parameter.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fsuccess* to the default value of FALSE.

LM_SETTOPINDEX

This message is used to scroll a particular item to the top of the list box.

Parameters

param1

sItemIndex *(SHORT)*
Index of the item to be made top.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fsuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The list box control window procedure responds to this message by scrolling the item whose index is identified by *sItemIndex* to the top of the list box.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fsuccess* to the default value of FALSE.

WM_CHAR (in List Boxes)

For the cause of this message, see "WM_CHAR" on page 12-24.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-24.

Remarks

The list box control window procedure responds to this message by sending it to its owner if it has not processed the key stroke. This is the most common means by which the input focus is switched around the various controls in a dialog box.

The key strokes processed by a list box control are:

Down Arrow Moves the selection down one item, scrolling the list box by one item, if necessary, to make the next item visible. When the selection reaches the bottom, the Down Arrow has no effect.

Up Arrow Moves the selection up one item, scrolling the list box by one item, if necessary, to make the previous item visible. When the selection reaches the top, the Up Arrow has no effect.

Page Down Moves the selection down one page, scrolling the list box by the number of items visible in the list box.

For example, if the list box displays seven items and item 1 is selected and positioned at the top of the list box, pressing the Page Down key causes item 8 to be selected and displayed at the top of the list box. Pressing Page Down when the last item is selected has no effect.

Page Up Moves the selection up one page, scrolling the list box by the number of items visible in the list box.

For example, if the list box displays seven items and item 8 is selected and positioned at the top of the list box, pressing the Page Up key causes item 1 to be selected and displayed at the top of the list box. Pressing the Page Up key when the first item is selected has no effect.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_QUERYCONVERTPOS (in List Boxes)

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-51.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-51.

Remarks

The list box control window procedure returns QCP_NOCONVERT.

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-51.

WM_QUERYWINDOWPARAMS (in List Boxes)

Occurs when an application queries the list box control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Remarks

The list box control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure sets the *ulText*, *ulPresParams*, and *ulCtlData* parameters of the WNDPARAMS data structure, identified by *pwndparams*, to 0 and sets *fresult* to FALSE.

WM_SETWINDOWPARAMS (in List Boxes)

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

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-60.

Remarks

The list box control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 17. Menu Control Window Processing

This system-provided window procedure processes the actions on a menu control (WC_MENU).

Purpose

A menu control is a child or pull-down window that contains a list of selection items. These items can be represented by text strings, separators, bit maps or menu buttons. Menu templates can be loaded as resources and the menu can be created automatically when the parent window is created. The application can build the menu dynamically by sending MM_INSERTITEM messages. An application can change a menu by sending messages to it.

Menus enable the operator to select one of the items in the list, using the pointing device or the keyboard. When a selection is made, the menu parent is notified by posting a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message and a unique identifier representing the operator's selection.

Menus automatically resize themselves when items are added and removed. Menus are automatically destroyed when their owner is destroyed.

Typically, an application has an action bar menu and several submenus. The action bar is normally visible, and is a child window in the parent window frame. The submenus are normally hidden and become visible when selections are made on the action bar.

Menu Control Styles

These menu control styles are available :

MS_ACTIONBAR

The items in the list are displayed side-by-side. This style is used to implement a top level menu. Menus that do not have this style are displayed in one or more columns and are submenus associated with an action bar.

All menu controls have styles CS_SYNCPAINT and CS_PARENTCLIP.

MS_CONDITIONALCASCADE

This style is used to specify that the items in this list are a conditional cascade menu. Conditional cascade menus act like normal cascade menus with the exception that the cascade does not automatically open when the user selects it. To open the conditional cascade menu, the mini-pushbutton on the menu item must be selected. If the menu is selected without opening the cascade, the default item in the cascade is selected. The default action on the cascade is identified by a check mark.

MS_TITLEBUTTON

Used to identify menus that can be used as buttons in the title bar. Can only be used with MS_ACTIONBAR.

This style causes the menu to be drawn using the CUA colors specified for the title bar rather than the action bar.

MS_VERTICALFLIP

Normally, pull-down menus (the default, without the MS_VERTICALFLIP style) are displayed below their associated action bar item. If there is not room on the screen to display the entire pull-down in this manner, and if there is room to display the pull-down above the action bar, it is displayed above the action bar. Pull-down menus with the MS_VERTICALFLIP style are flipped vertically. That is, they are displayed above the menu if possible, otherwise below it. The vertical flip style must be set explicitly by the application when the window is minimized, and must be reset when it is restored.

If an application action bar contains this style, the style is applied to all pull-down menus belonging to the action bar (the style does

not directly affect the display of the action bar). This provides a convenient means for the application to flip the appearance of all pull-down menus.

Menu Item Styles

These menu item styles are available:

MIS_SUBMENU The item is a submenu. When the user selects this type of item, a submenu is displayed from which the user must make further selection. Items that are not submenu items are command items.

MIS_SEPARATOR The display object is a horizontal dividing line. This type of item can only be used in pull-down menus. This type of item cannot be enabled, checked, disabled, highlighted, or selected by the user. The functional object is NULL when this style is specified.

MIS_BITMAP The display object is a bit map.

MIS_TEXT The display object is a text string.

MIS_BUTTONSEPARATOR The item is a menu button. Any menu can have zero, one, or two items of this type. These are the last items in a menu and are automatically displayed after a separator bar. The user cannot move the cursor to these items, but can select them with the pointing device or with the appropriate key.

MIS_BREAK The item begins a new row or column.

MIS_BREAKSEPARATOR Same as MIS_BREAK, except that it draws a separator between rows or columns of a pull-down menu. This style can only be used within a submenu.

MIS_SYSCOMMAND If this item is selected, the menu notifies the owner by posting a WM_SYSCOMMAND message rather than a WM_COMMAND message.

MIS_OWNERDRAW Items with this style are drawn by the owner. WM_DRAWITEM and WM_MEASUREITEM notification messages are sent to the owner to draw the item or determine its size.

MIS_HELP If the item is selected, the menu notifies the owner by posting a WM_HELP message rather than a WM_COMMAND message.

MIS_STATIC This type of item exists for information purposes only. It cannot be selected with the pointing device or keyboard.

Menu Item Attributes

These menu item attributes are available:

Applications can get and set the state of these attributes by sending MM_QUERYITEMATTR and MM_SETITEMATTR messages.

MIA_HILITED The state of this attribute is TRUE, if and only if, the item is selected.

MIA_CHECKED If this attribute is TRUE a check mark appears next to the item.

MIA_DISABLED This attribute is TRUE if the item is disabled and cannot be selected. The item is drawn in a disabled state.

MIA_FRAMED If this attribute is TRUE a frame is drawn around the item.

MIA_NODISMISS If this item is selected, the pull-down menu containing this item should not be hidden before notifying the application window of the selection. A menu with this attribute is not hidden until such time as the application or user explicitly does so, for example by selecting either another menu on the action bar or by pressing the escape key.

Default Colors

The following system colors are used when the system draws button controls:

SYSCLR_WINDOWFRAME
SYSCLR_BUTTONDARK
SYSCLR_BUTTONLIGHT
SYSCLR_SHADOW
SYSCLR_TITLEBOTTOM
SYSCLR_DIALOGBACKGROUND

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_FOREGROUNDCOLOR
PP_HILITEFOREGROUNDCOLOR
PP_BORDERCOLOR
PP_DISABLEDFOREGROUNDCOLOR

Menu Control Notification Messages

These messages are initiated by the menu control window procedure to notify its owner of significant events.

WM_COMMAND (in Menu Controls)

For the cause of this message, see "WM_COMMAND" on page 12-27.

Parameters

For a description of the parameters, see "WM_COMMAND" on page 12-27.

The menu control window procedure sets *uscmd* to the menu-item identity.

Remarks

The menu control window procedure generates this message if the WM_MENUSELECT (in Menu Controls) message returns a *fresult* of TRUE. when an item is selected that does not have the style of MIS_SYSCOMMAND or MIS_HELP. The menu control window procedure posts the message to the queue of the window owner.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_DRAWITEM (in Menu Controls)

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

Parameters

param1

idMenu *(USHORT)*
Window identifier.

The window identity of the menu control sending this notification message.

param2

pOwnerItem *(POWNERITEM)*
Owner-item structure.

This points to an owner-item structure; see OWNERITEM on page A-76.

Returns

reply

fDrawn *(BOOL)*
Item-drawn indicator:

TRUE The owner draws the item, and so the menu 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 menu control draws the item.

Remarks

The menu control window procedure only draws items that are represented by text strings and emphasizes selected items by inverting them.

If an application uses menu controls containing items that are not represented by text strings, or requires that the emphasized state of an item is to be drawn in a special manner, then the menu control must specify the style MIS_OWNERDRAW and those items must be drawn by the owner.

The menu control window procedure generates this message and sends it to its owner, informing the owner that an item is to be drawn, offering the owner the opportunity to draw that item, and to indicate that either the item has been drawn, or that the menu control is to draw it.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fDrawn* to the default value of FALSE.

WM_HELP (in Menu Controls)

For the cause of this message, see "WM_HELP" on page 12-36.

Parameters

For a description of the parameters, see "WM_HELP" on page 12-36.

The menu control window procedure sets *uscmd* to the menu-item identity.

Remarks

This message is identical to a WM_COMMAND message, but implies that the application should respond to this message by displaying help information.

The menu control window procedure generates this message and posts it to the queue of its owner when an item is selected that has the style of MIS_HELP, but only if WM_MENUSELECT (in Menu Controls) returns a *fresult* of TRUE.

Default Processing

The default window procedure sends this message to the parent window, if it exists and is not the desktop. Otherwise, it sets *flreply* to 0.

WM_INITMENU (in Menu Controls)

For the cause of this message, see "WM_INITMENU" on page 12-39.

Parameters

For a description of the parameters, see "WM_INITMENU" on page 12-39.

Remarks

This message offers the owner the opportunity to perform some initialization on the menu items before they are presented.

The menu control window procedure generates this message and sends it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_MEASUREITEM (in Menu Controls)

This notification is sent to the owner of a menu control to establish the height for an item in that control.

Parameters

param1

sMenu *(SHORT)*
Menu identifier.

param2

pOwnerItem *(POWNERITEM)*
Owner-item structure.

This points to an OWNERITEM structure.

Returns

reply

sHeight *(SHORT)*
Height of item.

Remarks

This message is only sent at the time the menu control is created. When the owner receives this message, it must calculate and return the height of an item to the control.

All items in a menu must have the same height, and that must be greater than or equal to the height of the current font.

In particular, this notification is sent to the owner of a menu that has a style of MIS_OWNERDRAW, to offer the owner an opportunity to establish the height of an item that accommodates any special requirements for the drawing of items in that menu.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sHeight* to the default value of 0.

WM_MENUEND (in Menu Controls)

For the cause of this message, see "WM_MENUEND" on page 12-41.

Parameters

For a description of the parameters, see "WM_MENUEND" on page 12-41.

Remarks

The menu control window procedure generates this message and sends it to its owner, informing the owner of this event.

Default Processing

The default window procedure takes no action on this message, other than to set *fIreply* to 0.

WM_MENUSELECT (in Menu Controls)

For the cause of this message, see "WM_MENUSELECT" on page 12-42.

Parameters

For a description of the parameters, see "WM_MENUSELECT" on page 12-42.

Remarks

The menu control window procedure generates this message and sends it to its owner, informing the owner of this event.

When the message is returned from its owner, menu control acts on *fresult* as appropriate.

It must not be posted to the menu control.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to TRUE.

WM_NEXTMENU (in Menu Controls)

For the cause of this message, see "WM_NEXTMENU" on page 12-44.

Parameters

For a description of the parameters, see "WM_NEXTMENU" on page 12-44.

Remarks

The menu control generates this message and sends it to its owner, informing the owner of this event.

Default Processing

The default window procedure takes no action on this message, other than to set *hwndNewMenu* to NULLHANDLE.

WM_SYSCOMMAND

For the cause of this message, see "WM_SYSCOMMAND" on page 12-63.

Parameters

For a description of the parameters, see "WM_SYSCOMMAND" on page 12-63.

The menu control window procedure sets *uscmd* to the menu-item identity.

Remarks

The menu control window procedure generates this message and posts it to the queue of its owner, when an item is selected that has the style of MIS_SYSCOMMAND, but only if the WM_MENUSELECT (in Menu Controls) message returns a *fresult* of TRUE.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

Menu Control Window Messages

This section describes the menu control window procedure actions on receiving the following messages.

MM_DELETEITEM

This message deletes a menu item.

Parameters

param1

usItem *(USHORT)*
Item identifier.

usIncludesubmenus *(USHORT)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and delete it.

FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sItemsLeft *(SHORT)*
Number remaining.

The number of items in the menu after the item is deleted.

Remarks

The menu control window procedure responds to this message by deleting the identified item from the menu or its submenus.

Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemsLeft* to the default value of 0, which is equivalent to 0.

MM_ENDMENUMODE

This message is sent to a menu control to terminate menu selection.

Parameters

param1

usdismiss *(USHORT)*
Dismiss menu indicator:

TRUE	Dismiss the submenu or subdialog window
FALSE	Do not dismiss the submenu or subdialog window.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The menu control window procedure responds to this message by terminating menu selection.

Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and, therefore, takes no action on it, other than to set *fireply* to the default value of 0.

MM_INSERTITEM

This message inserts a menu item into a menu.

Parameters

param1

pmenuItem *(PMENUITEM)*
Menu-item data structure.

This points to a MENUITEM structure.

param2

pItemText *(PSTRL)*
Item text.

Returns

reply

sIndexInserted *(SHORT)*
Index of inserted item:

MIT_MEMERROR	The menu control cannot allocate space to insert the menu item in the menu.
MIT_ERROR	An error other than MIT_MEMERROR occurred.
Other	The zero-based index of the offset of the item within the menu.

Remarks

The menu control window procedure responds to this message by inserting the identified item into the menu at the position indicated by the specified MENUITEM data structure (contained within the menu-item structure). If the position is MIT_END, the item is added to the end of the menu. If the style of the item includes MIS_TEXT, the text of the item is specified by *pItemText*

The menu control window procedure sets *sIndexInserted* to the zero-based index of the position of the item within the menu.

Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sIndexInserted* to the default value of 0.

MM_ISITEMVALID

This message returns the selectable status of a specified menu item.

Parameters

param1

usItem *(USHORT)*
Item identifier.

usIncludeSubmenus *(USHORT)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier.
FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fResult *(BOOL)*
Selectable indication.

A menu item can be selected and entered under these conditions:

- The item is enabled and, if it is a submenu item, the item in the action bar associated with the submenu is enabled. If the action bar item is not enabled, the user cannot display the submenu.
- The item is enabled, and the submenu is displayed and being tracked with the pointing device or keyboard. It is unlikely, but possible, that the associated action bar is disabled in this instance.

TRUE The user can select and enter the specified item.
FALSE The user cannot select and enter the specified item.

Remarks

The menu control window procedure responds to this message by setting the return value depending on the selectable status of the specified item.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fresult* to the default value of FALSE.

MM_ITEMIDFROMPOSITION

This message returns the identity of a menu item of a specified index.

Parameters

param1

sItemIndex *(SHORT)*
Item index.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sIdentity *(SHORT)*
Item identity:

MIT_ERROR	Error occurred; for example, because *sItemIndex* is not valid.
Other	Item identity.

Remarks

The menu control window procedure responds to this message by setting *reply* to the identity of the item whose position is identified by the index specified in *sItemIndex*.

Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *reply* to the default value of 0.

MM_ITEMPOSITIONFROMID

This message returns the index of a menu item of a particular identity.

Parameters

param1

usItem *(USHORT)*
Item identifier.

usIncludesubmenus *(USHORT)*
Include submenus indicator:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sIndex *(SHORT)*
Item index:

MIT_NONE	Item does not exist
Other	Item index.

Remarks

The menu control window procedure responds to this message by setting *sIndex* to the zero-based index of the item identified by *usItem*.

Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sIndex* to the default value of MIT_NONE.

MM_QUERYITEM

This message returns the definition of the specified menu item.

Parameters

param1

usItem *(USHORT)*
Item identifier.

usIncludesubmenus *(USHORT)*
Include submenus flag:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and copy its definition.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

pmenuItem *(PMENUITEM)*
Menu-item data structure.

This points to a MENUITEM structure.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The menu control window procedure responds to this message by copying the item definition specified by *usItem*, from the menu, to the structure specified by *pmenuItem*.

Note: This message does not retrieve the text for items with a style of MIS_TEXT. The item text is obtained by use of the MM_QUERYITEMTEXT message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

MM_QUERYITEMATTR

This message returns the attributes of a menu item.

Parameters

param1

usItem *(USHORT)*
Item identity.

usIncludeSubmenus *(USHORT)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and return its state.

FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

usAttributeMask *(USHORT)*
Attribute mask.

Returns

reply

usState *(USHORT)*
State.

Remarks

The menu control responds to this message by returning the state of the specified attributes of the identified menu item.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *usState* to the default value of 0.

MM_QUERYITEMCOUNT

This message returns the number of items in the menu.

Parameters

param1 *(ULONG)*

0 Reserved value, 0.

param2 *(ULONG)*

0 Reserved value, 0.

Returns

reply

sResult *(SHORT)*
Item count.

Remarks

The menu control window procedure responds to this message by returning the count of the number of items in the menu.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *usState* to the default value of 0.

MM_QUERYITEMRECT

This message returns the bounding rectangle of a menu item.

Parameters

param1

usItem *(USHORT)*
Item identity.

fIncludeSubmenus *(BOOL)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and return its state.
FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

prect *(PRECTL)*
Bounding rectangle of the menu item in device coordinates relative to the menu window.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Specified item was found.
FALSE Specified item was not found.

Remarks

The menu control responds to this message by returning the bounding rectangle of identified menu item.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *usState* to the default value of 0.

MM_QUERYITEMTEXT

This message returns the text of the specified menu item.

Parameters

param1

usItem *(USHORT)*
Item identifier.

smaxcount *(SHORT)*
Maximum count:

Copy the item text as a null-terminated string, but limit the number of characters copied, including the null termination character, to this value, which must be greater than 0.

param2

pItemText *(PSTRL)*
Buffer into which the item text is to be copied.

This points to a PSZ.

Returns

reply

sTextLength *(SHORT)*
Length of item text.

The length of the text string, excluding the null termination character.

0	Error occurred. For example, no item of the specified identity exists or the item has no text. No text is copied.
Other	Length of item text.

Remarks

The menu control window procedure responds to this message by copying up to *smaxcount* characters as a null-terminated string from the text of the item specified by *usItem*, if it has the style MIS_TEXT, into the buffer specified by *pItemText*.

The length of the item text can be determined by using the MM_QUERYITEMTEXTLENGTH message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sTextLength* to the default value of 0.

MM_QUERYITEMTEXTLENGTH

This message returns the text length of the specified menu item.

Parameters

param1

usItem *(USHORT)*
Item identifier.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sLength *(SHORT)*
Length of item text.

The length of the text string, excluding the null termination character.

0	Error occurred. For example, no item of the specified identity exists or the item has no text. No text is copied.
Other	Length of item text.

Remarks

The menu control window procedure responds to this message by returning the length in characters of the text of the identified item, if it has a style of MIS_TEXT.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sLength* to the default value of 0.

MM_QUERYSELITEMID

This message returns the identity of the selected menu item.

Parameters

param1

fsReserved *(USHORT)*
Reserved.

0

usIncludesubmenus *(USHORT)*
Include submenus indicator:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for a selected item with the specified identifier.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for a selected item with the specified identifier.

param2 *(ULONG)*
Reserved.

0	Reserved value, 0.

Returns

reply

sresult *(SHORT)*
Selected item identifier:

MID_ERROR	Error occurred
MIT_NONE	No item selected
Other	Selected item identifier.

Remarks

The menu control window procedure responds to this message by returning the identity of the selected item in the menu. Submenus and subdialogs are not searched unless *usincludesubmenus* is set to TRUE.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sresult* to the default value of 0.

MM_REMOVEITEM

This message removes a menu item.

Parameters

param1

usitem *(USHORT)*
Item identifier.

usincludesubmenus *(USHORT)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and delete it.

FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sItemsLeft *(SHORT)*
Count of remaining items.

Remarks

The menu control window procedure responds to this message by removing the identified item from the menu and setting *sItemsLeft* to the count of items in the menu after the item is deleted.

The difference between this message and MM_DELETEITEM is that MM_DELETEITEM destroys any submenu window, and deletes any bit map associated with the item, whereas MM_REMOVEITEM does not.

Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemsLeft* to the default value of 0.

MM_SELECTITEM

This message selects or deselects a menu item.

Parameters

param1

sItem *(SHORT)*
Item identifier:

MIT_NONE	Deselect all the items in the menu
Other	Item identifier.

usIncludesubmenus *(USHORT)*
Include submenus indicator:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and select or deselect it.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

fsreserved *(USHORT)*
Reserved.

0 Reserved value, 0.

usDismissed *(USHORT)*
Dismissed flag:

TRUE	Dismiss the menu
FALSE	Do not dismiss the menu.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	A selection has been made, or *sItem* is MIT_NONE.
FALSE	A selection has not been made, or a deselection has been made, or *sItem* is not MIT_NONE.

Remarks

The menu control window procedure responds to this message by setting the selection state of the (sub)menu which contains the specified item to indicate that the item is selected or deselected. If *usincludesubmenus* is set to TRUE, the selection state of the (sub)menu owning the submenu which contains the specified item is also set. This process continues up the menu hierarchy until the top level menu is reached.

If an item is selected, and *usdismissed* is set to TRUE, a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message, as appropriate, is posted to the owner, and the menu is dismissed.

Note: This message must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

MM_SETITEM

This message sets the definition of a menu item.

Parameters

param1

fsreserved *(USHORT)*
Reserved.

0 Reserved value, 0.

usincludesubmenus *(USHORT)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and set its definition.

FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

pmenuitem *(PMENUITEM)*
Menu-item data structure.

This points to a MENUITEM structure.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The menu control window procedure responds to this message by using the specified structure to update the definition of the identified menu item.

The *iPosition* field of the structure specified by *pmenuitem* is ignored, as the position of the item cannot be changed by use of this message.

Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

MM_SETITEMATTR

This message sets the attributes of a menu item.

Parameters

param1

usItem *(USHORT)*
Item identifier.

usIncludeSubmenus *(USHORT)*
Include submenus indicator:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and set its attributes.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

usAttributeMask *(USHORT)*
Attribute mask.

usAttributeData *(USHORT)*
Attribute data. Set this to attribute mask. Not TRUE/FALSE

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The menu control window procedure responds to this message by setting the state of the specified attributes for the identified item.

Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

MM_SETITEMHANDLE

This message sets the handle of a menu item.

Parameters

param1

usItem *(USHORT)*
Item index.

param2

ulItemHandle *(ULONG)*
Item handle.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The menu control window procedure responds to this message by setting the handle of the indexed menu item.

This is used to set a handle for menu items that have a style of MIS_BITMAP or MIS_OWNERDRAW.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

MM_SETITEMTEXT

This message sets the text of a menu item.

Parameters

param1

usItem *(USHORT)*
Item identifier.

param2

pItemText *(PSTRL)*
Item text.

This points to a PSZ.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The menu control responds to this message by setting the text of the identified item, if it has a style of MIS_TEXT, using the specified null-terminated string.

Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

MM_STARTMENUMODE

This message is used to begin menu selection.

Parameters

param1

usshowsubmenu *(USHORT)*
Show submenu flag:

TRUE Show the submenu (pull-down menu) of the selected action bar item when the menu enters selection mode. If the action bar is not visible, the submenu is shown, otherwise it is not shown. If the item selected does not have a submenu, this parameter is ignored.

FALSE Do not show the submenu (pull-down menu) of the selected action bar item when the menu enters selection mode.

usresumemenu *(USHORT)*
Resume menu mode flag:

TRUE Resume the user interaction with the menu from where it left off. The menu is assumed to have been used previously and left without dismissing one of the submenus, and therefore is resumed in that submenu.

FALSE Begin user interaction with the menu from the action bar, subject to the value of the *usshowsubmenu* parameter.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

It is posted to the menu when the operator presses the menu key.

Note: It must be posted, not sent, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

WM_QUERYCONVERTPOS (in Menu Controls)

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-51.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-51.

Remarks

The menu control window procedure returns QCP_NOCONVERT.,

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-51.

WM_QUERYWINDOWPARAMS (in Menu Controls)

Occurs when an application queries the menu control window procedure parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Remarks

The menu control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure sets the *ulText*, *ulPresParams*, and *ulCtlData* parameters of the WNDPARAMS data structure, identified by *pwndparams*, to 0 and sets *fresult* to FALSE.

WM_SETWINDOWPARAMS (in Menu Controls)

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

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-60.

Remarks

The menu control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

Chapter 18. Multi-Line Entry Field Control Window Processing

This system-provided window procedure processes the actions on a multi-line entry field control (WC_MLE).

Purpose

A multi-line entry field control is a rectangular window that displays multiple lines of text that the operator can edit. When it has the focus, the cursor marks the current **insertion** or **replacement** point.

How to Use

The text is displayed within a rectangular window. Scroll bars appear if requested.

On all four sides of the text within the window there exists a thin margin area. This margin remains drawn in the window's background color, and characters are never drawn into this margin. Mouse events that occur in the margin are processed differently from mouse events that occur in the text area. The margin should be large enough to be easily clicked on, but not so large as to take up a large quantity of screen space. It is suggested, but not required, that the left and right margins be half the average character width of the system font, and that the top and bottom margins be half the maximum baseline extent of the system font.

Text is defined as a stream of characters, with hard line-break characters in the text. Between any two bytes in the text stream, and at either end of the document, there is an insertion point. Note that in a DBCS environment, it is possible to have an insertion point in the middle of a DBCS character. If such an insertion point is specified in a function, the function will either round the insertion point in a sensible way, or the function will fail with an error code indicating the problem.

The text always contains a selection region, defined by an anchor point and a cursor point. The anchor and cursor points are insertion points. If the MLE window has the focus, the text between these two points is drawn highlighted and the cursor point is indicated by a flashing text cursor. The selection region can be affected by some import/export operations.

The cursor point and the anchor point define the range of the selection. These two points are often the same, in which case no text is selected and only a text cursor (but no highlighting) is displayed. A user can use SHIFT+cursor movement combinations to extend the selection, which leaves the anchor point alone, and moves the cursor point to a new position in the document.

The MLE has three modes:

READ-ONLY The keyboard user interface disallows any operations that would change the content of the text, although applications using the MLE can still change the text contents. The application can query this mode, in order that it can disallow application-specific operations.

WORD-WRAP When this mode is in effect, soft line-breaks are inserted into the text at word boundaries so that the user need not scroll the display horizontally to see all the text. When this mode is off, text is allowed to trail off the right-hand edge of the window.

INSERT/OVERTYPE This mode determines whether keystrokes are inserted into the text, or whether they overtype existing text. Unlike the other two modes, this mode is maintained by the system. The MLE must merely be aware of the system mode.

Notes:

1. The MLE is intended for text under 4KB in size. Performance will be fast for text up to 32KB in size. Text greater than this will be supported but performance may not be acceptable.
2. In this chapter 'CR' denotes carriage-return, and 'LF' denotes line-feed.

Multi-Line Entry Field Control Styles

These multi-line entry field control styles are available:

MLS_BORDER	A thin border is drawn around the multi-line entry field window.
MLS_READONLY	The multi-line entry field is initially in read-only mode.
MLS_WORDWRAP	The multi-line entry field initially word-wraps text.
MLS_HSCROLL	The multi-line entry field displays and handles a horizontal scroll bar.
MLS_VSCROLL	The multi-line entry field displays and handles a vertical scroll bar.
MLS_IGNORETAB	The multi-line entry field ignores tab key strokes. It passes the appropriate WM_CHAR to its owner window.
MLS_DISABLEUNDO	The multi-line entry field will not allow undo actions.

Multi-Line Entry Field Control Data

See MLECTLDATA on page A-69.

Multi-Line Entry Field Control Notification Messages

This message is initiated by the multi-line entry field window procedure to notify its owner of significant events.

WM_CONTROL (in Multiline Entry Fields)

For the cause of this message, see "WM_CONTROL" on page 12-28.

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

An *fAction* 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 pels that would not fit within the current text limit.

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

An *fAction* 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 pels that would not fit within the current text limit.

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

An *fAction* 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 *fAction* 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 *fAction* 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

This parameter depends on the MLN_* notification code.

For a *usnotifycode* of MLN_TEXTOVERFLOW:

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

plxOver *(PIX)*
Linear distance of overflow in pels.

pErrInfo *(POVERFLOW)*
Overflow error information structure.

The *ulErrInd* 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.

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

pmrg *(PMARGSTRUCT)*
Margin structure.

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.

flReserved *(ULONG)*
Reserved.

0

Returns

reply

For a *usnotifycode* of MLN_TEXTOVERFLOW, MLN_PIXHORZOVERFLOW, MLN_PIXVERTOVERFLOW, MLN_MARGIN, MLN_SEARCHPAUSE:

fAction *(BOOL)*
Action taken by application:

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.

flReserved *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

The multiline entry field control window procedure generates this message and sends it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

Multi-Line Entry Field Window Messages

This section describes the multi-line entry field control window procedure actions on receiving the following messages.

MLM_CLEAR

This message clears the current selection.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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

Remarks

The multi-line entry field control window procedure responds to this message by clearing the current selection and returning the number of bytes cleared.

Default Processing

The default window procedure takes no action on this message, other than to set *ulClear* to 0.

MLM_COPY

This message copies the current selection to the clipboard.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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

Remarks

The multi-line entry field control window procedure responds to this message by copying the selected text to the clipboard. The text is translated to standard clipboard format, which is the same as exporting with MLE_CFTEXT format.

The text is placed on the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64KB).

This may cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set *ulCopy* to 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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

Remarks

The multi-line entry field control window procedure responds to this message by copying the selected text to the clipboard and then deleting it. The text is translated to standard clipboard format, which is the same as exporting with MLE_CFTEXT format.

The text is placed on the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64KB).

This may cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set *ulCopy* to 0.

MLM_CHARFROMLINE

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

Parameters

param1

lLineNum *(LONG)*
Line number of interest.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

iptFirst *(IPT)*
First insertion point on line.

Remarks

For any line number, the insertion point just before the first character on that line is returned. If the line number is −1, the line containing the cursor is used.

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

Default Processing

The default window procedure takes no action on this message, other than to set *iptFirst* to 0.

MLM_DELETE

This message deletes text.

Parameters

param1

iptBegin *(IPT)*
Starting point of deletion.

param2

ulDel *(ULONG)*
Number of bytes to delete.

Returns

reply

ulSuccess *(ULONG)*
Number of bytes successfully deleted.

Remarks

This message takes an insertion point and a length, and deletes that number of characters from the text. If the insertion point is −1, the selection is used and the effect is identical to the MLM_CLEAR message.

This may cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set *ulSuccess* to 0.

MLM_DISABLEREFRESH

This message disables screen refresh.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion.
FALSE An error occurred.

Remarks

This message disables screen refreshes. This allows an application to make changes throughout a document while avoiding unnecessary overhead caused by attempts to keep the screen display current. When an MLM_ENABLEREFRESH message is sent, the screen display is brought up to date with the contents of the text.

While refresh is disabled, mouse and keyboard messages are processed by beeping and ignoring them, except for mouse moves, which do not beep; the mouse pointer changes to the system standard wait symbol (a clock face).

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

MLM_ENABLEREFRESH

This message enables screen refresh.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion.
FALSE An error occurred.

Remarks

This message enables screen refreshes. This allows an application to make changes throughout a document while avoiding unnecessary overhead caused by attempts to keep the screen display current. When an MLM_ENABLEREFRESH message is sent, the screen display is brought up to date with the contents of the text.

While refresh is disabled, mouse and keyboard messages are processed by beeping and ignoring them, except for mouse moves, which do not beep; the mouse pointer changes to the system standard wait symbol (a clock face).

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

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

reply

ulSuccess *(ULONG)*
Number of bytes successfully exported.

Remarks

This message takes an insertion point and length as parameters, and copies text, starting from that insertion point, into the buffer set by MLM_SETIMPORTEXPORT. Text is in the format set by MLM_FORMAT. If the insertion point is –1, the selection is used for both *pBegin* and *pCopy*.

On return, *pBegin* is updated to follow the last byte exported, and the number of bytes to be exported is decremented by the number actually exported. This is done to prepare those parameter values for the next export. The return value indicates the number of bytes actually put into the buffer. This number is less than, or equal to, the buffer size (see MLM_SETIMPORTEXPORT).

Note: All exports are done in full characters. Therefore, if either the length of the buffer or the number of bytes to be exported result in the last byte transferred being only half of a DBCS character, the MLE will *not* transfer that byte.

It returns the number of bytes placed in the export buffer.

Default Processing

The default window procedure takes no action on this message, other than to set *ulSuccess* to 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

usFormat *(USHORT)*
Previous format value.

Remarks

The default format is MLFIE_CFTEXT.

The keyword MLFIE_RTF is reserved.

Default Processing

The default window procedure takes no action on this message, other than to set *usFormat* to 0.

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

reply

ulSuccess *(ULONG)*
Number of bytes successfully inserted.

Remarks

This message takes an insertion point and length as parameters. It assumes a buffer has been set using MLM_SETIMPORTEXPORT, and inserts the contents of the buffer at the insertion point in the text. The contents are interpreted as being in the format set by MLM_FORMAT. If the insertion point is −1, the cursor point is used.

The insertion point *pBegin* is updated by the MLE to the point after the last character imported. This provides the application with the location for the next import.

The return value indicates how many bytes were actually transferred.

All imports are done in full characters, therefore, if the number of bytes to be imported results in the last byte transferred being only half of a DBCS character, or part of a line-break sequence (CR LF or CR CR LF), the MLE does not transfer that byte. If the return value indicates that less than the full amount was transferred, a check must be made to determine if it is the beginning of a multi-byte sequence, and if so, the parts must be mated and imported as a whole.

This can cause an overflow, see MLN_OVERFLOW.

Note: The buffer is not zero-terminated; NULL characters can be inserted into the text.

Default Processing

The default window procedure takes no action on this message, other than to set *ulSuccess* to 0.

MLM_INSERT

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

Parameters

param1

pText *(PSTRL)*
Null-terminated text string.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

ulCount *(ULONG)*
Number of bytes actually inserted.

Remarks

This message inserts the text string at the current selection, deleting that selection in the same manner as typing at the keyboard would. The text string must be in CF_TEXT format (or one of the formats acceptable to MLM_IMPORT) and null-terminated. The line-break (CR LF, LF, and so on) is counted as one byte, regardless of the number of bytes occupied in the buffer, and the null terminator is not counted.

This interacts with the format rectangle and text limits, and a return of less than the full count can be the result. If so, a notification message is sent.

Default Processing

The default window procedure takes no action on this message, other than to set *ulCount* to 0.

MLM_LINEFROMCHAR

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

Parameters

param1

iptFirst *(IPT)*
Insertion point of interest

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

lLineNum *(LONG)*
Line number of insertion point.

Remarks

For any insertion point, the corresponding line number is returned. If the insertion point is −1, the number of the line containing the first insertion point of the selection is returned.

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

Default Processing

The default window procedure takes no action on this message, other than to set *lLineNum* to 0.

MLM_PASTE

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

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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

Remarks

The multi-line entry field control window procedure responds to this message by replacing the selected text with text from the clipboard. The text is translated from standard clipboard format, which is the same as importing with MLE_CFTEXT format.

The text is assumed to be in the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64Kb).

This can cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set *ulCopy* to 0.

MLM_QUERYBACKCOLOR

This message queries the background color.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

lColor *(LONG)*
Text color.

Remarks

This message returns the color in which the background is to be drawn.

The color values are the same as those used by GpiSetColor.

Default Processing

The default window procedure takes no action on this message, other than to set *lColor* to 0.

MLM_QUERYCHANGED

This message queries the changed flag.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fChanged *(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.

Remarks

The multi-line entry field control window procedure responds to this message by returning the changed flag for the text without altering it. See also MLN_CHANGE.

Default Processing

The default window procedure takes no action on this message, other than to set *fChanged* to 0.

MLM_QUERYFIRSTCHAR

This message queries the first visible character.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

iptFVC *(IPT)*
First visible character.

Remarks

Returns the insertion point immediately preceding the character visible in the upper left-hand corner of the screen. If a partial character is displayed, that character counts as the first visible character.

Note: In situations where no character is visible, because the text is scrolled to the right beyond the end of the top line, this returns the insertion point of the last character on the line (EOL not considered). In situations where there are no characters on the line, the insertion point at the beginning is returned.

Default Processing

The default window procedure takes no action on this message, other than to set *iptFVC* to 0.

MLM_QUERYFONT

This message queries which font is in use.

Parameters

param1

pFattrs *(PFATTRS)*
Font attribute structure.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSystem *(BOOL)*
System font indicator:

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

Remarks

This message puts the attributes of the current drawing font into the font attribute structure.

Default Processing

The default window procedure takes no action on this message, other than to set *fSystem* to FALSE.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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

Remarks

For any insertion point, the number of bytes between that insertion point and the end of the line is returned, after the current formatting is applied. If the insertion point is −1, the cursor position is used. This message differs from MLM_QUERYLINELENGTH in that the byte count returned reflects the effects of the current formatting set by MLM_FORMAT.

Default Processing

The default window procedure takes no action on this message, other than to set *iptLine* to 0.

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

reply

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

Remarks

This message returns the length in bytes of a range of characters after the current formatting is applied. This differs from MLM_QUERYTEXTLENGTH in that:

- A range of insertion points can be queried.
- The byte count returned reflects the effects of the current formatting set by MLM_FORMAT.

Default Processing

The default window procedure takes no action on this message, other than to set *ulText* to 0.

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

flreply *(ULONG)*
Reserved

Default Processing

The default window procedure takes no action on this message, other than to set *freply* to 0.

MLM_QUERYIMPORTEXPORT

This message queries the current transfer buffer.

Parameters

param1

pBuff *(PBUFFER)*
Transfer buffer.

param2

pBuff *(PULONG)*
Size of transfer buffer in bytes.

Returns

reply

ulCount *(ULONG)*
Success indicator:

Remarks

This message returns the values from the most recent MLM_SETIMPORTEXPORT, or 0 for either value if it has not been set.

Default Processing

The default window procedure takes no action on this message, other than to set *ulCount* to 0.

MLM_QUERYLINECOUNT

This message queries the number of lines of text.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

ulLines *(ULONG)*
The number of lines of text.

Remarks

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

The multi-line edit control always maintains one CR LF line-break in the buffer, therefore the number of lines returned may be one greater than the number actually visible.

Default Processing

The default window procedure takes no action on this message, other than to set *ulLine* to 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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

Remarks

For any insertion point, the number of bytes between that insertion point and the end of the line is returned. If the insertion point is −1, the cursor position is used. If the line contains a hard line-break, it is counted as one byte.

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

Default Processing

The default window procedure takes no action on this message, other than to set *iptLine* to 0.

MLM_QUERYREADONLY

This message queries the read-only mode.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fReadOnly *(BOOL)*
Current read-only status.

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

Default Processing

The default window procedure takes no action on this message, other than to set *fReadOnly* to FALSE.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

For *usQueryMode* = MLFQS_MINMAXSEL:

sMinSel *(SHORT)*
Minimum insertion point of selection.

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

sMaxSel *(SHORT)*
Maximum insertion point of selection.

This value is rounded down to 65 535 if necessary.

For *usQueryMode* = MLFQS_MINSEL, MLFQS_MAXSEL, MLFQS_ANCHORSEL, or MLFQS_CURSORSEL:

iptIpt *(IPT)*
Requested insertion point.

Remarks

This message returns the location of the selection in several different forms. The insertion points lie between characters, and start at a zero origin before the first character in the MLE. Subtracting the minimum from the maximum gives the number of characters in the selection. *This is not necessarily the number of bytes of ASCII.* The line-break character is a CR LF (2 bytes) and all DBCS characters are 2 bytes. To determine the number of bytes, use MLM_QUERYFORMATTEXTLENGTH, being sure that the format choice set by MLM_FORMAT is set to what is used when the data is exported from the MLE (for example, MLE_CFTEXT for MLM_QUERYSELTEXT).

Note the following:

- If anchor point > cursor point, minimum point = cursor point and maximum point = anchor point.
- If anchor point < cursor point, minimum point = anchor point and maximum point = cursor point.

Default Processing

The default window procedure takes no action on this message, other than to set *reply* to 0.

MLM_QUERYSELTEXT

This message copies the currently selected text into a buffer.

Parameters

param1

pBuff *(PSTRL)*
Buffer for text string.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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

Remarks

This message copies the currently selected text into the buffer pointed to by *pBuff*. The text string is null-terminated. The byte count includes the text in CF_TEXT format (CR LF) and the null terminator.

Default Processing

The default window procedure takes no action on this message, other than to set *ulCount* to 0.

MLM_QUERYTABSTOP

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

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

pixTabset *(PIX)*
Tab width in pels.

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

Remarks

This message fails and returns a negative value, if the reserved values are not 0.

Default Processing

The default window procedure takes no action on this message, other than to set *pixTabset* to 0.

MLM_QUERYTEXTCOLOR

This message queries the text color.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

lColor *(LONG)*
Text color.

Remarks

This message returns the color in which text is to be drawn.

The color values are the same as those used by GpiSetColor.

Default Processing

The default window procedure takes no action on this message, other than to set *lColor* to 0.

MLM_QUERYTEXTLENGTH

This message returns the number of characters in the text.

Parameters

param1 *(ULONG)*

Reserved.

0 Reserved value, 0.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

reply

iptText *(IPT)*

Count of text in bytes.

Remarks

This message returns the number of characters in the text. Hard line-breaks are counted as 1 and soft line-breaks as 0.

This message differs from the WinQueryWindowTextLength call in that it returns a *LONG*.

Default Processing

The default window procedure takes no action on this message, other than to set *iptText* to 0.

MLM_QUERYTEXTLIMIT

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

Parameters

param1 *(ULONG)*

Reserved.

0 Reserved value, 0.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

reply

lSize *(LONG)*

Maximum number of bytes allowed in the MLE.

Remarks

The multi-line entry field control window procedure responds to this message by returning the current limit set, either by default, or by MLM_SETTEXTLIMIT. If the limit is unbounded, a non-positive value is returned.

Default Processing

The default window procedure takes no action on this message, other than to set *lSize* to 0.

MLM_QUERYUNDO

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

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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.

fUndoRedo *(BOOL)*
Undo or redo indicator.

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

Default Processing

The default window procedure takes no action on this message, other than to set *reply* to 0.

MLM_QUERYWRAP

This message queries the wrap flag.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fWrap *(BOOL)*
Wrap flag.

TRUE Word-wrap enabled
FALSE Word-wrap disabled.

Default Processing

The default window procedure takes no action on this message, other than to set *fWrap* to FALSE.

MLM_RESETUNDO

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

Parameters

param1 *(ULONG)*

Reserved.

0 Reserved value, 0.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

reply

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.

fUndoRedo *(BOOL)*

Undo or redo indicator.

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

Remarks

This message resets the undo state of the MLE to indicate that the last operation cannot be undone (null return from MLM_QUERYUNDO). This can be used by the application when it performs an operation that it can undo, that supersedes the last MLE operation. The application can then reset its own undo state upon receipt of an MLN_CHANGE, indicating that later changes have occurred through the MLE.

Default Processing

The default window procedure takes no action on this message, other than to set *reply* to 0.

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

reply

fSuccess *(BOOL)*
Success indicator:

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

Remarks

This message searches the MLE text for a specified string, starting at a specified insertion point and continuing until the second specified insertion point has been reached, or the requested string has been matched.

When an MLM_SEARCH message is sent, the text is scanned starting with the character that follows the insertion point indicated in the *iptStart* field of the MLE_SEARCHDATA structure. The search proceeds until the point indicated in the *iptStop* field, until a match is found, or until TRUE is returned from MLN_SEARCHPAUSE notification (see WM_CONTROL (in Multiline Entry Fields)). If a negative value is specified for the *iptStart*, the current cursor point is used. If a negative value is specified for *iptStop*, the end of the text is used. If *iptStop*, is less than or equal to *iptStart*, after performing the two indicated substitutions, the search wraps from the end of the text to the beginning of the text.

If the MLFSEARCH_CASESENSITIVE option is specified, the bytes of the search string must exactly match those in the text. If MLFSEARCH_CASESENSITIVE is not specified, the WinUpperChar of the search string must match the WinUpperChar of the text.

When a match is found, the *iptStart* field of the search specification structure is set to indicate the insertion point immediately preceding the first character of the match, and the *cchFind* field is set to indicate the number of characters in the match. The cursor selection is not altered unless MLFSEARCH_SELECTMATCH is specified. If it is, an MLM_SETSEL is done with the anchor point at *iptStart* and the cursor at *iptStart* + *cchFind*.

While searching, the MLE occasionally sends an MLN_SEARCHPAUSE notification message. If the owner responds to this message with the value TRUE, the MLE stops the search. When a search is stopped from MLN_SEARCHPAUSE, *iptStart* is set to the point where the search terminated. If the response is FALSE, the search continues (see also the definition of MLN_SEARCHPAUSE). The interval at which MLN_SEARCHPAUSE notifications are sent is implementation-dependent, but must not exceed reasonable user-response thresholds, nor should it be so often as to introduce undue messaging overhead. Sending this notification every half second is a reasonable compromise.

When no match is found the *iptStart* value is unchanged.

If the application needs to continue the search, the proper way is to change the *iptStart* value to be the point following the string found, adjusting for any text changes done after the search that may have moved the relative location of the point.

Applications using this message are advised to change the system pointer to the wait icon (clock face) if it is expected that the search will take some time.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

MLM_SETBACKCOLOR

This message sets the background color.

Parameters

param1

lColor *(LONG)*
Color.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

lOldColor *(LONG)*
Color previously used.

Remarks

This message sets the color in which the MLE background is to be drawn, and updates the display as necessary.

The color values are the same as those used by GpiSetColor.

Default Processing

The default window procedure takes no action on this message, other than to set *lOldColor* to 0.

MLM_SETCHANGED

This message sets or clears the changed flag.

Parameters

param1

usChangedNew *(USHORT)*
Value to set changed flag to.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fChanged *(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.

Remarks

This message can generate a MLN_CHANGE notification.

Default Processing

The default window procedure takes no action on this message, other than to set *fChanged* to FALSE.

MLM_SETFIRSTCHAR

This message sets the first visible character.

Parameters

param1

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

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE An error occurred.

Remarks

This message scrolls the text to place the character following the insertion point into the upper left-hand corner of the window. If the insertion point specified is beyond the end of a line, or the end of the file, it is resolved in the same way as it is for a mouse click.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

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 *(ULONG)*
Reserved.

0	Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

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

Remarks

For any *PFATTRS*, this message sets the display to use the appropriate font. If NULL, the system font is used. The screen is updated appropriately.

This can cause an overflow, see MLN_OVERFLOW.

When setting an outline font it is necessary to ensure that the FATTRS structure contains the correct maximum baseline extent and average character width for the desired point size and that the font use is marked as FATTR_FONTUSE_TRANSFORMABLE.

Baseline extent and character width are calculated by multiplying the desired point size by the current display device font resolution (CAPS_VERTICAL_FONT_RES and CAPS_HORIZONTAL_FONT_RES; see DevQueryCaps) and dividing by 72, the number of points in an inch.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

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 *LONG*s designating the diagonally-opposite corner of the rectangle, assuming 0,0 for the first. Therefore, they are the width and height in pels 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, *fSuccess* 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

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE An error occurred.

Remarks

The multi-line entry field control window procedure responds to this message by setting formatting dimensions and mode.

Any addition of text that causes the text to exceed the rectangle limits causes a notification before proceeding (see MLN_PIXHORZOVERFLOW and MLN_PIXVERTOVERFLOW).

Any activity that would cause the rectangle to be unable to contain the existing text (resize, undo, increasing font size, or word-wrap on or off) is rejected and results in a notification message for information (see MLN_OVERFLOW).

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

MLM_SETIMPORTEXPORT

This message sets the current transfer buffer.

Parameters

param1

pBuff *(PBUFFER)*
Transfer buffer.

param2

ulLength *(ULONG)*
Size of transfer buffer in bytes.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	An error occurred.

Remarks

Given a far pointer to a buffer, and the size of the buffer, this message sets it as the current transfer buffer for the MLE. This buffer is used by the MLM_IMPORT and MLM_EXPORT messages. The system segment limit must be observed when specifying the buffer size.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

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

reply

fSuccess *(BOOL)*
Success indicator:

Remarks

This message sets the anchor and cursor points. The screen display is updated appropriately, ensuring that the cursor point is visible (which may involve scrolling). Note that the text cursor and inversion are not displayed if the MLE window does not have the input focus. A negative value for a point leaves that point alone.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

MLM_SETREADONLY

This message sets or clears read-only mode.

Parameters

param1

usReadOnly *(USHORT)*
New read-only value.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fOld *(BOOL)*
Previous read-only value.

Remarks

When read-only mode is set, characters typed at the keyboard do not get inserted into the MLE text. The API insertion interface, however, is still functional, as are selection-manipulation activities and copy-to-clipboard operations. This is useful as a means of preventing text modification (such as in a help system), and for providing a minimal blocking printing semaphore.

Default Processing

The default window procedure takes no action on this message, other than to set *fOld* to FALSE.

MLM_SETTEXTCOLOR

This message sets the text color.

Parameters

param1

lColor *(LONG)*
Color.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

lOldColor *(LONG)*
Color previously used.

Remarks

This message sets the color in which the MLE text is to be drawn, and updates the display as necessary.

The color values are the same as those used by GpiSetColor.

Default Processing

The default window procedure takes no action on this message, other than to set *lOldColor* to 0.

MLM_SETTABSTOP

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

Parameters

param1

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

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

pixTabset *(PIX)*
Success indicator:

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

Remarks

This message fails if the reserved value is not 0.

This message can cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set *pixTabset* to 0.

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 MLE NO_TRANS format.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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.

Remarks

The multi-line entry field control window procedure responds to this message by limiting the text size to *lSize* bytes. Text size is calculated using the MLFIE_NOTRANS format. Note that this is bytes and *not* characters; DBCS programmers should calculate accordingly.

This message returns 0 if the text limit exceeds or is equal to the existing text. Otherwise it returns the number of bytes by which the text would have overflowed, and does not change the limit.

The default, which is unbounded, can be specified by entering a non-positive limit.

Default Processing

The default window procedure takes no action on this message, other than to set *ulFit* to 0.

MLM_SETWRAP

This message sets the wrap flag.

Parameters

param1

usWrap *(USHORT)*
New value for wrap flag.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE An error occurred.

Remarks

The multi-line entry field control window procedure responds to this message by setting the word wrap mode and updating the screen as appropriate.

When word-wrap is turned on, the text is wrapped to fit the formatting rectangle width. When word-wrap is turned off, the text is allowed to trail off to the right until it reaches an end-of-line marker.

Word-wrapping is defined as follows. Words are sequences of non-white-space characters (white-space characters are space, line break, and tab). When word-wrapping is enabled, the whole word must appear on one line within the formatting rectangle, unless the word by itself is too long to fit. In this case the word is split following the last character that fits, and the remainder starts a new line.

This definition then applies recursively to the remainder of the word. The word continues to be visible. For editing purposes (for example, for word-selection) the word is viewed as a single word drawn over multiple lines.

Blank characters are always accumulated onto the current line, even if they exceed the horizontal formatting dimension, that is, blanks are allowed to trail off the right-hand edge. Line-break characters are also allowed to exceed the horizontal dimension, and any subsequent text must begin on a new line. The line-break following a line-break character is sometimes referred to as a hard line-break. Other line breaks, due to word-wrapping, and not to explicit formatting characters, are referred to as soft line-breaks.

Tab characters must always be visible. If a tab character occurs after the last tab stop within the horizontal formatting dimension, a soft line-break occurs after the tab.

This message can cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

MLM_UNDO

This message performs any available undo operation.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

usUndone *(USHORT)*
Success indicator:

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

Remarks

The last operation is undone (note that an undo can be undone.)

This can cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set *usUndone* to FALSE.

WM_BUTTON1DBLCLK (in Multiline Entry Fields)

For the cause of this message, see "WM_BUTTON1DBLCLK" on page 12-10.

Parameters

For a description of the parameters, see "WM_BUTTON1DBLCLK" on page 12-10.

Remarks

This message indicates that mouse button 1 has clicked twice within the system double-click time.

Double-Click
If the click point is in the middle of a non-white-space character, the token (word) surrounding the clicked-on character, and any trailing spaces, are selected. If the click point is in a space character, the previous word (along with the trailing spaces including the clicked-on space) is selected. If there is no preceding word (either because the spaces are at the beginning of the text or immediately follow a line-break character) the run of spaces is selected. If the click point is on a tab or line-break character, that character is selected.

Shift-Double-Click
Double-clicking while the Shift key is pressed leaves the anchor point alone, and moves the cursor point to the beginning or end of the clicked-on token. If the click point is before the anchor point in the text, the cursor point is moved to the beginning of the surrounding word, otherwise, the cursor point is moved to the end of the surrounding word. When shift-double-clicking, the selection is extended to include the token that was double-clicked on.

Margin Mouse Event
All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the owner window of the MLE. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_BUTTON1DOWN (in Multiline Entry Fields)

For the cause of this message, see "WM_BUTTON1DOWN" on page 12-13.

Parameters

For a description of the parameters, see "WM_BUTTON1DOWN" on page 12-13.

Remarks

This message delimits mouse button click events. Between a button-down and a button-up event, the mouse is considered to be dragging. A mouse click is considered to happen on button-down, and dragging is terminated by a button-up.

Click
Clicking in the text sets the cursor and anchor points to the nearest insertion point. If the MLE is in overtype mode, the anchor is extended one character further in the text, subject to the end-of-text and new-line boundary conditions, defined under WM_CHAR (in Multiline Entry Fields).

Shift-Click
Clicking while the shift key is held down sets the cursor point to the nearest insertion point, while leaving the anchor point alone.

Margin Mouse Event
All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the owner window of the MLE. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_BUTTON1UP (in Multiline Entry Fields)

For the cause of this message, see "WM_BUTTON1UP" on page 12-19.

Parameters

For a description of the parameters, see "WM_BUTTON1UP" on page 12-19.

Remarks

This message delimits mouse button click events. Between a button-down and a button-up event the mouse is considered to be dragging. A mouse click is considered to happen on button-down, and dragging is terminated by a button-up.

Margin Mouse Event
All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the owner window of the MLE. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_CHAR (in Multiline Entry Fields)

For the cause of this message, see "WM_CHAR" on page 12-24.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-24.

Remarks

The behavior of the MLE, when typing, depends on whether it is in insert or overtype mode, and whether the selection is empty or not. The selection is defined to be empty when the cursor point is equal to the anchor point.

When a character is typed, it replaces the current selection. If the selection is empty, the character is viewed as replacing nothing, so the character is effectively inserted into the text. If one or more characters are selected, those characters are deleted from the text and replaced by the typed character.

If the MLE is in insert mode, the cursor and anchor points are moved to immediately follow the newly typed character.

If the MLE is in overtype mode, the cursor is moved to immediately follow the newly typed character. If there is no character after the cursor (the new character is at the end of the text) or if the character after the cursor is a line-break character, the anchor is set to be equal to the cursor point. In any other case, the anchor is extended one character past the cursor point, defining the next character as the current selection.

If the typing causes the cursor to go off the screen in any direction, the display is automatically scrolled. If word-wrap is on, text continues on a new line, otherwise, the screen is scrolled horizontally.

Scrolling of the text in the window is independent of cursor movement. The cursor and selection remain unaltered at the same location within the text during all scrolling but the converse is not true. Any movement of the cursor causes auto-scrolling, if necessary, to ensure that the text location of the cursor is visible within the window.

Tabs: Tabs are represented as a single character in the text model, and are displayed as enough white-space to reach the next tab stop. Tab stops are set at pel intervals, starting with zero and occurring every n pels, where n is a value set by the MLM_SETTABSTOP message, and defaulting to eight times the average character width of the system font. When a tab is drawn, it uses the number of pels defined by the following formula:

```
pelWidth = pelTab - (pelDraw mod pelTab))
```

where pelTab is the tab interval, in pels, and pelDraw is the pel at which drawing is to begin.

Return: Return (ASCII newline) causes a hard line-break, and the following text begins on a new line. A line-break character is inserted in the text, which is drawn as a few pels of white-space (for selection purposes).

Keystroke commands: For all the following keys, unless otherwise noted, the display is scrolled, if necessary, to keep the cursor point visible. Where noted, the cursor setting behaves differently in insert mode than in overtype mode. This is subject to the boundary conditions noted above.

Del	Causes the contents of the selection region to be deleted. If the selection region contains no text, it causes the character to the right of the cursor to be deleted.
Shift+Del	Causes the contents of the selection region to be cut to the clipboard.
Insert	Toggles between insert and overtype mode. The MLE ignores the Insert key when it occurs without a modifier.
Shift+Ins	Causes the contents of the clipboard to replace the selection region.
Ctrl+Ins	Causes the selection region to be copied to the clipboard. The selection region is not otherwise affected.
Backspace	Functions similar to Del. If the selection is not empty, Backspace deletes the selection. If the selection is empty, Backspace deletes the character to the left of the cursor point. If the MLE is in overtype mode, the anchor point is set, and the cursor point is moved to be one character previous in the text. If no such character exists (because the anchor is set to the beginning of the text) the cursor is set to the anchor point. If the MLE is in insert mode, the cursor and anchor points are set, as defined at the start of this chapter.
Down Arrow	Sets the cursor point to the closest insertion point on the following line, then sets the anchor point to the cursor point (insertion mode) or one character following (overtype mode).
Shift+Down Arrow	Causes the cursor point to be moved to the closest insertion point on the following line. The anchor point does not move.
Up Arrow	Sets the cursor point to the closest insertion point on the preceding line, then sets the anchor point to the cursor point (insert mode) or one character following (overtype mode).
Shift+Up	Sets the cursor point to the closest insertion point on the preceding line. The anchor point is not moved.
Right Arrow	Sets the cursor point to the insertion point one character following the cursor point. The anchor point is set to the cursor point (insert mode) or one character following (overtype mode).
Shift+Right	Causes the cursor point to be set to the insertion point immediately following the previous cursor point. The anchor point is not moved.
Left and Shift+Left	Work analogously.
Ctrl+Right	Moves the cursor point to the insertion point immediately preceding the next word in the text including trailing spaces, and sets the anchor point to be equal to (insert mode) or one character following (overtype mode) the cursor point. The EOL (hard line-break) and tab characters are treated as words.
Ctrl+Shift+Right	Moves only the cursor point in the same way as Ctrl+Right, but leaves the anchor point unmoved.

Ctrl+Left Moves the cursor point to the preceding insertion point at the beginning of a word, and sets the anchor point to be equal to (insert mode) or one character following (overtype mode) the cursor point. The EOL (hard line-break) and tab characters are treated as words.

Ctrl+Shift+Left Moves only the cursor point in the same way as Ctrl+Left but leaves the anchor point unmoved.

Pagedown and Pageup
Cause the display to be scrolled one screen at a time in either direction. This behavior is the same as would be encountered during a page-down or page-up caused by the scroll-bar.

Ctrl+Pagedown and Ctrl+Pageup
Cause the display to be scrolled one screen at a time to the right or left respectively. This behavior is the same as would be encountered during a page-right or page-left caused by the scroll-bar.

Home Sets the cursor point to the insertion point at the beginning of the line containing the cursor point, and sets the anchor point equal to (insert mode) or one character following (overtype mode).

Shift+Home Moves the cursor point to the insertion point at the beginning of the line. The anchor point is not moved.

End Sets the anchor point to the insertion point at the end of the line containing the cursor point. If the last character on the line is a line-break character, the anchor is positioned just before it. The cursor is set equal to (insert mode) or one character previous to (overtype mode) the anchor.

Shift+End Moves the cursor point to the insertion point at the end of the line, as above. The anchor point is not moved.

Ctrl+Home Moves the cursor point to the insertion point at the beginning of the document. The anchor point is set equal to (insert mode) or one character following it (overtype mode).

Ctrl+End Moves the anchor point to the insertion point at the end of the document. The cursor point is set to be equal to the anchor point (insert mode) or one character preceding it (overtype mode).

Ctrl+Shift+Home Moves the cursor point in the same way as Ctrl+Home, but leaves the anchor point unmoved.

Ctrl+Shift+End Moves the cursor point in the same way as Ctrl+End, but leaves the anchor point unmoved.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_ENABLE (in Multiline Entry Fields)

For the cause of this message, see "WM_ENABLE" on page 12-31.

Parameters

For a description of the parameters, see "WM_ENABLE" on page 12-31.

Remarks

The multi-line entry field control window procedure responds to this message by setting the enable state and by setting *fIreply* to 0.

Disabling the window is similar, but not identical, to MLM_DISABLEREFRESH. Enabling the window is similar, but not identical, to MLM_ENABLEREFRESH. (Note that this also applies to window styles.) The difference is that a disabled window receives no mouse or keyboard input whereas with MLM_DISABLEREFRESH it receives the input but discards it.

Default Processing

The default window procedure takes no action on this message, other than to set *fIreply* to 0.

WM_MOUSEMOVE (in Multiline Entry Fields)

For the cause of this message, see "WM_MOUSEMOVE" on page 12-43.

Parameters

For a description of the parameters, see "WM_MOUSEMOVE" on page 12-43.

Remarks

The mouse pointer moves and is of interest to the MLE. If refresh is disabled, the pointer is set to the wait icon (a clock face). If refresh is enabled, the pointer is set to an I-beam. This message can occur during dragging or when simply tracking the mouse.

Dragging
Dragging sets the selection anchor to be the point where dragging begins, and moves the cursor point along with it as the mouse is moved. Moving the pointer into the margins while dragging produces a scroll in the appropriate direction and continues selecting.

Margin Mouse Event
All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the owner window MLE. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

Default Processing

The default window procedure takes no action on this message, other than to set *fProcessed* to 0.

WM_QUERYWINDOWPARAMS (in Multiline Entry Fields)

This message occurs when an application queries the entry field control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Remarks

The multi-line entry field control window procedure responds to this message by returning the window parameters indicated by the *ulStatus* parameter of the WNDPARAMS data structure, identified by the *pwndparams* parameter.

In response to the WPM_CCHTEXT flag, the text length is reported in the CF_TEXT format. If it exceeds 64KB—1, then this value is reported. In response to the WPM_TEXT flag, text up to the amount returned for the WPM_CCHTEXT value is placed at the indicated location in CF_TEXT format.

Default Processing

The default window procedure sets the *ulText*, *ulPresParams*, and *ulCtlData* parameters of the WNDPARAMS data structure, identified by *pwndparams*, to 0 and sets *fresult* to FALSE.

WM_SETWINDOWPARAMS (in Multiline Entry Fields)

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

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-60.

Remarks

The multi-line entry field control window procedure responds to this message by setting the window parameters indicated by the *ulStatus* parameter of the WNDPARAMS data structure, identified by the *pwndparams* parameter.

If the MLE text is to be set by this message, it is assumed to be in CF_TEXT format (see MLM_FORMAT) and all existing text is deleted before the new text is inserted. Note that a Control Data structure can be associated with the window parameters, in which case any field in that structure can cause a change to the MLE.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

Chapter 19. Prompted Entry Field Control Window Processing

This system-provided window procedure processes the actions on a prompted entry field (combo box) control (WC_COMBOBOX).

Purpose

A combo box consists of an entry field control and a list box control merged into a single control. The list, which is usually limited in size, is displayed below the entry field, and offset one dialog-box unit to its right.

When the combo box control has the focus, the text in the entry field is given selected emphasis and, if the list box control has a matching entry, it is scrolled to show that match at the top of the list.

A combo box, while sometimes only showing the entryfield, also owns the area occupied by the invisible list box. Another window can and will be clipped to it if they have clipping flags set.

Combo Box Control Styles

These combo box control styles are available:

CBS_SIMPLE Both the entry field control and the list box control are visible. When the selection changes in the list box control, the text of the selected item in the list box control is placed in the entry field. Also, the text in the entry field is completed by extending the text of the entry field with the closest match from the list box.

CBS_DROPDOWN Inherits all the properties of a combo box control with a style of CBS_SIMPLE and, in addition, the list box control is hidden until the user requests that it should be displayed.

CBS_DROPDOWNLIST In which the entry field control is replaced by a static control, that displays the current selection from the list box control. The user must explicitly cause the display of the list box control in order to make alternative selections in the list box.

Combo Box Control Data

None.

Default Colors

The following system colors are used when the system draws button controls:

SYSCLR_WINDOWFRAME
SYSCLR_ENTRYFIELD
SYSCLR_WINDOW
SYSCLR_BUTTONMIDDLE
SYSCLR_BUTTONDARK
SYSCLR_BUTTONLIGHT
SYSCLR_OUTPUTTEXT
SYSCLR_WINDOWTEXT
SYSCLR_HIGHLITEFOREGROUND
SYSCLR_HIGHLITEBACKGROUND
SYSCLR_FIELDBACKRGOUND
SYSCLR_WINDOWFRAME.

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_FOREGROUNDCOLOR
PP_DISABLEDFOREGROUNDCOLOR
PP_HIGHLIGHTFOREGROUNDCOLOR
PP_FONTNAMESIZE
PP_BORDERCOLOR.

Combo Box Control Notification Messages

The combo box control uses most of the same window messages as the entry field control and the list box control to notify its owner of significant events.

WM_CONTROL (in Combination Boxes)

For the cause of this message, see "WM_CONTROL" on page 12-28.

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) box-control window handle.

Returns

fIreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The entry field control window procedure generates this message and sends it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set *fIreply* to 0.

Combo Box Control Window Messages

The combo box control uses most of the same messages as the entry field control and the list box control. In particular, the following messages are supported to achieve the functions of a combo box. These messages are explained in detail in the entry field control window messages and the list box control window messages sections.

WM_SETWINDOWPARAMS (In Entry Fields) To set the text of the entry field.

WM_QUERYWINDOWPARAMS (In Entry Fields) To obtain the text of the entry field.

LM_QUERYITEMCOUNT	To obtain the count of items in the list box control.
LM_INSERTITEM	To insert an item into the list box control.
LM_SETTOPINDEX	To scroll the list box control so that the specified item is at the top.
LM_QUERYTOPINDEX	To obtain the index of the item at the top of the list box control.
LM_DELETEITEM	To delete an item from the list box control. If necessary, this also changes the content of the entry field to the item at the top of the list box control.
LM_SELECTITEM	To select a specified item in the list box control. Also, this changes the content of the entry field to the item at the top of the list box control and, if the list box control is not visible, causes the list box control to 'dropdown' below the entry field control.
LM_QUERYSELECTION	To obtain the current selection in the list box control.
LM_SETITEMTEXT	To change the text of an item in the list box control. If necessary, this also changes the content of the entry field control.
LM_QUERYITEMTEXT	To obtain the text of an item in the list box control.
LM_QUERYITEMTEXTLENGTH	To obtain the length of the text of an item in the list box control.
LM_SEARCHSTRING	To obtain the index of an item in the list box control containing a specified string.
LM_DELETEALL	To delete all the items in the list box control.
WM_ENABLE	To enable the combo box control to respond to input.
EM_QUERYFIRSTCHAR	To obtain the character displayed at the left edge of the entry field control.
EM_SETFIRSTCHAR	To scroll the entry field control so that the specified character is displayed at the left edge of the entry field control.
EM_QUERYCHANGED	To obtain the changes to the entry field control.
EM_QUERYSEL	To obtain the current selection of the entry field control.
EM_SETSEL	To set the current selection of the entry field control.
EM_SETTEXTLIMIT	To set the maximum number of characters to be contained in the entry field control.
EM_CUT	To place the contents of the selection of the entry field control into the clipboard and then delete those contents from the entry field control.
EM_PASTE	To place the contents of the clipboard into the entry field control.
EM_COPY	To place the contents of the selection of the entry field control into the clipboard.
EM_CLEAR	To clear the current selection of the entry field control.

This section describes the combo box control window procedure actions on receiving these messages:

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fChanged *(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.

Remarks

The combo box control window procedure responds to this message by setting the highlighting state of the entry field control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fChanged* to the default value of FALSE.

CBM_ISLISTSHOWING

This message determines if the list box control is showing.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fShowing *(BOOL)*
Showing indicator:

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

Remarks

The combo box control window procedure responds to this message by indicating if the list box control is showing.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fShowing* to the default value of FALSE.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fChanged *(BOOL)*
Changed indicator:

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

Remarks

The combo box control window procedure responds to this message by setting the showing state of the list box control.

This message has no effect on a combo box control whose style is CBS_SIMPLE.

Hiding the list box control has no effect on the selection in the list box control. The selection in the list box control must be changed by the use of a LM_SELECTITEM message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fChanged* to the default value of FALSE.

Chapter 20. Scroll Bar Control Window Processing

This system-provided window procedure processes the actions on a scroll bar control (WC_SCROLLBAR).

Purpose

Scroll bars are controls used to indicate that additional information can be displayed in a window, logically to the left or right for horizontal scroll bars, logically above or below for vertical scroll bars. The user interface for scroll bars allows for scrolling one unit or one page at a time, or alternatively picking up the scroll bar slider and moving it to a position in the scroll bar that indicates a logical position in the data.

Scroll Bar Control Styles

These scroll bar control styles are available:

SBS_HORZ	Create a horizontal scroll bar.
SBS_VERT	Create a vertical scroll bar.
SBS_THUMBSIZE	Indicates the presence of the *cVisible* and *cTotal* parameters in the SBCDATA data structure.
SBS_AUTOTRACK	The slider scrolls as more information is being displayed on the screen.
SBS_AUTOSIZE	The scroll bar slider changes size to reflect the amount of data contained in the window.

Scroll Bar Control Data

See SBCDATA on page A-114.

Default Colors

The following system colors are used when the system draws button controls:

SYSCLR_SCROLLBAR
SYSCLR_WINDOWFRAME
SYSCLR_FIELDBACKGROUND
SYSCLR_WINDOW
SYSCLR_BUTTONMIDDLE.

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_FOREGROUNDCOLOR
PP_BORDERCOLOR
PP_HILITEFOREGROUNDCOLOR.

Scroll Bar System Values

Applications can use the following system values to create and add control scroll bars:

SV_CXVSCROLL	Width of the vertical scroll-bar.
SV_CYHSCROLL	Height of the horizontal scroll-bar.
SV_CYVSCROLLARROW	Height of the vertical scroll-bar arrow bit maps.
SV_CXHSCROLLARROW	Height of the vertical scroll-bar arrow bit maps.
SV_FIRSTSCROLLRATE	The delay (in milliseconds) before autoscrolling starts, when using a scroll bar.
SV_SCROLLRATE	The delay (in milliseconds) between scroll operations, when using a scroll bar.
SYSCLR_SCROLLBAR	Color for drawing scroll-bar backgrounds.
TID_SCROLL	Timer ID for a reserved scrolling time. This is used for sending notification messages when a scroll-arrow or scroll-bar background is selected.

Scroll Bar Control Notification Messages

These messages are initiated by the scroll bar control window procedure to notify its owner of significant events.

WM_HSCROLL (in Horizontal Scroll Bars)

For the cause of this message, see "WM_HSCROLL" on page 12-38.

Parameters

For a description of the parameters, see "WM_HSCROLL" on page 12-38.

Remarks

The scroll bar control window procedure generates this message and posts it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set *reply* to 0.

WM_VSCROLL (in Vertical Scroll Bars)

For the cause of this message, see "WM_VSCROLL" on page 12-68.

Parameters

For a description of the parameters, see "WM_VSCROLL" on page 12-68.

Remarks

The scroll bar control window procedure generates this message and posts the message to the owner of the procedure, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set *fireply* to 0.

Scroll Bar Control Window Messages

This section describes the scroll bar control window procedure actions on receiving the following messages.

SBM_QUERYPOS

This message returns the current slider position in a scroll bar window.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sslider *(SHORT)*
Slider position.

Remarks

The scroll bar control window procedure responds to this message by returning the current slider position.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sslider* to the default value of 0.

SBM_QUERYRANGE

This message returns the scroll bar range minimum and maximum values.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

sfirst *(SHORT)*
First bound.

slast *(SHORT)*
Last bound.

Remarks

The scroll bar control window procedure responds to this message by returning the first and last bounds of the scroll bar range.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *reply* to the default value of *sfirst* and *slast* 0.

SBM_SETPOS

This message sets the position of the slider in a scroll bar window.

Parameters

param1

sslider *(SHORT)*
Position of slider.

If this value is outside the scroll-bar range, the slider is moved to the nearest valid position within the range.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion

Remarks

The scroll bar control window procedure responds to this message by setting the position of the slider.

The scroll bar control is redrawn to reflect the change.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it.

SBM_SETSCROLLBAR

This message sets the scroll-bar range and slider position.

Parameters

param1

sslider *(SHORT)*
Position of slider.

If this value is outside the scroll-bar range, the slider is moved to the nearest valid position within the range.

param2

sfirst *(SHORT)*
First bound.

This value must not be less than 0. If a value less than 0 is supplied, 0 is used as the value.

slast *(SHORT)*
Last bound.

The value must not be less than 0 or *sfirst*. If a value less than this is supplied, the higher of 0 or *sfirst* is used as the value.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion

Remarks

The scroll bar control window procedure responds to this message by setting the values of the information range and the position of the slider.

The scroll bar is redrawn to reflect the change.

For example, if a scroll-bar is to allow scrolling through 100 lines of text, of which 50 are visible at any one time, and the top display line is currently number 25, *sfirst* should be set to 1, *slast* to 51 (since there are only 51 positions at which the slider may be placed), and *sslider* to 25. The SBM_SETTHUMBSIZE message should be used in this example to set the slider size to 50 visible parts out of 100.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it.

SBM_SETTHUMBSIZE

This message sets the scroll bar slider size.

Parameters

param1

svisible *(SHORT)*
Size of the visible part of the document.

stotal *(SHORT)*
Size of the entire document.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion

Remarks

The scroll bar control window procedure responds to this message by setting the size of the slider proportional to the visible part of the document. If the visible part exceeds or is equal to the entire document the scroll bar is disabled, otherwise the scroll bar is enabled.

The scroll bar is redrawn to reflect the change.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it.

WM_QUERYCONVERTPOS (in Scroll Bars)

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-51.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-51.

Remarks

The scroll bar control window procedure returns QCP_NOCONVERT.,

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-51.

WM_QUERYWINDOWPARAMS (in Scroll Bars)

This message occurs when an application queries the scroll bar control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Remarks

The scroll bar control window procedure responds to this message by returning the window parameters indicated by the *ulStatus* parameter of the WNDPARAMS data structure identified by the *pwndparams* parameter.

Default Processing

The default window procedure sets the *ulText*, *ulPresParams*, and *ulCtlData* parameters of the WNDPARAMS data structure, identified by *pwndparams*, to 0 and sets *fresult* to FALSE.

WM_SETWINDOWPARAMS (in Scroll Bars)

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

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-60.

Remarks

The scroll bar control window procedure responds to this message by setting the window parameters indicated by the *ulStatus* parameter of the WNDPARAMS data structure identified by the *pwndparams* parameter.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

Chapter 21. Spin Button Control Window Processing

This system-provided window procedure processes the actions on a spin button control (WC_SPINBUTTON).

Purpose

A spin button control (WC_SPINBUTTON window class) is a visual component whose specific purpose is to give users quick access to a finite set of data. The spin button allows users to select from a scrollable ring of choices. Since users can see only one item at a time, the spin button control should be used only with data that is intuitively related, such as a list of months of the year, or an alphabetic list of cities or states.

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

You can create multifield 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 spin field could contain a list of numbers and the third spin field could contain a list of years.

Spin Button Control Styles

Create a spin button using the style bits listed below. These styles can be joined together by using logical ORs (|).

- Specify one of the following to determine whether a spin field will be a master or a servant. If neither is specified, SPBS_SERVANT is the default.

  SPBS_MASTER The spin button component consists of at least one single line entry field (SLE), or spin field, and two arrows, the Up Arrow and the Down Arrow. When a spin button contains more than one spin field, the master component contains the spin arrows. If the component contains only one spin field, it should be a master.

  SPBS_SERVANT You can create a multifield spin button by spinning servants from the master.

- Specify one of the following to determine the type of characters allowed in the spin field:

  SPBS_ALLCHARACTERS Any character can be typed in the spin field. This is the default.

  SPBS_NUMERICONLY Only the digits 0—9 and the minus sign (—) can be typed in the spin field.

  SPBS_READONLY Nothing can be typed in the spin field.

- Specify one of the following to determine how the text is to be presented in the spin field:

  SPBS_JUSTLEFT Left-justify the text. This is the default.

  SPBS_JUSTRIGHT Right-justify the text.

  SPBS_JUSTCENTER Center the text.

- Specify the following when you do not want a border around the spin button:

  SPBS_NOBORDER Suppresses drawing a border.

- Specify the following to increase the spin speed:

  SPBS_FASTSPIN Enables the spin button to increase the spin speed with time. The speed doubles every two seconds.

  Note: The spin button skips information when this option is specified. Do not use SPBS_FASTSPIN if the application requires that this field be checked each time a spin up

or spin down occurs. Do not specify this option on a master component that has servants spun from it.

- Specify the following to pad numeric fields with 0s. This is useful when the spin field contains values that represent time or money.

  SPBS_PADWITHZEROS The output number is padded at the front between the first non-zero digit and the field width, or 11 characters, whichever is the lesser. The negative sign, if there is one, is retained. The maximum number of characters required to display a LONG number is 11.

Spin Button Control Notification Message

This message is initiated by the spin button control window to notify its owner of significant events.

WM_CONTROL (in Spin Button Controls)

For the cause of this message, see "WM_CONTROL" on page 12-28.

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

reply *(ULONG)*

Reserved.

0 Reserved value, 0.

Remarks

This message is sent when, as specified by *notifycode*, the spin button component must tell its owner of a significant event.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return 0.

Spin Button Control Window Messages

This section describes the spin button control window procedure actions on receiving the following messages.

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

reply

fResult *(BOOL)*

Return.

TRUE Successful completion.
FALSE Error occurred.

Remarks

The application sends this message to the component to set or reset numeric limits.

This message is functionally identical to SPBM_SETLIMITS, except that the current value of the spin button does not change if it is out of range.

When the upper limit is less than the lower limit, FALSE is returned.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_QUERYLIMITS

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

Parameters

param1

lUpLimit *(LONG)*
Upper limit.

param2

lLowLimit *(LONG)*
Lower limit.

Returns

reply

fResult *(BOOL)*
Return.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

The application sends this message to the component to determine the limits of a numeric spin field.

When the spin button has no data, or when it is spinning an array, FALSE is returned.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

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

Consists of two USHORT parameters.

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

reply

fResult *(BOOL)*
Return.

TRUE Successful completion.
FALSE Error occurred.

Remarks

The application sends this message to the component to determine what value is in the spin field. The application sets up a field for the component to deposit the value, and sets a flag to determine what the function does when the value matches or does not match the given spin-list values.

TRUE is returned when a matched value is found, or the data is in the range.

FALSE is returned when no match is found, the value is out of range, or no spin data exists.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SETARRAY

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

Parameters

param1

pszStrl *(PSZ)*
Pointer.

Pointer to the new array of values.

param2

usItems *(USHORT)*
Number of items.

Number of items in the array.

Returns

reply

fResult *(BOOL)*
Return.

TRUE Successful completion.
FALSE Error occurred.

Remarks

The application sends this message to the component to set or reset the array of data.

The component tries to leave the current value unchanged. However, if the current value is out of range for the new array, it is moved to the closest extreme. Thus, if the current value is less than 0, it is moved to 0. If the current value is greater than the previous value, it is set to the previous value.

If the data exceeds 64KB, or if *param1* or *param2* equal 0, FALSE is returned.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

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.

0 Reserved value, 0.

Returns

reply

fResult *(BOOL)*
Return.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

The application sends this message to the component to set or reset the current numeric value or array index.

FALSE is returned when the value is out of range or there is no spin data.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SETLIMITS

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

Parameters

param1

lUpLimit *(LONG)*
Upper limit.

param2

lLowLimit *(LONG)*
Lower limit.

Returns

reply

fResult *(BOOL)*
Return.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

The application sends this message to the component to set or reset numeric limits. The component sets the current value to the content in the spin field when it is a valid number. When the current value is out of the range of the limits, it is moved to the nearest limit, upper or lower.

If the upper limit is less than the lower limit, FALSE is returned.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SETMASTER

This message causes the component to identify its master.

Parameters

param1

hwndHwnd *(HWND)*
Component handle.

Handle of master component.

param2

ulReserved *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fResult *(BOOL)*
Return.

TRUE Successful completion.
FALSE Error occurred.

Remarks

The application sends this message to the component to tell a component who its master is.

When the application wants to take control of the spin button, it must set the *param1* of each spin button to NULLHANDLE. This must be done, for example, when a spin button with a non-contiguous list of spin values is created (2, 4, 6, 8, 10...). When the *param1* of a spin button is NULLHANDLE, the spin button does not perform the following default functions:

- Spin up or down on its own when the Up or Down Arrow key is pressed.
- Spin up or down when the Up or Down Arrow of the master is pressed.
- A master does not take the focus when its arrows are pressed and none of its servants have focus.
- The spin button does not send itself an SPBM_QUERYVALUE message with the SPBQ_ALWAYSUPDATE flag to update the current value when an SPBM_SPINUP or SPBM_SPINDOWN message is received.
- The spin button does not fast spin.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

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.

0 Reserved value, 0.

Returns

reply

fResult *(BOOL)*
Return.

TRUE Successful completion.
FALSE Error occurred.

Remarks

The application sends this message to set the maximum number of characters allowed in the spin field. The size limit of the spin field is 255 characters. This is the default.

When the size exceeds 255 characters, FALSE is returned,

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SPINDOWN

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

Parameters

param1

ulItem *(ULONG)*
Number of values.

Number of values to spin down.

param2

ulReserved *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fResult *(BOOL)*
Return.

TRUE Successful completion.
FALSE Error occurred.

Remarks

The application sends this message to the component when it wants the previous value shown (spin backward).

When there is no data to spin, FALSE is returned.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SPINUP

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

Parameters

param1

ulItem *(ULONG)*
Number of values.

Number of values to spin up.

param2

ulReserved *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fResult *(BOOL)*
Return.

TRUE Successful completion.
FALSE Error occurred.

Remarks

The application sends this message to the component when it wants the next value shown (spin forward).

When there is no data to spin, FALSE is returned.

Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

Chapter 22. Static Control Window Processing

This system-provided window procedure processes the actions on a static control (WC_STATIC).

Purpose

Static controls are simple text fields, bit maps, icons, and boxes that can be used to label or box other controls. Static controls do not accept user input, nor do they send notification messages to their owner.

Static Control Styles

These static control styles are available:

SS_TEXT

Creates a box with formatted text. The text is formatted before it is displayed according to the setting of these text drawing-style flags:

Flag	Meaning
DT_LEFT	Left-justified text
DT_CENTER	Centered text
DT_RIGHT	Right-justified text

ORed with one of:

Flag	Meaning
DT_TOP	Text is aligned to top of window
DT_VCENTER	Text is aligned vertically in center of window
DT_BOTTOM	Text is aligned to bottom of window

The following text drawing style can also be ORed, but only if DT_TOP and DT_LEFT are also specified:

DT_WORDBREAK	Text is multi-line with word-wrapping at ends of lines.

Note: For "static" text that can be selected, a Button Control with a style of BS_NOBORDER can be used.

SS_GROUPBOX

A group box static control is a box that has an identifying text string in its upper left corner. Group boxes are used to collect a group of radio buttons or other controls into a single unit.

SS_ICON

Draws an icon. The text of the static control is a string that is used to derive the resource ID from which the icon is loaded. The format of the string is:

- The first byte is X'FF', the second byte is the low byte of the resource ID, and the third byte is the high byte of the resource ID.
- The first character is "#"; subsequent characters make up the decimal text representation of the resource ID. This format can be used for specifying a system icon in a resource file. The decimal string is the value of the appropriate SPTR_* constant

If the string is empty or does not follow the format above, no resource is loaded.

The resource is assumed to reside in the resource file of the current process.

This control is resized to the size of the icon.

SS_SYSICON

This style is the same as SS_ICON except that the icon ID is specified as one of the system pointer ID values (SPTR_* values) rather than a resource ID. This style provides a convenient way to include system icons in application dialog boxes.

SS_BITMAP	Draws a bit map. The text of the static control names the bit-map resource, as for SS_ICON.
SS_FGNDRECT	Creates a rectangle filled with the color of the foreground.
SS_BKGNDRECT	Creates a rectangle filled with the color of the background.
SS_FGNDFRAME	Creates a box with frame color equal to the foreground color.
SS_BKGNDFRAME	Creates a box with frame color equal to the background color.
SS_HALFTONERECT	Creates a rectangle filled with halftone shading.
SS_HALFTONEFRAME	Creates a box with halftone shading frame.
SS_AUTOSIZE	The static control will be sized to make sure the contents fit.

Static Control Data

None.

Default Colors

The following system colors are used when the system draws button controls:

SYSCLR_WINDOWFRAME
SYSCLR_WINDOWSTATICTEXT
SYSCLR_WINDOW
SYSCLR_BACKGROUND.

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_BORDERCOLOR
PP_FOREGROUNDCOLOR.

Static Control Notification Messages

No notification messages are initiated by the static control window procedure.

Static Control Window Messages

This section describes the static control window procedure actions on receiving the following messages.

SM_QUERYHANDLE

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

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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.

Remarks

The static control window procedure responds to this message by setting *hbmHandle* to the handle of the icon or bit-map of the static control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *hbmHandle* to the default value of 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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.

Remarks

The static control window procedure responds to this message by setting the icon or bit-map handle of a static control to the value specified by *hbmHandle*, and causes the static control to be redrawn, using the new item handle.

It should only be sent to a control with a style of SS_BITMAP, SS_ICON, or SS_SYSICON.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *hbmHandle* to the default value of 0.

WM_MATCHMNEMONIC (in Static Controls)

For the cause of this message, see "WM_MATCHMNEMONIC" on page 12-40.

Parameters

For a description of the parameters, see "WM_MATCHMNEMONIC" on page 12-40.

Remarks

The static control window procedure responds to this message by setting *fresult* as appropriate.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

WM_QUERYCONVERTPOS (in Static Controls)

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-51.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-51.

Remarks

The static control window procedure returns QCP_NOCONVERT.,

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-51.

WM_QUERYWINDOWPARAMS (in Static Controls)

This message occurs when an application queries the static control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Remarks

The static control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure sets the *ulText*, *ulPresParams*, and *ulCtlData* parameters of the WNDPARAMS data structure, identified by *pwndparams*, to zero and sets *fresult* to FALSE.

WM_SETWINDOWPARAMS (in Static Controls)

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

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-60.

Remarks

The static control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

Chapter 23. Title Bar Control Window Processing

This system-provided window procedure processes the actions on a title bar control (WC_TITLEBAR).

Purpose

The title bar control is the frame control that is used to display the application window title. It is also used to display the active or inactive status of the frame window.

The title bar control also implements the user interface for moving the frame window.

The standard identifier for a title bar control in a frame window is FID_TITLEBAR.

Title Bar Control Styles

There is only one title bar style, the default.

Title Bar Control Data

None.

Default Colors

The following system colors are used when the system draws button controls:

SYSCLR_ACTIVETITLETEXTBGND
SYSCLR_ACTIVETITLE
SYSCLR_ACTIVETITLETEXT,
SYSCLR_ACTIVETITLETEXTBGND
SYSCLR_INACTIVETITLE
SYSCLR_INACTIVETITLETEXT
SYSCLR_INACTIVETITLETEXTBGND
SYSCLR_TITLEBOTTOM
SYSCLR_(IN)ACTIVETITLETEXTBGND
SYSCLR_(IN)ACTIVETITLE.

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_FONTNAMESIZE
PP_ACTIVECOLOR
PP_INACTIVECOLOR
PP_ACTIVETEXT*COLOR
PP_INACTIVETEXT*COLOR
PP_ACTIVETEXTFGNDCOLOR
PP_INACTIVETEXTFGNDCOLOR
PP_BORDERCOLOR.

Title Bar Control Notification Messages

These messages are initiated by the title bar control to notify its owner of significant events.

WM_SYSCOMMAND (in Title Bar Controls)

For the cause of this message, see "WM_SYSCOMMAND" on page 12-63.

Parameters

For a description of the parameters, see "WM_SYSCOMMAND" on page 12-63.

The title bar control window procedure sets *uscmd* to the title bar control identity and *ussource* to CMDSRC_OTHER.

Remarks

The title bar control window procedure generates this message when a mouse input message is received. The window procedure posts the message to the queue of the window owner.

The purpose of this message is to notify the owner window to maximize or restore depending on its current state.

Default Processing

The default window procedure takes no action on this message, other than to set *flreply* to 0.

WM_TRACKFRAME (in Title Bar Controls

For the cause of this message, see "WM_TRACKFRAME" on page 12-66.

Parameters

For a description of the parameters, see "WM_TRACKFRAME" on page 12-66.

Remarks

The title bar control window procedure generates this message and sends it to its owner, informing the owner that a mouse button down message has been received.

Default Processing

The default window procedure takes no action on this message, other than to set *fresult* to FALSE.

Title Bar Control Window Messages

This section describes the title bar control window procedure actions on receiving the following messages.

TBM_QUERYHILITE

This message returns the highlighting state of a title-bar control.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fHighlighted *(BOOL)*
Highlighting state:

TRUE Title-bar control is highlighted
FALSE Title-bar control is not highlighted.

Remarks

The title bar control window procedure responds to this message by returning the highlighting state of the title-bar window.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fHighlighted* to the default value of FALSE.

TBM_SETHILITE

This message is used to highlight or unhighlight a title-bar control.

Parameters

param1

usHighlighted *(USHORT)*
Highlighting indicator:

TRUE Highlight the title-bar control
FALSE Remove highlight from the title-bar control.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

fSuccess *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The title bar control window procedure responds to this message by setting the highlighting state according to *usHighlighted*. If the title bar highlighting state is changed by this message, the title bar will repaint.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *fSuccess* to the default value of FALSE.

WM_QUERYCONVERTPOS (in Title Bar Controls)

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-51.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-51.

Remarks

The title bar control window procedure returns QCP_NOCONVERT.

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-51.

WM_QUERYWINDOWPARAMS (in Title Bars)

This message occurs when an application queries the title bar control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Default Processing

The title bar control window procedure queries the appropriate window parameters in accordance with *pwndparams* and sets *fresult* to TRUE if the operation is successful, otherwise to FALSE.

WM_SETWINDOWPARAMS (in Title Bar Controls)

This message occurs when an application sets or changes the title bar control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-60.

Default Processing

The title bar control window procedure sets the appropriate window parameters in accordance with *pwndparams* and sets *fresult* to TRUE if the operation is successful, otherwise to FALSE.

Chapter 24. Container Control Window Processing

This system-provided window procedure processes the actions on a container control (WC_CONTAINER).

Purpose

A container control is a visual component whose specific purpose is to hold objects. These objects, or container items, can be anything that either your application or a user might store in a container. Examples are executable programs, word processing files, graphics images, and database records.

Container item data is stored in RECORDCORE or MINIRECORDCORE data structures. Both the application and the container have access to the data stored in these records. See RECORDCORE on page A-110 and MINIRECORDCORE on page A-69 for descriptions of these data 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.

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 following list shows which types of data can be displayed for each container view. Refer to the description of the container control in the *OS/2 Programming Guide* for more information about the types of views.

View Types	Data
Icon view	Icons or bit maps with text strings beneath
Name view	Icons or bit maps with text strings to the right
Text view	Text strings
Tree view	Icons or bit maps, and text strings
Details view	Icons or bit maps, text strings, numbers, times, and dates.

Direct editing of container item text is supported in all views, including blank text fields.

The container control is designed according to the Common User Access (CUA) guidelines. For example, the CUA direct manipulation protocol is fully supported, enabling a user to visually drag an object in a container window and drop it on another object or container window. In addition, the container control supports CUA-defined selection types and techniques for selecting container items, as well as selection mechanisms, such as pointing devices and the keyboard, and multiple forms of emphasis. For a complete description of CUA containers, refer to the *SAA CUA Guide to User Interface Design* and to the *SAA CUA Advanced Interface Design Reference.*

The container control automatically provides or enables either horizontal or vertical scroll bars, or both, whenever all or part of one or more container items are not visible in a container window's client area.

Container Control Window Words

The container control reserves 4 bytes in its window words for application use. This memory can be accessed using the WinSetWindowULong and WinQueryWindowULong functions at offset QWL_USER.

Container Control Styles and Selection Types

Containers are WC_CONTAINER class windows that have the following CCS_container styles and selection types. Container control styles and selection types are specified when the container control is created.

Container Control Styles

The following list defines container style bits that your application can use. These style bits must be set by your application.

CCS_AUTOPOSITION
Automatic positioning, which causes container items displayed in the icon view to be arranged when any of the following occur:

- The window size changes
- Container items are inserted, removed, sorted, invalidated, or filtered
- The font or font size changes
- The window title text 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 (CV_ICON).

CCS_MINIRECORDCORE
A record style bit that causes the container to interpret all container records as being smaller than they would otherwise be. If a CM_ALLOCRECORD message is received, all records are interpreted and allocated according to the information in the MINIRECORDCORE data structure instead of the RECORDCORE data structure, which is used if this style bit is not specified.

CCS_READONLY
A read-only style bit for an entire container, which prevents a user from editing any of the text in a container window. If you do not set this style bit, a user can edit any of the text in a container window unless you set the following read-only attributes in the appropriate data structures:

CA_TITLEREADONLY
Sets the container title to read-only. This is an attribute of the CNRINFO data structure's *flWindowAttr* field.

CRA_RECORDREADONLY
Sets text fields in records to read-only. This is an attribute of the RECORDCORE and MINIRECORDCORE data structures' *flRecordAttr* field.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, the MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CFA_FIREADONLY
Sets column data to read-only. This is an attribute of the FIELDINFO data structure's *flData* field.

CFA_FITITLEREADONLY
Sets column headings to read-only. This is an attribute of the FIELDINFO data structure's *flTitle* field.

CCS_VERIFYPOINTERS
A pointer verification style bit, which verifies that the application pointers are members of the container's linked list before they are used. If it is not set, the container does not verify the pointers.

Notes

1. The CCS_VERIFYPOINTERS style bit does not verify the validity of a pointer. It only verifies whether a pointer is a member of a container's linked list.
2. After your code has been developed and tested, you may want to remove the CCS_VERIFYPOINTERS style bit in order to improve the container's performance. Otherwise, the container will attempt to verify all pointers, which will slow its response to actions that users perform.

Container Control Selection Types

If a selection type is not specified, single selection is the default. For the tree view, single selection is the only type supported. Refer to the description of the selection types in the *SAA CUA Advanced Interface Design Reference* for more information.

CCS_SINGLESEL
Single selection, which allows a user to select only one container item at a time. Each time a user selects a container item, the selection of any other container item is cancelled.

CCS_EXTENDSEL
Extended selection, which allows a user to select one or more container items. A user can select one item, a range of items, or multiple ranges of items.

CCS_MULTIPLESEL
Multiple selection, which allows a user to select zero or more container items.

Container Control Data

See the following for information on the container control data structures:

- CDATE on page A-10
- CNRDRAGINFO on page A-12
- CNRDRAGINIT on page A-12
- CNRDRAWITEMINFO on page A-13
- CNREDITDATA on page A-13
- CNRINFO on page A-15
- CTIME on page A-22
- FIELDINFO on page A-39
- FIELDINFOINSERT on page A-41
- MINIRECORDCORE on page A-69
- NOTIFYDELTA on page A-73
- NOTIFYRECORDEMPHASIS on page A-73
- NOTIFYRECORDENTER on page A-74
- NOTIFYSCROLL on page A-74
- OWNERBACKGROUND on page A-75
- QUERYRECFROMRECT on page A-108
- QUERYRECORDRECT on page A-109
- RECORDCORE on page A-110
- RECORDINSERT on page A-111
- SEARCHSTRING on page A-115
- TREEITEMDESC on page A-122.

Container Control Notification Messages

These messages are initiated by the container control window to notify its owner of significant events.

WM_CONTROL (in Container Controls)

For the cause of this message, see "WM_CONTROL" on page 12-28.

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 "Container Control Notification Codes" on page 24-8.

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_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_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* in "Container Control Notification Codes" on page 24-8.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The container control window procedure generates this message and sends it to its owner, informing the owner of this event.

Default Processing

For a description of the default processing, see "WM_CONTROL" on page 12-28.

WM_CONTROLPOINTER (in Container Controls)

For the cause of this message, see "WM_CONTROLPOINTER" on page 12-29.

Parameters

For a description of the parameters, see "WM_CONTROLPOINTER" on page 12-29.

Remarks

For the appropriate remarks, see "WM_CONTROLPOINTER" on page 12-29.

Default Processing

For the default processing, see "WM_CONTROLPOINTER" on page 12-29.

WM_DRAWITEM (in Container Controls)

For the cause of this message, see "WM_DRAWITEM" on page 12-31.

Parameters

param1

id *(USHORT)*
Container control ID.

param2

pOwnerItem *(POWNERITEM)*
Pointer.

Pointer to an OWNERITEM data structure. The following list defines the OWNERITEM data structure fields as they apply to the container control. See OWNERITEM on page A-76 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 *(USHORT)*
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 *(USHORT)*
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 A-110 and MINIRECORDCORE on page A-69 for descriptions of these data structures.

fsStateOld *(USHORT)*
Previous emphasis. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

fsAttributeOld *(USHORT)*
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 *(SHORT)*
Identifies the item being drawn. It can be one of the following:

- CMA_TEXT
- CMA_ICON
- CMA_TREEICON.

This field is not used for the details view and is set to 0.

hItem *(PCNRDRAWITEMINFO)*
Pointer to a CNRDRAWITEMINFO structure.

See CNRDRAWITEMINFO on page A-13 for descriptions of this structure's fields.

Returns

reply

drawn *(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.

Remarks

CA_OWNERDRAW is an attribute of the CNRINFO data structure's *flWindowAttr* field.

The container control window procedure generates this message and sends it to the owner of the container control to offer the owner the opportunity to draw that item.

Default Processing

For a description of the default processing, see "WM_DRAWITEM" on page 12-31.

Container Control Notification Codes

The following WM_CONTROL (in Container Controls) notification codes are sent by the container control to its owner.

CN_BEGINEDIT

The container control sends the WM_CONTROL (in Container Controls) message with the CN_BEGINEDIT notification code to its owner whenever container text is about to be edited.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_BEGINEDIT *(USHORT)*
Notification code.

param2

pCnrEditData *(PCNREDITDATA)*
Pointer.

Pointer to the CNREDITDATA structure. See CNREDITDATA on page A-13 for definitions of this structure's fields as they apply to the CN_BEGINEDIT notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The CN_BEGINEDIT notification code is sent when direct editing of container text begins.

Warning: Once your application receives the CN_BEGINEDIT notification code, it must not send any messages to the container until it receives the CN_ENDEDIT notification code, which indicates that direct editing of container text has ended. If any messages are sent to the container before your application receives the CN_ENDEDIT notification code, the results of direct editing are unpredictable.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_COLLAPSETREE

The container control sends the WM_CONTROL (in Container Controls) message with the CN_COLLAPSETREE notification code to its owner whenever the container collapses a parent item in the tree view.

Parameters

param1

Id *(USHORT)*
Container control ID.

CN_COLLAPSETREE *(USHORT)*
Notification code.

param2

pRecord *(PRECORDCORE)*
Pointer.

Pointer to the record that was 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.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_CONTEXTMENU

The container control sends the WM_CONTROL (in Container Controls) message with the CN_CONTEXTMENU notification code to its owner when the container receives a WM_CONTEXTMENU message.

Parameters

param1

Id *(USHORT)*
Container control ID.

CN_CONTEXTMENU *(USHORT)*
Notification code.

param2

pRecord *(PRECORDCORE)*
Pointer.

Pointer to the RECORDCORE structure that currently has the input focus. If the user is using a pointing device, this RECORDCORE structure is the structure that the pointing device pointer is over. If the pointing device pointer is over white space, this field is NULL.

If the user is using the keyboard, this RECORDCORE structure is the structure that has the selection cursor.

Returns

reply *(ULONG)*

Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DRAGAFTER

The container control sends a WM_CONTROL (in Container Controls) message with the CN_DRAGAFTER notification code to its owner whenever the container receives a DM_DRAGOVER message. The CN_DRAGAFTER notification code is sent only if the CA_ORDEREDTARGETEMPHASIS or CA_MIXEDTARGETEMPHASIS attribute of the CNRINFO data structure is set and the current view is the name, text, or details view.

Parameters

param1

id *(USHORT)*

Container control ID.

CN_DRAGAFTER *(USHORT)*

Notification code.

param2

pCnrDraginfo *(PCNRDRAGINFO)*

Pointer.

Pointer to a CNRDRAGINFO structure. See CNRDRAGINFO on page A-12 for definitions of this structure's fields as they apply to the CN_DRAGAFTER notification code.

Returns

reply

Reserved.

usDrop *(USHORT)*

Drop indicator.

DOR_DROP The record can be dropped. The drop will not occur unless DOR_DROP is returned. When this response is returned, the container control applies ordered target emphasis to the target record.

DOR_NODROP The record is acceptable and the current operation is supported by the target, but the record cannot be dropped in the current location. For example, the container control returns DOR_NODROP if the record being dragged is positioned over another record on which it cannot be dropped.

If the container returns DOR_NODROP, the DM_DRAGOVER message will continue to be sent to it when the user does any of the following:

- Moves the pointer
- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NODROPOP The record is 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, if the default operation is copy and the target does not support this operation, the drop may become valid if the user presses a keyboard augmentation key to change to a different operation, such as move.

If the container returns DOR_NODROPOP, no further DM_DRAGOVER messages are sent until the user does any of the following:

- Presses a keyboard key

- Moves the pointer out of and back into the container window.

DOR_NEVERDROP The record cannot be dropped. Ordered target emphasis is not drawn. If the container returns DOR_NEVERDROP, no further DM_DRAGOVER messages are sent until the user drags the record outside of and back into the container window.

usDefaultOp *(USHORT)*
Default operation.

Target-defined default operation.

DO_COPY	Operation is a copy.
DO_DEFAULT	Operation is the default drag operation. No modifier keys are pressed.
DO_LINK	Operation is a link.
DO_MOVE	Operation is a move.
DO_UNKNOWN	Operation is application-defined.

Remarks

The container control draws ordered target emphasis of container records. The target emphasis provided by the container control is a black line that is drawn below the target record. Therefore, it is not necessary for the application to draw any emphasis for the container when it receives this notification code.

If the container returns anything except DOR_DROP, the target emphasis is automatically changed to a symbol that indicates no drop is allowed. This gives the user a visual cue that a drop cannot occur. The symbol reverts to the black line when the container returns a DOR_DROP reply.

The CN_DRAGAFTER notification code is sent only for the details, name, and text views when the CA_ORDEREDTARGETEMPHASIS or CA_MIXEDTARGETEMPHASIS attribute of the CNRINFO data structure is set. If this attribute is not set, the CN_DRAGOVER notification code is sent.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DRAGLEAVE

The container control sends a WM_CONTROL (in Container Controls) message with the CN_DRAGLEAVE notification code to its owner when the container receives a DM_DRAGLEAVE message.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_DRAGLEAVE *(USHORT)*
Notification code.

param2

pCnrDraginfo *(PCNRDRAGINFO)*
Pointer.

Pointer to a CNRDRAGINFO structure. See CNRDRAGINFO on page A-12 for definitions of this structure's fields as they apply to the CN_DRAGLEAVE notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This notification code is sent to the owner of the container control in response to a DM_DRAGLEAVE message. It informs the owner that one of the following has occurred:

- A container record was being dragged over the container and has left the container's boundaries.
- The drag ended when help was requested or a user pressed the Esc key while the container record was over the container.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DRAGOVER

The container control sends a WM_CONTROL (in Container Controls) message with the CN_DRAGOVER notification code to its owner when the container receives 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.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_DRAGOVER *(USHORT)*
Notification code.

param2

pCnrDraginfo *(PCNRDRAGINFO)*
Pointer.

Pointer to a CNRDRAGINFO structure. See CNRDRAGINFO on page A-12 for definitions of this structure's fields as they apply to the CN_DRAGOVER notification code.

Returns

reply
Reserved.

usDrop *(USHORT)*
Drop indicator.

DOR_DROP The record can be dropped. When this response is returned, the container control applies target emphasis.

DOR_NODROP The record is acceptable and the current operation is supported by the target, but the record cannot be dropped in the current location. For example, the container control returns DOR_NODROP if the record being dragged is positioned over another record on which it cannot be dropped.

If the container returns DOR_NODROP, the DM_DRAGOVER message will continue to be sent to it when the user does any of the following:

- Moves the pointer
- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NODROPOP The record is 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, if the default operation is copy and the target does not support this operation, the drop may become valid if the user presses a keyboard augmentation key to change to a different operation, such as move.

If the container returns DOR_NODROPOP, no further DM_DRAGOVER messages are sent until the user does any of the following:

- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NEVERDROP The record cannot be dropped. Target emphasis is not drawn. If the container returns DOR_NEVERDROP, no further DM_DRAGOVER messages are sent until the user drags the record outside of and back into the container window.

usDefaultOp *(USHORT)*
Default operation.

Target-defined default operation.

DO_COPY	Operation is a copy.
DO_DEFAULT	Operation is the default drag operation. No modifier keys are pressed.
DO_LINK	Operation is a link.
DO_MOVE	Operation is a move.
DO_UNKNOWN	Operation is application-defined.

Remarks

This notification code shows where direct manipulation is occurring by applying target emphasis to indicate whether an item that is being dragged over the container can be dropped. It is not necessary for the application to draw any target emphasis for the container when it receives this notification code.

If the pointer is over a container record and the item that is being dragged can be dropped on that record, the container draws a black rectangle around the target record. If the pointer is over white space and the item that is being dragged can be dropped on the white space, the container draws a black border around the edge of the client area.

If the container returns anything except DOR_DROP, the target emphasis is automatically changed to a symbol that indicates no drop is allowed. This gives the user a visual cue that a drop cannot occur. The symbol reverts to the black rectangle or black border when the container returns a DOR_DROP reply.

The CN_DRAGOVER notification code is sent only for the icon and tree views, or when the CA_ORDEREDTARGETEMPH attribute of the CNRINFO data structure is not set. If this attribute is set and the current view is the name, text, or details view, the CN_DRAGAFTER notification code is sent.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DROP

The container control sends a WM_CONTROL (in Container Controls) message with the CN_DROP notification code to its owner when the container receives a DM_DROP message.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_DROP *(USHORT)*
Notification code.

param2

pCnrDraginfo *(PCNRDRAGINFO)*
Pointer.

Pointer to a CNRDRAGINFO structure. See CNRDRAGINFO on page A-12 for definitions of this structure's fields as they apply to the CN_DROP notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This notification code is sent to the container's owner when dragged container records are dropped over the container window.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DROPHELP

The container control sends a WM_CONTROL (in Container Controls) message with the CN_DROPHELP notification code to its owner when the container receives a DM_DROPHELP message.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_DROPHELP *(USHORT)*
Notification code.

param2

pCnrDraginfo *(PCNRDRAGINFO)*
Pointer.

Pointer to a CNRDRAGINFO structure. See CNRDRAGINFO on page A-12 for definitions of this structure's fields as they apply to the CN_DROPHELP notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This notification code is sent to the container's owner when help for direct manipulation is requested over the container window.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_EMPHASIS

The container control sends a WM_CONTROL (in Container Controls) message with the CN_EMPHASIS notification code to its owner whenever a container record's attributes change.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_EMPHASIS *(USHORT)*
Notification code.

param2

pNotifyRecordEmphasis *(PNOTIFYRECORDEMPHASIS)*
Pointer.

Pointer to the NOTIFYRECORDEMPHASIS structure. See NOTIFYRECORDEMPHASIS on page A-73 for definitions of this structure's fields as they apply to the CN_EMPHASIS notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_ENDEDIT

The container control sends a WM_CONTROL (in Container Controls) message with the CN_ENDEDIT notification code to its owner whenever direct editing of container text has ended.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_ENDEDIT *(USHORT)*
Notification code.

param2

pCnrEditData *(PCNREDITDATA)*
Pointer.

Pointer to the CNREDITDATA structure. See CNREDITDATA on page A-13 for definitions of this structure's fields as they apply to the CN_ENDEDIT notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

Direct editing of container text is completed. Any changes made to the text are saved when a user presses the select button outside the window that contains the multiple-line entry (MLE) field used to edit text in a container. However, a user can end the direct editing of text without saving any changes to the text by doing any of the following:

- Pressing the Esc key
- Dragging the container item that is being edited
- Pressing the Alt key and the select button before 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.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_ENTER

The container control sends a WM_CONTROL (in Container Controls) message with the CN_ENTER notification code to its owner when either of the following occurs:

- The Enter key is pressed while the container window has the focus
- The select button is double-clicked while the pointer is over the container window.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_ENTER *(USHORT)*
Notification code.

param2

pNotifyRecordEnter *(PNOTIFYRECORDENTER)*
Pointer.

Pointer to the NOTIFYRECORDENTER structure. See NOTIFYRECORDENTER on page A-74 for definitions of this structure's fields as they apply to the CN_ENTER notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_EXPANDTREE

The container control sends the WM_CONTROL (in Container Controls) message with the CN_EXPANDTREE notification code to its owner whenever the container expands a parent item in the tree view.

Parameters

param1

Id *(USHORT)*
Container control ID.

CN_EXPANDTREE *(USHORT)*
Notification code.

param2

pRecord *(PRECORDCORE)*
Pointer.

Pointer to the record that was 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.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_HELP

The container control sends a WM_CONTROL (in Container Controls) message with the CN_HELP notification code to its owner whenever the container receives a WM_HELP message.

Parameters

param1

Id *(USHORT)*
Container control ID.

CN_HELP *(USHORT)*
Notification code.

param2

pRecord *(PRECORDCORE)*
Pointer.

Pointer to the record that has the selection cursor.

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.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This notification code is sent to the container's owner when help is requested for a container item.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_INITDRAG

The container control sends a WM_CONTROL (in Container Controls) message with the CN_INITDRAG notification code to its owner when the drag button is pressed and the pointer is moved while the pointer is over the container control.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_INITDRAG *(USHORT)*
Notification code.

param2

pCnrDraginit *(PCNRDRAGINIT)*
Pointer.

Pointer to the CNRDRAGINIT structure. See CNRDRAGINIT on page A-12 for descriptions of this structure's fields as they apply to the CN_INITDRAG notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This notification code is sent to the container's owner when the drag button is pressed and the pointer is moved while the pointer is over the container control.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_KILLFOCUS

The container control sends a WM_CONTROL (in Container Controls) message with the CN_KILLFOCUS notification code to its owner whenever the container is losing the focus.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_KILLFOCUS *(USHORT)*
Notification code.

param2

hwndCnr *(HWND)*
Container control handle.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_QUERYDELTA

The container control sends a WM_CONTROL (in Container Controls) message with the CN_QUERYDELTA notification code to its owner to query for more data when a user scrolls to a preset delta value.

Parameters

param1

id *(USHORT)*
Container control ID.

CN_QUERYDELTA *(USHORT)*
Notification code.

param2

pNotifyDelta *(PNOTIFYDELTA)*
Pointer.

Pointer to the NOTIFYDELTA structure. See NOTIFYDELTA on page A-73 for definitions of this structure's fields as they apply to the CN_QUERYDELTA notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The delta value is specified by the *cDelta* field of the CNRINFO data structure and is set with the CMA_DELTA attribute of the CM_SETCNRINFO message. If the value of the *cDelta* field is greater than 0 and a user scrolls to the threshold record, the container control sends a CN_QUERYDELTA notification code to the application. The application can then insert more records into the container. It may be necessary for the application to remove some records before inserting records.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_REALLOCPSZ

The container control sends a WM_CONTROL (in Container Controls) message with the CN_REALLOCPSZ notification code to its owner whenever container text is edited. It is sent before the CN_ENDEDIT notification code is sent.

Parameters

param1

Id *(USHORT)*
Container control ID.

CN_REALLOCPSZ *(USHORT)*
Notification code.

param2

pCnrEditData *(PCNREDITDATA)*
Pointer.

Pointer to the CNREDITDATA structure. See CNREDITDATA on page A-13 for definitions of this structure's fields as they apply to the CN_REALLOCPSZ notification code.

Returns

fSuccess *(BOOL)*
Success indicator.

TRUE	The application has sufficient memory for the new text string.
FALSE	The application has insufficient memory for the new text string or does not want the string to be copied.

Remarks

The CN_REALLOCPSZ notification code is sent after direct editing of container text is complete. It notifies the application that the container is about to copy the changed text to the application's text string. This allows the application to ensure that the correct amount of memory is allocated to accommodate the change.

If TRUE is returned by the application, the container control copies the new text to the application's text string. However, if the application returns FALSE, changed text is disregarded.

Warning: Once your application receives the CN_REALLOCPSZ notification code, it must not send any messages to the container until it receives the CN_ENDEDIT notification code, which indicates that direct editing of container text has ended. If any messages are sent to the container before your application receives the CN_ENDEDIT notification code, the results of direct editing are unpredictable.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return FALSE.

CN_SCROLL

The container control sends a WM_CONTROL (in Container Controls) message with the CN_SCROLL notification code to its owner whenever the container window scrolls.

Parameters

param1

Id *(USHORT)*
Container control ID.

CN_SCROLL *(USHORT)*
Notification code.

param2

pNotifyScroll *(PNOTIFYSCROLL)*
Pointer.

Pointer to the NOTIFYSCROLL structure. See NOTIFYSCROLL on page A-74 for definitions of this structure's fields as they apply to the CN_SCROLL notification code.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_SETFOCUS

The container control sends a WM_CONTROL (in Container Controls) message with the CN_SETFOCUS notification code to its owner whenever the container receives the focus.

Parameters

param1

Id *(USHORT)*
Container control ID.

CN_SETFOCUS *(USHORT)*
Notification code.

param2

hwndCnr *(HWND)*
Container control handle.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

Container Control Window Messages

This section describes the container control window procedure actions on receiving the following messages.

CM_ALLOCDETAILFIELDINFO

This message allocates memory for one or more FIELDINFO structures.

Parameters

param1

nFieldInfo *(USHORT)*
Number of FIELDINFO structures.

Number of FIELDINFO structures to be allocated. The value of this parameter must be greater than 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

pFieldInfo *(PFIELDINFO)*
Pointer or error.

Returns a pointer to one or more FIELDINFO structures if allocation is successful.

Returns an error if allocation fails.

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.

Remarks

The container control requires that the application use the CM_ALLOCDETAILFIELDINFO message to allocate memory for any FIELDINFO structures that are used.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 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.

Returns a pointer to one or more RECORDCORE structures if allocation is successful.

Returns an error if allocation fails.

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 *pNextRecord* field in each RECORDCORE data structure. The last pointer is set to NULL.

Remarks

The container control requires that the application use the CM_ALLOCRECORD message to allocate memory for container records.

When a record is allocated, the *cb* field of the record will be initialized with the size of the record structure type currently in use, either RECORDCORE or MINIRECORDCORE. If the CCS_MINIRECORDCORE style bit is not specified, the record is allocated according to the size of the RECORDCORE data structure. However, if the CCS_MINIRECORDCORE style bit is specified, the record is allocated according to the size of the MINIRECORDCORE data structure. This size should not be modified by the application.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_ARRANGE

This message arranges the container records in the icon view of the container control.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fSuccess *(BOOL)*
Success indicator.

TRUE Icon/text or bit-map/text pairs were successfully arranged.
FALSE An error occurred.

Remarks

The container items fill the topmost row until the width of the client area is reached. The container items then wrap to form another row immediately below the filled row. This process is repeated until all of the container items are positioned in rows. Default spacing is implemented according to the guidelines for the CUA user interface. A vertical scroll bar is enabled, if necessary.

Before the relocation of the container items, the origin of the client area rectangle is reset to coincide with the origin of the container's workspace. Arranging the container items does not affect the record attributes.

If the CCS_AUTOPOSITION style bit is set, you do not need to send the CM_ARRANGE message, since this style bit causes the container control to arrange the container items for the application.

If the current view is not the icon view, no visible change occurs until the current view is switched to the icon view. For example, if the name view is the current view and the CM_ARRANGE message is sent, the display does not change.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_CLOSEEDIT

This message closes the window that contains the multiple-line entry (MLE) field used to edit container text directly.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fSuccess *(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.

Remarks

The application sends this message to the container control to end the direct editing of container text. The application can assign this message to a key or key combination, a menu choice, or both so that the user can end the direct editing of container text from the keyboard.

When the container control receives this message, it sends the CN_REALLOCPSZ and CN_ENDEDIT notification codes to the application.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_COLLAPSETREE

This message causes one parent item in the tree view to be collapsed.

Parameters

param1

pRecord *(PRECORDCORE)*

Pointer.

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 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

fSuccess *(BOOL)*

Success indicator.

TRUE The item was successfully collapsed.

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

PMERR_INVALID_PARAMETERS.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fSuccess *(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.

Remarks

The container record is not removed and memory is not freed; only the visual appearance is changed. The visibility flag associated with the container record is not changed.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_EXPANDTREE

This message causes one parent item in the tree view to be expanded.

Parameters

param1

pRecord *(PRECORDCORE)*
Pointer.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fSuccess *(BOOL)*

Success indicator.

TRUE The item was successfully expanded.

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

PMERR_INVALID_PARAMETERS.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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.

Pointer to an application-supplied filter function.

param2

pStorage *(PVOID)*

Application use.

Available for application use.

Returns

fSuccess *(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.

Remarks

Filtering is enabled by setting the CRA_FILTERED attribute of container records that are to be excluded from the viewable subset.

The *pfnFilter* parameter points to an application-provided function that determines whether a record is to be included in the viewable subset. The *pfnFilter* parameter must be declared as:

```
BOOL PFN pfnFilter (PRECORDCORE p, PVOID pStorage);
```

where **p** points to a RECORDCORE structure that describes the container record to be tested. The *pfnFilter* parameter returns TRUE if the record is to be included in the viewable subset, or FALSE if it is to be excluded. The container sets the CRA_FILTERED attribute for the record based on the return from the *pfnFilter* parameter.

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 the CRA_FILTERED attribute is set for the record, the record is not visible. If the CCS_AUTOPOSITION style bit is set and the container is showing the icon view, the container records are arranged when a record is filtered out.

The CM_FILTER message supports only one level of filtering.

It is the application's responsibility to provide a National Language Support-enabled (NLS-enabled) function for the *pfnFilter* parameter.

If the *pfnFilter* parameter value is NULL, a container is returned to an unfiltered state. If functions such as inserting a record into a container, arranging the records, or sorting the records are performed on a container whose records have been filtered, the effect of these functions remains if the container records are later unfiltered.

All messages act on the entire container. For example, a record that is filtered and is removed from the container will be removed from the container entirely; it is not present in the container when the container records are unfiltered.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_FREEDETAILFIELDINFO

This message frees the memory associated with one or more FIELDINFO structures.

Parameters

param1

pFieldInfoArray *(PVOID)*
Pointer.

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

fSuccess *(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.

Remarks

It is the application's responsibility to free all application-allocated memory associated with the structures, such as user data.

If a specified FIELDINFO structure is currently inserted into the container, the structure is not freed and the PMERR_FI_CURRENTLY_INSERTED error is set. FIELDINFO structures must be removed with the CM_REMOVEDETAILFIELDINFO message before the CM_FREEDETAILFIELDINFO message is used.

If the number of pointers to FIELDINFO structures in the array exceeds the count of structures to be freed, only the number of structures in the *cNumFieldInfo* parameter is freed. If either the *pFieldInfoArray* or the *cNumFieldInfo* parameter is invalid, the PMERR_INVALID_PARAMETERS error is set and no FIELDINFO structures are freed.

If the PMERR_MEMORY_DEALLOCATION_ERR error occurs, any further processing is unreliable.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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.

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

fSuccess *(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.

Remarks

It is the application's responsibility to free all application-allocated memory associated with the container records, such as text strings.

If a specified record is currently inserted into the container, the record is not freed and the PMERR_RECORD_CURRENTLY_INSERTED error is set. Container records must be removed with the CM_REMOVERECORD message before the CM_FREERECORD message is used.

If the number of pointers to container records in the array exceeds the count of records to be freed, only the number of records in the *cNumRecord* parameter is freed. If either the *pRecordArray* or the *cNumRecord* parameter is invalid, the PMERR_INVALID_PARAMETERS error is set and no container records are freed.

If the PMERR_MEMORY_DEALLOCATION_ERR error occurs, any further processing is unreliable.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

fSuccess *(BOOL)*
Success indicator.

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

PMERR_INVALID_PARAMETERS.

Remarks

The *lScrollInc* parameter indicates a change in position. If the *lScrollInc* parameter value is greater than 0, the window specified in the *usWindow* parameter is scrolled to the right by the number of pixels specified in the *lScrollInc* parameter. If the value of the *lScrollInc* parameter is less than 0, the window specified in the *usWindow* parameter is scrolled to the left by the number of pixels specified in the *lScrollInc* parameter. This message is used to scroll either the left or right split window by an absolute amount.

The columns that are to appear in each split window are determined at the time the split window is created. Thereafter, columns in the left split window cannot be seen in the right split window, and vice versa.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_INSERTDETAILFIELDINFO

This message inserts one or more FIELDINFO structures into a container control.

Parameters

param1

pFieldInfo *(PFIELDINFO)*
Pointer.

Pointer to the FIELDINFO structure or structures to insert.

param2

pFieldInfoInsert *(PFIELDINFOINSERT)*
Pointer.

Pointer to the FIELDINFOINSERT data structure. See FIELDINFOINSERT on page A-41 for the descriptions of this structure's fields as they apply to the CM_INSERTDETAILFIELDINFO message.

Returns

cFields *(USHORT)*

Number of structures.

Number of FIELDINFO structures in the container.

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.

Remarks

The *pFieldInfoInsert* parameter is used to insert FIELDINFO structures into the container. The *pFieldInfoOrder* field of the FIELDINFOINSERT data structure is used to place FIELDINFO structures into the container in order, relative to the other structures. Specifying the CMA_FIRST attribute places the FIELDINFO structure at the front of the list of structures. If the CMA_END attribute is specified, the FIELDINFO structure is placed at the end of the list of structures. Otherwise, if the value of the *pFieldInfoOrder* field is a pointer to a FIELDINFO structure, the structure being inserted is placed after this structure.

If the value of the *cFieldInfoInsert* field of the FIELDINFOINSERT data structure is greater than 1, a linked list of FIELDINFO structures is inserted in the order specified by the *pFieldInfoOrder* field. Here, the *pFieldInfo* parameter points to the first of a linked list of FIELDINFO structures. This list of structures is linked together as they were when the FIELDINFO structures were allocated.

If one FIELDINFO structure is to be inserted, the *cFieldInfoInsert* field has a value of 1 and the *pFieldInfo* parameter points to the FIELDINFO structure to be inserted.

After the FIELDINFO structures have been inserted, if the *fInvalidateFieldInfo* field of the FIELDINFOINSERT data structure is FALSE, the CM_INVALIDATEDETAILFIELDINFO message must be sent to update the display with the inserted structures.

If the CCS_VERIFYPOINTERS style bit is set and the *pFieldInfo* parameter contains a pointer to a FIELDINFO structure that is currently inserted, the PMERR_FI_CURRENTLY_INSERTED error is set and no FIELDINFO structures are inserted.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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.

Pointer to the RECORDCORE structure or structures to insert.

param2

pRecordInsert *(PRECORDINSERT)*

Pointer.

Pointer to the RECORDINSERT data structure. See RECORDINSERT on page A-111 for definitions of this structure's fields as they apply to the CM_INSERTRECORD message.

Returns

cRecords *(ULONG)*

Number of structures.

Number of RECORDCORE structures in the root level of the container.

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.

Remarks

The *pRecordInsert* parameter is used to insert RECORDCORE structures into the container. The *pRecordOrder* and *pRecordParent* fields of the RECORDINSERT data structure are used to place each record into the container in order, relative to the other records. If the CMA_FIRST or CMA_END attributes are specified, records are inserted before the first child or after the last child of the record specified in the *pRecordParent* field. If the value of the *pRecordParent* field is NULL, the record or records are inserted before the first record or after the last record, respectively, at the root level. Otherwise, if the value of the *pRecordOrder* field is a pointer to a record, the record or records to be inserted are placed after this record.

A z-ordering of the records is maintained by the container control. The *zOrder* field of the RECORDINSERT data structure is used to specify the record's z-order in the container, relative to the other records. The CMA_TOP attribute is used to place the record at the end of the z-order list, while the CMA_BOTTOM attribute places the record at the beginning of the z-order list. Z-ordering is used for the icon view only.

If the value of the *cRecordsInsert* field of the RECORDINSERT data structure is greater than 1, a linked list of RECORDCORE structures is inserted in the order specified by the *pRecordOrder*, *pRecordParent*, and *zOrder* fields. Here, the *pRecord* parameter points to the first RECORDCORE structure of a linked list of structures.

If one RECORDCORE structure is to be inserted, the *cRecordsInsert* field has a value of 1 and the *pRecord* parameter points to the RECORDCORE structure to be inserted.

When containers display the icon view, the coordinates specified by the RECORDCORE structure's *ptlIcon* field are used to position inserted container records in the container's workspace. If the coordinates are not specified and the CCS_AUTOPOSITION style bit is not set, all of the inserted container records are positioned at **(0,0)** and a CM_ARRANGE message must be sent to position them elsewhere. If the CCS_AUTOPOSITION style bit is set, the container records are positioned without the CM_ARRANGE message being sent.

After the container records have been inserted:

- If the *flInvalidateRecord* field of the RECORDINSERT data structure is FALSE, the CM_INVALIDATERECORD message must be sent to update the display with the inserted records. If the current view is the icon view and either the CCS_AUTOPOSITION style bit is set or the *flInvalidateRecord* field is TRUE, the view is updated without the CM_INVALIDATERECORD message being sent.
- The *pNextRecord*, *flRecordAttr*, and *ptlIcon* fields of the external RECORDCORE structure are not updated as changes occur within the container. However, if records are shared among multiple containers, the *flRecordAttr* and *ptlIcon* fields are modified internally. Refer to the *OS/2 2.00 Programming Guide* for more information about the modification of these fields.

If the CCS_VERIFYPOINTERS style bit is set and the *pRecord* parameter contains a pointer to a RECORDCORE structure that is currently inserted, the PMERR_RECORD_CURRENTLY_INSERTED error is set and no RECORDCORE structures are inserted.

If the RECORDCORE structures are sorted on insertion, the *pRecordOrder* and *zOrder* fields are ignored.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*

Reserved.

0 Reserved value, 0.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

fSuccess *(BOOL)*

Success indicator.

TRUE FIELDINFO structures were successfully refreshed.

Remarks

If any or all FIELDINFO structures are changed, removed, or inserted, the CM_INVALIDATEDETAILFIELDINFO message must be sent. Since each FIELDINFO structure potentially affects every record in the container, the entire view is refreshed, even if only one FIELDINFO structure has changed.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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.

Pointer to an array of pointers to RECORDCORE structures that are to be refreshed.

param2

cNumRecord *(USHORT)*

Number of records.

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.

flnvalidateRecord *(USHORT)*

Flags.

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

fSuccess *(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.

Remarks

If the number of pointers to container records in the array exceeds the count of records to be refreshed, only the number of records specified in the *cNumRecord* parameter is refreshed. If the CCS_VERIFYPOINTERS style bit is set and the *pRecordArray* parameter contains pointers to a RECORDCORE structure or structures that do not exist, the PMERR_INVALID_PARAMETERS error is set and nothing is refreshed.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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.

Pointer to the CNREDITDATA structure. See CNREDITDATA on page A-13 for definitions of this structure's fields as they apply to the CM_OPENEDIT message.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fSuccess *(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.

Remarks

The application sends this message to the container control to start the direct editing of container text. The application can assign this message to a key or key combination, a menu choice, or both so that the user can start editing container text directly from the keyboard.

When the container control receives this message, it sends the CN_BEGINEDIT notification code to the application.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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.

Pointer to the OWNERBACKGROUND structure. See OWNERBACKGROUND on page A-75 for definitions of this structure's fields as they apply to the CM_PAINTBACKGROUND message.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fProcess *(BOOL)*
Process indicator.

TRUE The application processed the CM_PAINTBACKGROUND message.
FALSE The application did not process the CM_PAINTBACKGROUND message.

Remarks

The CM_PAINTBACKGROUND message is provided so that an application can subclass the container control and paint its own background. If the application does not subclass the container control or subclasses the container control and returns FALSE, the container uses the system window color, which is specified by SYSCLR_WINDOW. This color can be changed by using the PP_BACKGROUNDCOLOR or PP_BACKGROUNDCOLORINDEX presentation parameter of the WM_PRESPARAMCHANGED (in Container Controls) message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_QUERYCNRINFO

This message returns the container's CNRINFO structure.

Parameters

param1

pCnrInfo *(PCNRINFO)*
Pointer.

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.

Remarks

The number of bytes specified in the *cbBuffer* parameter is returned in the buffer addressed by the *pCnrInfo* parameter.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

CM_QUERYDETAILFIELDINFO

This message returns a pointer to the requested FIELDINFO structure.

Parameters

param1

pFieldInfo *(PFIELDINFO)*
Pointer.

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.

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.

Pointer to the FIELDINFO structure for which data was requested.

NULL	No FIELDINFO structures to retrieve.
Negative one	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.

Remarks

If the *cmd* parameter has the value of the CMA_FIRST or CMA_LAST attribute, the *pFieldInfo* parameter is ignored and the first or last column data, respectively, is returned. If the CMA_NEXT or the CMA_PREV attribute is set in the *cmd* parameter, the column data next to or before the column pointed to by the *pFieldInfo* parameter is returned.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

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.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

hImage *(LHANDLE)*
Image handle.

Handle of the image currently displayed for a record.

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.

Remarks

If the CCS_MINIRECORDCORE style bit is specified, this function will always return the MINIRECORDCORE structure's *hptrIcon* field.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULLHANDLE.

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.

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.

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, lowest to highest.
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.

Pointer to the RECORDCORE structure for which data was requested.

NULL	No RECORDCORE structures to retrieve.
Negative one	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.

Remarks

If the *cmd* parameter has the value of CMA_FIRST or CMA_LAST, the *pRecord* parameter in *param1* is ignored and the first or last record, respectively, in the container is returned.

Depending on the value of the *fsSearch* parameter, the container records are enumerated in item order or in z-order.

See RECORDCORE on page A-110 or MINIRECORDCORE on page A-69 for a complete list and descriptions of all container record attributes.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_QUERYRECORDEMPHASIS

This message queries for a container record with the specified emphasis attributes.

Parameters

param1

pSearchAfter *(PRECORDCORE)*
Pointer.

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:

CMA_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_CURSORED
CRA_INUSE
CRA_SELECTED

Returns

pRecord *(PRECORDCORE)*
Pointer.

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.
Negative one	The container record data was not returned. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS.
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.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_QUERYRECORDFROMRECT

This message queries for a container record that is bounded by the specified rectangle.

Parameters

param1

pSearchAfter *(PRECORDCORE)*
Pointer.

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.

Pointer to the QUERYRECFROMRECT data structure. See QUERYRECFROMRECT on page A-108 for definitions of this structure's fields as they apply to the CM_QUERYRECORDFROMRECT message.

Returns

pRecord *(PRECORDCORE)*
Pointer.

Pointer to the container records within the bounding rectangle.

NULL	No container records are within the bounding rectangle.
Negative one	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.

Remarks

This message returns the pointer to the first container record found in the rectangle after the starting position specified in the *pSearchAfter* parameter.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_QUERYRECORDINFO

This message updates the specified records with the current information for the container.

Parameters

param1

pRecordArray *(PVOID)*
Pointer.

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 in 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

fSuccess *(BOOL)*
Success indicator.

TRUE Record information was successfully updated.
FALSE An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

Remarks

This message is needed only if the application is sharing records among multiple containers in the same process.

The *flRecordAttr* and *ptlIcon* fields are updated internally when they change, but not in the external RECORDCORE structure. Therefore, the application's external record does not always have current information in these fields. This message is only needed if the application is sharing records among multiple containers in the same process.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_QUERYRECORDRECT

This message returns the rectangle of the specified container record, relative to the container window origin.

Parameters

param1

prclItem *(PRECTL)*
Pointer.

Pointer to the RECTL structure, into which the rectangular coordinates are placed.

param2

pQueryRecordRect *(PQUERYRECORDRECT)*
Pointer.

Pointer to the QUERYRECORDRECT structure. See QUERYRECORDRECT on page A-109 for definitions of this structure's fields as they apply to the CM_QUERYRECORDRECT message.

Returns

fSuccess *(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.

Remarks

The coordinates of the returned rectangle are in window coordinates.

If the input record is not found in the container, the output rectangle is empty.

For a container using the details view (CV_DETAIL), all of the data for a row is returned in the rectangle.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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.

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

fSuccess *(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.

Remarks

The virtual coordinates of the client area rectangle are written into the structure addressed by the *prclViewport* parameter.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_REMOVEDETAILFIELDINFO

This message removes one, multiple, or all FIELDINFO structures from the container control.

Parameters

param1

pFieldInfoArray *(PVOID)*
Pointer.

Pointer to an array of pointers to FIELDINFO structures that are to be removed.

param2

cNumFieldInfo *(USHORT)*
Number of structures.

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.

Number of FIELDINFO structures remaining in the container.

Negative one 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.

Remarks

The FIELDINFO structures are removed from the list of columns inserted into the container control.

If the CMA_FREE attribute is not specified, the container control removes the specified FIELDINFO structures without freeing the memory. The application is responsible for freeing the memory associated with the FIELDINFO structures by using the CM_FREEDETAILFIELDINFO message.

If the *cNumFieldInfo* parameter has a value of 0 and the CMA_FREE attribute is specified, all of the FIELDINFO structures in the container control are removed and the memory associated with the FIELDINFO structures is freed. It is the application's responsibility to free all of the application-allocated memory associated with the FIELDINFO structures.

If the number of pointers to FIELDINFO structures in the array exceeds the count of FIELDINFO structures to be removed, only the number of structures specified in the *cNumFieldInfo* parameter are removed. If the CCS_VERIFYPOINTERS style bit is set and the *pFieldInfoArray* parameter contains pointers to a FIELDINFO structure or structures that do not exist, the PMERR_INVALID_PARAMETERS error is set.

If you do not want to show a column, you can hide it by setting the CFA_INVISIBLE attribute of the FIELDINFO data structure and notifying the container control with the CM_INVALIDATEDETAILFIELDINFO message.

If the CMA_INVALIDATE attribute is specified, the container is repainted.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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.

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 set. In the icon view, the container record is refreshed if the CCS_AUTOPOSITION style bit is not set, regardless of whether the CMA_INVALIDATE attribute is set.

Returns

cRecords *(LONG)*
Number of structures.

Number of root level RECORDCORE structures that remain in the container.

Negative one 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.

Remarks

When parent item records are removed, all associated child item records are removed, as well.

If the CMA_FREE attribute is not specified, the container control removes the specified RECORDCORE structures without freeing the memory. The application is responsible for freeing the memory associated with the RECORDCORE structure by using the CM_FREERECORD message.

If the *cNumRecord* parameter has a value of 0 and the CMA_FREE attribute is specified, all of the RECORDCORE structures in the container control are removed and the memory associated with the RECORDCORE structures is freed. It is the application's responsibility to free all of the application-allocated memory associated with the RECORDCORE structures.

If the number of pointers to RECORDCORE structures in the array exceeds the count of RECORDCORE structures to be removed, only the number of records specified in the *cNumRecord* parameter is removed. If the CCS_VERIFYPOINTERS style bit is set and the *pRecordArray* parameter contains pointers to a RECORDCORE structure or structures that do not exist, the PMERR_INVALID_PARAMETERS error is set.

If the CMA_INVALIDATE attribute is specified, the container is repainted if the removed record or records are visible.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

CM_SCROLLWINDOW

This message scrolls an entire container window.

Parameters

param1

flScrollDirection *(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

fSuccess *(BOOL)*
Success indicator.

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

PMERR_INVALID_PARAMETERS.

Remarks

If the *lScrollInc* parameter value is greater than 0 and the CMA_HORIZONTAL attribute is specified, the container window is scrolled to the right. The container window is scrolled down if the *lScrollInc* parameter value is greater than 0 and the CMA_VERTICAL attribute is specified. Similarly, the container window is scrolled left and up, respectively, if the *lScrollInc* parameter value is less than 0 and the same two attributes are specified.

If you want the container window to be scrolled by an amount that is indicated with a key, such as the PgUp, PgDn, Home, and End keys, the application can send a key event to the scroll bar.

If the container window is displaying the split details view, the CM_HORZSCROLLSPLITWINDOW message is used for horizontal scrolling.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_SEARCHSTRING

This message returns the pointer to a container record whose text matches the string.

Parameters

param1

pSearchString *(PSEARCHSTRING)*
Pointer.

Pointer to the SEARCHSTRING structure. See SEARCHSTRING on page A-115 for definitions of this structure's fields as they apply to the CM_SEARCHSTRING message.

param2

pSearchAfter *(PRECORDCORE)*
Pointer.

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.

Pointer to the found container record.

NULL	No container record's text matches the search string.
Negative one	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.

Remarks

The CM_SEARCHSTRING message is NLS-enabled.

In the details view, the string is searched for in each column of each record.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_SETCNRINFO

This message sets or changes the data for the container control.

Parameters

param1

pCnrInfo *(PCNRINFO)*
Pointer.

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" on page 24-51 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).

Returns

fSuccess *(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.

Remarks

The data for a container is set from the buffer addressed by the *pCnrInfo* parameter. The flags in the *ulCnrInfoFl* parameter show which part or parts of the *pCnrInfo* parameter are set. The flag values can be combined by using a logical OR operator (|).

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_SETRECORDEMPHASIS

This message sets the emphasis attributes of the specified container record.

Parameters

param1

pRecord *(PRECORDCORE)*
Pointer.

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)*
Flag.

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.

Emphasis attribute of the container record. The following states can be combined by using a logical OR operator (|):

CRA_CURSORED
CRA_INUSE
CRA_SELECTED

Returns

fSuccess *(BOOL)*

Success indicator.

TRUE Successful completion.

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

- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY.

Remarks

For single-selection containers, the selection of the previous container record is cancelled before another record is selected. The selection cursor is set with the CRA_CURSORED attribute for single-selection containers. Only one selection cursor is allowed.

The selection cursor must always be available to the user. Therefore, if you attempt to disable the selection cursor by specifying FALSE for the *usChangeEmphasis* parameter and CRA_CURSORED for the *fEmphasisAttribute* parameter, the PMERR_INVALID_PARAMETERS error is set. In order to change the selection cursor attribute, TRUE should be specified for the *usChangeEmphasis* parameter and CRA_CURSORED for the *fEmphasisAttribute* parameter. The *pRecord* parameter should point to the record to which the selection cursor should be applied. The container control removes the selection cursor from the record with the cursor and applies it to the new record.

A CN_EMPHASIS notification code is sent to the container owner if the record emphasis attribute is changed.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_SORTRECORD

This message sorts the container records in the container control.

Parameters

param1

pfnCompare *(PFN)*

Pointer.

Pointer to a comparison function.

param2

pStorage *(PVOID)*

Application use.

Available for application use.

Returns

fSuccess *(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.

Remarks

The *pfnCompare* parameter must be declared as:

```
SHORT PFN pfnCompare(PRECORDCORE p1, PRECORDCORE p2, PVOID pStorage);
```

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 *pfnCompare* parameter points to an application-provided function that compares two RECORDCORE structures and returns a SHORT value that specifies their relationship. The *pfnCompare* parameter is called one or more times during the sorting process and is passed pointers to two RECORDCORE structures on each call. The routine must compare the RECORDCORE structures, and then return one of the following values:

Value	Meaning
Less than 0	*p1* is less than *p2*.
0	*p1* is equal to *p2*.
Greater than 0	*p1* is greater than *p2*.

The container records are sorted in increasing order, as defined by the *pfnCompare* parameter. The records can be sorted in reverse order by reversing the sense of "greater than" and "less than" in the *pfnCompare* parameter.

If the container has only one record, the PMERR_COMPARISON_FAILED error is set.

The application must provide an NLS-enabled function for the *pfnCompare* parameter. The container control does not provide NLS enablement for sorting.

An alternative to using the CM_SORTRECORD message is to provide an application-defined comparison function to sort the container records, which can be specified in the CNRINFO structure's *pSortRecord* field. If this function is provided, the container records are sorted as they are inserted into the container control. If this field is NULL, the records are not sorted on insertion.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

WM_PRESPARAMCHANGED (in Container Controls)

For the cause of this message, see "WM_PRESPARAMCHANGED" on page 12-48.

Parameters

param1

attrtype *(ULONG)*
Attribute type.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The application uses the WinSetPresParam function to change presentation parameters. This results in a WM_PRESPARAMCHANGED (in Container Controls) message being sent to the container.

Default Processing

For a description of the default processing, see "WM_PRESPARAMCHANGED" on page 12-48.

Chapter 25. Notebook Control Window Processing

This system-provided window procedure processes the actions on a notebook control (WC_NOTEBOOK).

Purpose

A notebook control (WC_NOTEBOOK window class) is a visual component whose specific purpose is to organize information on individual pages so that a user can find and display that information quickly and easily. It simulates a real-world notebook while improving it by overcoming its natural limitations. A user can select and display pages by using either a pointing device, such as a mouse, or the keyboard.

The notebook is designed to be customizable to meet varying application requirements, while providing an easy-to-use user interface component that can be used 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 Control Styles

Notebook control window styles can be set with a notebook is created. The following styles can be set when creating a notebook control window. If no styles are specified, defaults, which are identified in the following descriptions, are used.

- Specify one of the following to determine whether the notebook has a solid or spiral binding:

  BKS_SOLIDBIND
  Paints a solid binding on the notebook. This is the default.

  BKS_SPIRALBIND
  Paints a spiral binding on the notebook.

- Specify one of the following to determine where the back pages are positioned:

  BKS_BACKPAGESBR
  Paints back pages on the notebook's bottom and right sides. This is the default.

  BKS_BACKPAGESBL
  Paints back pages on the notebook's bottom and left sides.

  BKS_BACKPAGESTR
  Paints back pages on the notebook's top and right sides.

  BKS_BACKPAGESTL
  Paints back pages on the notebook's top and left sides.

- Specify one of the following to determine the side of the notebook on which the major tabs are positioned. Valid combinations with back pages styles are noted in each definition.

  BKS_MAJORTABRIGHT
  Places major tabs on the notebook's right edge. Only valid in combination with BKS_BACKPAGESBR or BKS_BACKPAGESTR. This is the default when either of these back pages styles is used.

  BKS_MAJORTABLEFT
  Places major tabs on the notebook's left edge. Only valid in combination with BKS_BACKPAGESBL or BKS_BACKPAGESTL. This is the default when BKS_BACKPAGESTL is used.

* Trademark of IBM Corporation

BKS_MAJORTABTOP
Places major tabs on the notebook's top edge. Only valid in combination with BKS_BACKPAGESTR or BKS_BACKPAGESTL.

BKS_MAJORTABBOTTOM
Places major tabs on the notebook's bottom edge. Only valid in combination with BKS_BACKPAGESBR or BKS_BACKPAGESBL. This is the default when BKS_BACKPAGESBL is used.

- Specify one of the following to set the shape of the notebook tabs:

BKS_SQUARETABS
Draws tabs with square edges. This is the default.

BKS_ROUNDEDTABS
Draws tabs with rounded edges.

BKS_POLYGONTABS
Draws tabs with polygon edges.

- Specify one of the following to position the status line text:

BKS_STATUSTEXTLEFT
Left-justifies status line text. This is the default.

BKS_STATUSTEXTRIGHT
Right-justifies status line text.

BKS_STATUSTEXTCENTER
Centers status line text.

- Specify one of the following to position the tab text:

BKS_TABTEXTCENTER
Centers tab text. This is the default.

BKS_TABTEXTLEFT
Left-justifies tab text.

BKS_TABTEXTRIGHT
Right-justifies tab text.

Notebook Control Data

See the following for descriptions of the notebook control data structures:

- BOOKTEXT on page A-9
- DELETENOTIFY on page A-24
- PAGESELECTNOTIFY on page A-78.

Notebook Control Notification Messages

These messages are initiated by the notebook control window to notify its owner of significant events.

WM_CONTROL (in Notebook Controls)

For the cause of this message, see "WM_CONTROL" on page 12-28.

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_PAGESELECTED	Indicates a new page has been brought to the top of the notebook.
BKN_PAGEDELETED	Indicates a page has been deleted from the notebook.

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, 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

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The notebook control window procedure generates this message and sends it to its owner, informing the owner of this event.

Default Processing

For a description of the default processing, see "WM_CONTROL" on page 12-28.

Notebook Control Window Messages

This section describes the notebook control window procedure actions on receiving the following messages.

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.

Points 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

fSuccess *(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.

Remarks

The application can use this message to determine the size of either the notebook window or the application page window. It can also be used when the application handles the position and size of the application page window.

To calculate the application page rectangle, specify the coordinates of the notebook window in the *pRectl* parameter and TRUE in the *bPage* parameter. The notebook control then uses the coordinates specified in the *pRectl* parameter to calculate and return the coordinates of the application page window.

To calculate the notebook rectangle, specify the coordinates of the application page window in the *pRectl* parameter and FALSE in the *bPage* parameter. The notebook control then uses the coordinates specified in the *pRectl* parameter to calculate and return the coordinates of the notebook window.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

fSuccess *(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.

Remarks

The notebook frees all storage that it has allocated for the deleted page or pages. The application is responsible for deleting the application page window and bit map, if created.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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.

Identifier for the inserted page.

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.

Remarks

The notebook control allocates and manages the storage needed for the new page. If neither the BKA_MAJOR or BKA_MINOR attribute is specified, the page is inserted with no tab attributes.

If the application does not specify the BKA_AUTOPAGESIZE attribute, it must handle the positioning and sizing of the application page window when it receives the BKN_NEWPAGESIZE notification code.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

BKM_INVALIDATETABS

This message repaints all of the tabs in the notebook.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fSuccess *(BOOL)*
Success indicator.

TRUE Tabs painted successfully.
FALSE Tabs were not painted.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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.

Number of pages in the notebook.

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.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

ulPageData *(ULONG)*
Page data.

Application-defined 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.

Remarks

This data is set by using the BKM_SETPAGEDATA message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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.

Retrieved page identifier.

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.

Remarks

If the BKA_FIRST, BKA_LAST, or BKA_TOP attribute is specified, the page ID in the *ulPageId* parameter is ignored.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

usPageStyle *(USHORT)*
Page style data.

BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the *ulPageId* parameter.
Other	Page style data.

Remarks

This style data is set when the page is inserted, which is done by using the BKM_INSERTPAGE message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

hwndPage *(HWND)*
Window handle.

Handle of the application page window associated with the specified page identifier.

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.

Remarks

The application page window handle is set by using the BKM_SETPAGEWINDOWHWND message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULLHANDLE.

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.

Pointer to a BOOKTEXT data structure. See BOOKTEXT on page A-9 for definitions of this structure's fields as they apply to the BKM_QUERYSTATUSLINETEXT message.

Returns

statusTextLen *(USHORT)*
String length.

Length of the status line 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_SETSTATUSLINETEXT) for the page specified in the *ulPageId* parameter.
Other	Length of the returned status line text string.

Remarks

The size of the status line text string can be queried by specifying 0 for the *textLen* field of the BOOKTEXT data structure. In this way, the application can determine the size of the buffer needed to store the status line text string. The null character at the end of the text string is not included in the returned length.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

hbm *(HBITMAP)*
Bit-map handle.

Handle of the bit map associated with the specified page identifier.

BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the *ulPageId* parameter.
NULLHANDLE	No bit-map handle is associated with the page specified in the *ulPageId* parameter.
Other	Handle of the bit map associated with the specified page identifier.

Remarks

The tab bit-map handle is set by using the BKM_SETTABBITMAP message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULLHANDLE.

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.

Pointer to a BOOKTEXT data structure. See BOOKTEXT on page A-9 for definitions of this structure's fields as they apply to the BKM_QUERYTABTEXT message.

Returns

tabTextLen *(USHORT)*
String length.

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.

Remarks

The size of the tab text string can be queried by specifying 0 for the *textLen* field in the BOOKTEXT data structure. In this way, the application can determine the size of the buffer needed to store the tab text string. The null character at the end of the text string is not included in the returned length.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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_MAJOR
- BKA_MINOR
- BKA_PAGEBUTTON.

Returns

fSuccess *(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.

Remarks

If either the BKA_MAJORTAB or BKA_MINORTAB attribute is specified for the *usType* parameter, the minimum width and height for display is 7 pels to allow space for the tab border and the selection cursor. If the tabs or page buttons are not to be displayed, the height and width can be set to 0.

If the new dimensions cause the notebook size to change, the notebook sends a BKN_NEWPAGESIZE notification code to the application.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

fSuccess *(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.

Remarks

The notebook background, border, selection cursor, and status line text colors are mapped to system presentation attributes. See "WM_PRESPARAMCHANGED (in Notebook Controls)" on page 25-21 for information about setting the color of these regions.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

fSuccess *(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.

Remarks

This data can be queried by using the BKM_QUERYPAGEDATA message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

fSuccess *(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.

Remarks

The notebook shows the application page window specified in the *hwndPage* parameter whenever the notebook page specified in the *ulPageId* parameter is brought to the top of the notebook. If the BKA_AUTOPAGESIZE attribute is specified when that page is inserted into the notebook, the notebook also handles the sizing and positioning of the application page window.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

pszString *(PSZ)*
Pointer.

Pointer to a text string that ends in a null character.

Returns

fSuccess *(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.

Remarks

If the text is longer that the status area length, only the text that fits in the status area is displayed.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

fSuccess *(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.

Remarks

When displayed, the bit map is stretched to fit the size of the tab. If a tab has rounded or polygonal edges, the bit map is sized to fit the rectangular area of the tab, as shown in Figure 25-1.

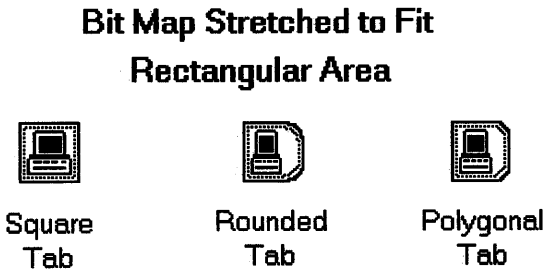


Figure 25-1. Tabs Showing Rectangular Area Used to Size a Bit Map

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

pszString *(PSZ)*
Pointer.

Pointer to a text string that ends with a null character.

Returns

fSuccess *(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.

Remarks

The text is centered from the tab edges.

The application can define a mnemonic key when sending this message by placing a tilde (~) character before the character that is to be the mnemonic key. The notebook brings this page to the top whenever the user presses the mnemonic key.

The mnemonic key processing is not case-sensitive, so the user can type the mnemonic character in either upper or lower case.

The application can remove or change the mnemonic key by sending additional BKM_SETTABTEXT messages for the specified page.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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 *(ULONG)*
Reserved.

0 Reserved value, 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.

Remarks

The application receives a BKN_PAGESELECTED notification code when the new page is brought to the top of the notebook.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

WM_CHAR (in Notebook Controls)

For the cause of this message, see "WM_CHAR" on page 12-24.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-24.

Remarks

If the application page window has the focus, the notebook will handle the following keyboard interaction:

Keyboard Input	Description
Alt+Up Arrow	Sets the focus to the notebook window.

If the notebook control has the focus, it handles the following keyboard interactions:

Keyboard Input	Description
Alt+Down Arrow	Sets the focus to the application page window.
Down Arrow or Right Arrow	Moves the selection cursor to the next major or minor tab. If either of these keys is pressed while the selection cursor is on a major tab, the cursor moves to the next major tab. If either of these keys is pressed while the selection cursor is on a minor tab, the cursor moves to the next minor tab. If the next tab is not visible, the tabs are scrolled to bring the next tab into view. If the end of the tabs is reached, scrolling ends.

Up Arrow or Left Arrow
Moves the selection cursor to the previous major or minor tab. If either of these keys is pressed while the selection cursor is on a major tab, the cursor moves to the previous major tab. If either of these keys is pressed while the selection cursor is on a minor tab, the cursor moves to the previous minor tab. If the previous tab is not visible, the tabs are scrolled to bring the previous tab into view. If the beginning of the tabs is reached, scrolling ends.

Tab Moves the selection cursor to the next tab position or control.

Ctrl+Tab Moves the selection cursor to the next control.

Shift+Tab Moves the selection cursor to the previous tab position or control.

Enter or Spacebar The cursored tab page becomes the top page of the notebook.

Mnemonics Mnemonic key definition is provided by using the BKM_SETTABTEXT message. Coding a mnemonic character (~) before a text character in the BKM_SETTABTEXT message causes that character to be underlined in the tab's text string and activates it as a mnemonic selection character. The notebook control brings the page whose tab contains the mnemonic character to the top whenever the user presses the mnemonic key. The mnemonic key pressing is not case-sensitive, so the user can type the mnemonic character in either upper or lower case.

PgDn or Alt + PgDn Brings the next page to the top of the notebook and sets the selection cursor on the associated tab, if there is a new one.

PgUp or Alt + PgUp Brings the previous page to the top of the notebook and sets the selection cursor on the associated tab, if there is a new one.

Home Brings the first page of the notebook to the top and sets the selection cursor on the associated tab, if there is a new one.

End Brings the last page of the notebook to the top and sets the selection cursor on the associated tab, if there is a new one.

Default Processing
For a description of the default processing, see "WM_CHAR" on page 12-24.

WM_CONTROLPOINTER (in Notebook Controls)
For the cause of this message, see "WM_CONTROLPOINTER" on page 12-29.

Parameters
For a description of the parameters, see "WM_CONTROLPOINTER" on page 12-29.

Remarks
For the appropriate remarks, see "WM_CONTROLPOINTER" on page 12-29.

Default Processing
For the default processing, see "WM_CONTROLPOINTER" on page 12-29.

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

ownerItem *(POWNERITEM)*
Pointer.

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 A-76 for the default field values.

hwnd *(HWND)*
Notebook window handle.

hps *(HPS)*
Presentation-space handle.

state *(USHORT)*
Notebook window style flags. See "Notebook Control Styles" on page 25-1 for descriptions of these style flags.

attribute *(USHORT)*
Page attribute flags for the tab page. See "BKM_INSERTPAGE" on page 25-6 for descriptions of these attribute flags.

stateOld *(USHORT)*
Reserved.

attributeOld *(USHORT)*
Reserved.

itemRectangle *(RECTL)*
Tab rectangle to be drawn in window coordinates.

identity *(SHORT)*
Reserved.

item *(ULONG)*
Current page ID (**ulPageId**) for which the content of a tab is to be drawn.

Returns

reply

drawn *(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.

Remarks

If an application uses notebook controls that contain tab pages, the default condition is for the application to draw the contents of the tab each time a tab page is displayed. This situation applies particularly if the content of the tab is not one of the supported formats.

The notebook control window procedure generates this message and sends it to its owner, informing the owner that the content of a tab is to be drawn. The owner is given the opportunity to draw the

content of the tab and to indicate that the content of the tab has been drawn or that the notebook control is to draw it. To indicate that the notebook control is to draw the content of the tab, the owner sends either a BKM_SETTABTEXT or a BKM_SETTABBITMAP message to the notebook control.

Default Processing

For a description of the default processing, see "WM_DRAWITEM" on page 12-31.

WM_PRESPARAMCHANGED (in Notebook Controls)

For the cause of this message, see "WM_PRESPARAMCHANGED" on page 12-48.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The application uses this message to notify the notebook that a given inherited presentation parameter has changed.

Default Processing

For a description of the default processing, see "WM_PRESPARAMCHANGED" on page 12-48.

WM_SIZE (in Notebook Controls)

For the cause of this message, see "WM_SIZE" on page 12-61.

Parameters

For a description of the parameters, see "WM_SIZE" on page 12-61.

Remarks

When the size of the notebook window changes, all of the regions are recalculated. The notebook sends a BKN_NEWPAGESIZE notification code to the application. The notebook sets the position and size of application page windows that are associated with pages for whom the BKA_AUTOPAGESIZE attribute is set.

Default Processing

For a description of the default processing, see "WM_SIZE" on page 12-61.

Chapter 26. Slider Control Window Processing

This system-provided window procedure processes the actions on a slider control (WC_SLIDER).

Purpose

A slider control (WC_SLIDER window class) is a visual component whose specific purpose is to allow a user to set, display, or modify a value by moving a slider arm along a slider shaft. Sliders are typically used to allow a user to easily set values that have familiar increments, such as feet, inches, degrees, decibels, and so forth.

However, they can also be used for other purposes when immediate feedback is necessary, such as to blend colors or to show the percentage of a task that has completed. For example, an application might allow a user to mix and match color shades by moving a slider arm, or a read-only slider could be provided that shows how much of a task has completed by filling in the slider shaft as the task progresses. These are just a few examples to show you the many ways in which sliders can be used.

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

The slider is designed to be customizable to meet varying application requirements, while providing an easy-to-use user interface component that can be used to develop products that conform to the 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*.

Slider Control Styles

Slider control window styles are set when a slider window is created. The following styles can be set when creating a slider control window. If no styles are specified, defaults, which are identified in the following descriptions, are used.

- Specify either of the following to determine the slider's orientation:

  SLS_HORIZONTAL
  The slider is positioned 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_VERTICAL
  The slider is positioned 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.

- Specify one of the following to position the slider within the slider window:

  SLS_CENTER
  The slider is centered in the slider window. This is the default positioning of the slider.

  SLS_BOTTOM
  The slider is positioned at the bottom of the slider window. This is valid for horizontal sliders only.

  SLS_TOP
  The slider is positioned at the top of the slider window. This is valid for horizontal sliders only.

SLS_LEFT
The slider is positioned at the left edge of the slider window. This is valid for vertical sliders only.

SLS_RIGHT
The slider is positioned at the right edge of the slider window. This is valid for vertical sliders only.

- Specify one of the following to determine the location of the scale on the slider shaft:

SLS_PRIMARYSCALE1
The slider uses 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. This is the default for a slider.

SLS_PRIMARYSCALE2
The slider uses the 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.

- Specify one of the following to determine the slider arm's home position:

SLS_HOMELEFT
The slider uses the left edge of the slider as the base value for incrementing. This is the default for horizontal sliders and is valid for horizontal sliders only.

SLS_HOMERIGHT
The slider uses the right edge of the slider as the base value for incrementing. This is valid for horizontal sliders only.

SLS_HOMEBOTTOM
The slider uses the bottom of the slider as the base value for incrementing. This is the default for vertical sliders and is valid for vertical sliders only.

SLS_HOMETOP
The slider uses the top of the slider as the base value for incrementing. This is valid for vertical sliders only.

- Specify one of the following to determine the location of the slider buttons. If you do not specify one of these styles, or if conflicting styles are specified, slider buttons are not included in the slider control.

SLS_BUTTONSLEFT
The slider includes incremental slider buttons with the control and places them to the left of the slider shaft. These slider buttons move the slider arm by one position, either left or right, in the direction that is selected. This is valid for horizontal sliders only.

SLS_BUTTONSRIGHT
The slider includes incremental slider buttons with the control and places them to the right of the slider shaft. These slider buttons move the slider arm by one position, either left or right, in the direction that is selected. This is valid for horizontal sliders only.

SLS_BUTTONSBOTTOM
The slider includes incremental slider buttons with the control and places them at the bottom of the slider shaft. These slider buttons move the slider arm by one position, either up or down, in the direction that is selected. This is valid for vertical sliders only.

SLS_BUTTONSTOP
The slider includes incremental slider buttons with the control and places them at the top of the slider shaft. These slider buttons move the slider arm by one position, either up or down, in the direction that is selected. This is valid for vertical sliders only.

- Other styles that you can specify:

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

SLS_READONLY

The slider is created as a read-only slider. This means that the user cannot interact with the slider. It is used merely as a mechanism 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.

SLS_RIBBONSTRIP

As the slider arm moves, the slider fills the slider shaft between the home position and the slider arm with a color value that is different from the slider shaft color, similar to the mercury in a thermometer.

SLS_OWNERDRAW

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

Slider Control Data

See SLDCDATA on page A-116.

Slider Control Notification Messages

These messages are initiated by the slider control window to notify its owner of significant events.

WM_CONTROL (in Slider Controls)

For the cause of this message, see "WM_CONTROL" on page 12-28.

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

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The slider control window procedure generates this message and sends it to its owner, informing the owner of this event.

Default Processing

For a description of the default processing, see "WM_CONTROL" on page 12-28.

WM_CONTROLPOINTER (in Slider Controls)

For the cause of this message, see "WM_CONTROLPOINTER" on page 12-29.

Parameters

For a description of the parameters, see "WM_CONTROLPOINTER" on page 12-29.

Remarks

For the appropriate remarks, see "WM_CONTROLPOINTER" on page 12-29.

Default Processing

For the default processing, see "WM_CONTROLPOINTER" on page 12-29.

WM_DRAWITEM (in Slider Controls)

If the SLS_OWNERDRAW style bit is set for a slider control, this notification message is sent to that slider control's owner whenever the slider shaft, ribbon strip, slider arm, and slider background are to be drawn.

Parameters

param1

id *(USHORT)*
Window identifier.

The window identifier of the slider control sending this notification message.

param2

owneritem *(POWNERITEM)*
Pointer.

Pointer to an OWNERITEM data structure. The following list defines the OWNERITEM data structure fields that apply to the slider control. See OWNERITEM on page A-76 for the default field values.

hwnd *(HWND)*
Slider window handle.

hps *(HPS)*
Presentation-space handle.

state *(USHORT)*
Slider window style flags. See "Slider Control Styles" on page 26-1 for descriptions of these style flags.

attribute *(USHORT)*
Reserved.

stateold *(USHORT)*
Reserved.

attributeold *(USHORT)*
Reserved.

itemrectangle *(RECTL)*
Item rectangle to be drawn in window coordinates.

identity *(SHORT)*
Identity of item to be drawn:

SDA_SLIDERSHAFT
Specifies that the slider shaft is to be drawn.

SDA_RIBBONSTRIP
Specifies that the slider shaft area that contains a ribbon strip is to be drawn.

SDA_SLIDERARM
Specifies that the slider arm is to be drawn.

SDA_BACKGROUND
Specifies that the slider background is to be drawn.

item *(ULONG)*
Reserved.

Returns

reply

drawn *(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 slider control draws the item.

Remarks

The slider control provides this message to give the application the opportunity to provide a custom slider shaft, custom ribbon strip, custom slider arm, and custom background. The application can specify one or all of these items and is given the opportunity to do so.

The slider control window procedure generates this message and sends it to its owner, informing the owner that an item is to be drawn. The owner is then given the opportunity to draw that item, and to indicate that an item has been drawn or that the slider control is to draw it.

Default Processing

For a description of the default processing, see "WM_DRAWITEM" on page 12-31.

Slider Control Window Messages

This section describes the slider control window procedure actions on receiving the following messages.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

ulDetentId *(ULONG)*
Detent ID.

Unique identifier for the detent being added to the slider. If 0 is returned, an error occurred. The WinGetLastError function may return the following errors:

- PMERR_HEAP_MAX_SIZE_REACHED
- PMERR_PARAMETER_OUT_OF_RANGE.

Remarks

The application uses this message to add detents along the slider to denote values that do not fall along an increment setting. An example of this would be a slider that represents temperature and has increments that are on multiples of 5. A detent could be located at 32, instead of 30 or 35, for special purposes.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

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.

Remarks

An application could use this message to place text above the detent or position an item relative to it.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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

pszTickText *(PSZ)*
Pointer.

Pointer to the buffer into which to place the text string for the tick mark.

Returns

sTextLen *(SHORT)*
Count of bytes.

Count of bytes copied to buffer.

> = 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.

Remarks

This message could be used to return text that represents the current position of the slider arm or to query the text for use in ownerdraw mode.

By specifying 0 as the value of the *usBufLen* parameter and then looking at the value returned in the *sTextLen* parameter, an application can determine the size of the buffer to allocate for copying the text. An application can then allocate a buffer of this size, adding one byte for the null termination character, and then specify this buffer and size on the query call.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

ulInfo *(ULONG)*
Return information.

One of the following items, depending on which SMA_* message attribute or attributes were set with the SLM_SETSLIDERINFO message.

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

  usShaftLength *(USHORT)*
  Length of the slider shaft, in pixels. It is the width of the slider shaft for horizontal sliders, and the height of the slider shaft for vertical sliders.

usShaftBreadth *(USHORT)*
Breadth of the slider shaft, in pixels. It is the height of the slider shaft for horizontal sliders, and the width of the slider shaft for vertical sliders.

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

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

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

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

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

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

- If the SMA_SLIDERARMPOSITION and SMA_RANGEVALUE attributes are set, the following is returned:

usArmPos *(USHORT)*
Number of pixels from the home position to the slider arm.

usSliderRange *(USHORT)*
Number of pixels over which the user could select a value on the slider.

- If the SMA_SLIDERARMPOSITION and SMA_INCREMENTVALUE attributes are set, the following is returned:

usIncrementPos *(USHORT)*
Increment that corresponds to the current position of the slider arm.

- If the SLDERR_INVALID_PARAMETERS error is returned, an error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

Remarks

The application uses this message to query for information about individual parts of a slider control, or the value selected by a user.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

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.

Remarks

This message could be used to get the position of a tick mark along the slider for use in ownerdraw mode if, for example, you want to place something other than text, such as bit maps or icons, above the tick marks.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

usTickSize *(USHORT)*
Tick mark length.

Specifies the length of the tick mark at the position queried, in pixels. If this value is 0, the tick mark is invisible.

If the SLDERR_INVALID_PARAMETERS error is returned, an error occurred. The WinGetLastError function may return the following error:

PMERR_PARAMETER_OUT_OF_RANGE.

Remarks

The application uses this message to query a scale along the slider to indicate what tick marks, tick mark sizes, or both are currently set for the slider.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fSuccess *(BOOL)*
Success indicator.

TRUE Detent was successfully removed.
FALSE An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

Remarks

The application uses this message to remove detents added previously to the slider to denote values that do not fall along an increment setting.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

pszTickText *(PSZ)*
Pointer.

Pointer to the text that is to be drawn at the position specified. If this value is NULL, no text is drawn.

Returns

fSuccess *(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.

Remarks

The application uses this message to draw text along the increments of the slider to clarify the magnitude of the range. This text could show the exact value for that tick mark, or could be a general remark, such as low, high, and so forth.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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

fSuccess *(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.

Remarks

The application uses this message to customize the slider for a specific use. In setting the shaft dimensions, only the breadth of the slider can be set. The length of the shaft is always determined by the number of increments and the spacing between increments, both of which are set for the primary scale when the slider is created.

Positioning of the shaft within the slider window could be used by applications that cannot use the default positioning provided by the slider control.

Setting of the slider arm dimensions could be used by applications that need a larger slider arm, such as touch screen applications.

Setting the slider arm position can be used to:

- Set the initial value of the slider before it becomes visible
- Change the value when it is tied to another control, such as an entry field
- Show the value of a quantity when the slider is being used to monitor an event, such as a read-only slider being used as a progress indicator.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fSuccess *(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.

Remarks

The application uses this message to draw a scale along the slider to indicate value positions in relation to the slider arm. The application can set varying lengths for different increments of the slider to help the user understand the magnitude of the value being set.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

WM_CHAR (in Slider Controls)

For the cause of this message, see "WM_CHAR" on page 12-24.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-24.

Remarks

The slider control window procedure responds to this message by sending it to its owner if it has not processed the key stroke. This is the most common means by which the input focus is switched around the various controls in a dialog box.

The keystrokes processed by a slider control are:

Down Arrow	Moves the slider arm down one increment. When the slider arm reaches the bottom of the slider shaft or when a horizontal slider is being used, the Down Arrow key has no effect.
Up Arrow	Moves the slider arm up one increment. When the slider arm reaches the top of the slider shaft or when a horizontal slider is being used, the Up Arrow key has no effect.
Left Arrow	Moves the slider arm left one increment. When the slider arm reaches the leftmost edge or when a vertical slider is being used, the Left Arrow key has no effect.
Right Arrow	Moves the slider arm right one increment. When the slider arm reaches the rightmost edge or when a vertical slider is being used, the Right Arrow key has no effect.
Shift + Down Arrow	Moves the slider arm to the next detent below the current position. If there are no more detents or if a horizontal slider is being used, the Shift + Down Arrow key combination has no effect.
Shift + Up Arrow	Moves the slider arm to the next detent above the current position. If there are no more detents or if a horizontal slider is being used, the Shift + Up Arrow key combination has no effect.
Shift + Left Arrow	Moves the slider arm to the next detent left of the current position. If there are no more detents or if a vertical slider is being used, the Shift + Left Arrow key combination has no effect.
Shift + Right Arrow	Moves the slider arm to the next detent right of the current position. If there are no more detents or if a vertical slider is being used, the Shift + Right Arrow key combination has no effect.
Home, Ctrl + Home	Moves the slider arm to the home position of the slider. Pressing the Home key or the Ctrl + Home key combination when the slider arm is at the home position has no effect. The default home position for a slider is the leftmost edge for horizontal sliders and the bottom edge for vertical sliders.
End, Ctrl + End	Moves the slider arm to the end position of the slider. Pressing the End key or the Ctrl + End key combination when the slider arm is at the end position has no effect. The default end position for a slider is the rightmost edge for horizontal sliders and the top edge for vertical sliders.

Default Processing

For a description of the default processing, see "WM_CHAR" on page 12-24.

WM_PRESPARAMCHANGED (in Slider Controls)

For the cause of this message, see "WM_PRESPARAMCHANGED" on page 12-48.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*
Reserved value.

0 Reserved value; must be 0.

Remarks

The application uses this message to notify the slider that a given inherited presentation parameter has changed.

Default Processing

For a description of the default processing, see "WM_PRESPARAMCHANGED" on page 12-48.

WM_QUERYWINDOWPARAMS (in Slider Controls)

For the cause of this message, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Parameters

param1

wndparams *(PWNDPARAMS)*
Pointer.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

result *(BOOL)*
Success indicator.

TRUE Successful completion.
FALSE Error occurred.

Remarks

The slider control window procedure responds to this message by returning the information in the buffer provided. If this message is sent to a slider window of another process, the information in, or identified by, the value of the *wndparams* parameter must be in memory shared by both processes.

Default Processing

For a description of the default processing, see "WM_QUERYWINDOWPARAMS" on page 12-53.

WM_SETWINDOWPARAMS (in Slider Controls)

For the cause of this message, see "WM_SETWINDOWPARAMS" on page 12-60.

Parameters

param1

wndparams *(PWNDPARAMS)*
Pointer.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

result *(BOOL)*
Success indicator.

TRUE Successful operation.
FALSE Error occurred.

Remarks

If this message is sent to a slider window of another process, the information in, or identified by, the value of the *wndparams* parameter must be in memory shared by both processes.

Default Processing

For a description of the default processing, see "WM_SETWINDOWPARAMS" on page 12-60.

Chapter 27. Value Set Control Window Processing

This system-provided window procedure processes the actions on a value set control (WC_VALUESET).

Purpose

Like radio buttons, a value set control (WC_VALUESET window class) is a visual component whose specific purpose is to allow a user to select one choice from a group of mutually exclusive choices. However, unlike radio buttons, a value set can use graphical images (bit maps or icons), as well as colors, text, and numbers, to represent the items that a user can select.

Even though text is supported, a value set's primary purpose is to display choices as graphical images. By using graphical images in a value set, you can preserve space on the display screen. You can also allow the user to see exactly what is being selected instead of having to rely on descriptions of the choices. This allows a user to make a selection faster than if the user had to read a description of each choice. For example, if you want to allow a user to choose from a variety of patterns, you can present those patterns as value set choices instead of having to provide a list of radio buttons with description of each pattern.

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

The value set is designed to be customizable to meet varying application requirements, while providing an easy-to-use user interface component that can be used 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 Control 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 27-1 provides an example of a value set with bit maps.

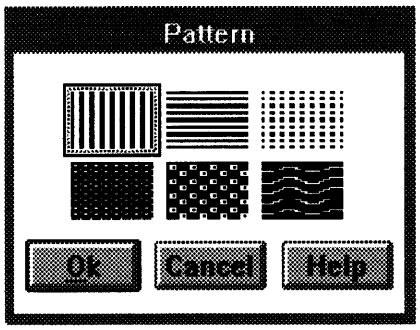


Figure 27-1. Value Set with Bit Maps

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 27-2 on page 27-2 provides an example of a value set with icons.

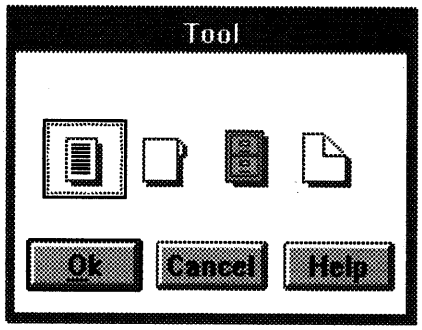


Figure 27-2. Value Set with Icons

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 27-3 provides an example of a value set with text strings.

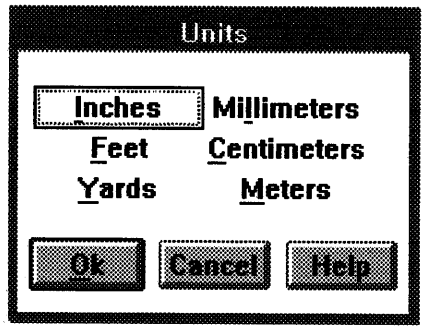


Figure 27-3. Value Set with Text Strings

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 27-4 on page 27-3 provides an example of a value set with colors.

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 27-4 on page 27-3 provides an example of a value set with colors.

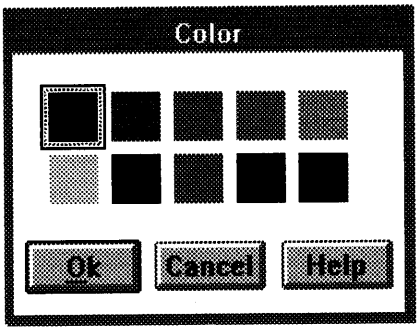


Figure 27-4. Value Set with Colors

- 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 27-5 provides an example of a value set with a border.

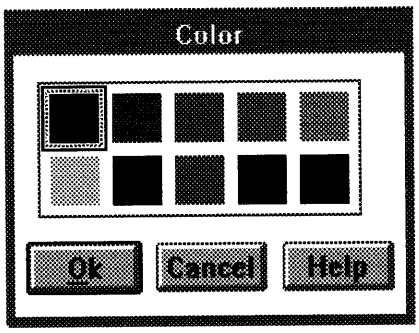


Figure 27-5. Value Set with Border

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 27-6 provides an example of a value set with item borders.

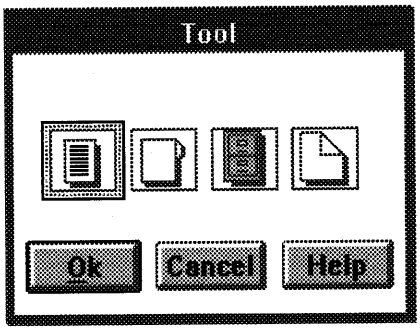


Figure 27-6. Value Set with Item Borders

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.

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

Value Set Control Data

For information on value set control data, see the following:

- VSCDATA on page A-123
- VSDRAGINFO on page A-123
- VSDRAGINIT on page A-124
- VSTEXT on page A-124.

Value Set Control Notification Messages

These messages are initiated by the value set control window to notify its owner of significant events.

WM_CONTROL (in Value Set Controls)

For the cause of this message, see "WM_CONTROL" on page 12-28.

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

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The value set control window procedure generates this message and sends it to its owner, informing the owner of this event.

Default Processing

For a description of the default processing, see "WM_CONTROL" on page 12-28.

WM_CONTROLPOINTER (in Value Set Controls)

For the cause of this message, see "WM_CONTROLPOINTER" on page 12-29.

Parameters

For a description of the parameters, see "WM_CONTROLPOINTER" on page 12-29.

Remarks

For the appropriate remarks, see "WM_CONTROLPOINTER" on page 12-29.

Default Processing

For the default processing, see "WM_CONTROLPOINTER" on page 12-29.

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

owneritem *(POWNERITEM)*
Pointer.

Pointer to an OWNERITEM data structure. The following list defines the OWNERITEM data structure fields that apply to the value set control. See OWNERITEM on page A-76 for the default field values.

hwnd *(HWND)*
Value set window handle.

hps *(HPS)*
Presentation-space handle.

state *(USHORT)*
Value set window style flags. See "Value Set Control Styles" on page 27-1 for descriptions of these style flags.

attribute *(USHORT)*
Item attribute flags for the indexed item. See "VM_SETITEMATTR" on page 27-14 for descriptions of these attribute flags.

stateold *(USHORT)*
Reserved.

attributeold *(USHORT)*
Reserved.

itemrectangle *(RECTL)*
Item rectangle to be drawn in window coordinates.

identity *(SHORT)*
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.

item *(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

reply

drawn *(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.

Remarks

The value set control draws only items that are represented in one of the formats described: text, color, bit maps, or icons.

If an application uses value set controls that contain items that are not represented by the supported formats or requires that the emphasized attribute of an item is to be drawn in a special manner, the application must specify those items as VIA_OWNERDRAW and those items must be drawn by the owner.

Through this message, the application can provide a custom value set background (the area between the items) and customize the area surrounding the value set (the area on the top and right sides of the value set that is left over when the value set calculates its size). The application can specify how either or both of these areas are drawn and is given the opportunity to do so.

The value set control window procedure generates this message and sends it to its owner, informing the owner that something is to be drawn. The owner is given the opportunity to draw and to indicate whether the value set control should continue with the normal drawing of that component.

Default Processing

For a description of the default processing, see "WM_DRAWITEM" on page 12-31.

Value Set Control Window Messages

This section describes the value set control window procedure actions on receiving the following messages.

VM_QUERYITEM

This message queries the contents of the item indicated by the values of the *usRow* and *usColumn* parameters. 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.

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* parameter. For all other attributes, the *param2* parameter is reserved and should be set to a NULL value.

See VSTEXT on page A-124 for definitions of this structure's fields as they apply to the VM_QUERYITEM message.

Returns

ulItemId *(ULONG)*
Item information.

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

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

  usTextLen *(USHORT)*
  Number of bytes copied to the buffer. This is the length of the text string, excluding the null termination character.

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

  hbmItem *(HBITMAP)*
  Handle of the bit map associated with the item indexed by the *param1* parameter. If the item is empty, a NULL value is returned.

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

  hptItem *(HPOINTER)*
  Handle of the icon associated with the item indexed by the *param1* parameter. If the item is empty, a NULL value is returned.

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

  rgbItem *(ULONG)*
  Color value associated with the item indexed by the *param1* parameter. If the item is empty, a NULL value is returned. 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 set, the following is returned:

  ulColorIndex *(ULONG)*
  Index of the color associated with the item indexed by the *param1* parameter.

The following is returned for any of the items to indicate an error condition:

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

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

Remarks

The application uses this message to query the contents of an individual value set item. When querying a text item, the application must provide a buffer for returning the text information. By specifying 0 as the value of the **usBufLen** field and then getting the value returned in the **usTextLen** parameter, an application can determine how large this buffer must be. The value returned is the length of the text string, excluding the null termination character.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

VM_QUERYITEMATTR

This message queries 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 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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

usItemAttr *(USHORT)*
Item information.

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

- 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.

- The following is returned if an error occurs:

  VMERR_INVALID_PARAMETERS
  The WinGetLastError function may return the following errors:

  - PMERR_INVALID_PARAMETERS
  - PMERR_PARAMETER_OUT_OF_RANGE.

Remarks

The application uses this message to query the specific attribute or attributes of a value set item.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 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.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

VM_QUERYSELECTEDITEM

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

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

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.

Remarks

The application uses this message to query the index of the currently selected value set item. If 0 is returned, no item is selected.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fSuccess *(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.

Remarks

The application uses this message to select the specified value set item.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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* parameters. 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* parameter 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* parameter 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* parameter 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* parameter 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* parameter 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

fSuccess *(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.

Remarks

The application uses this message to set the contents of an individual value set item. To set the values for the entire value set, an application would loop through the rows and columns, setting the value of each item during the initial value set window processing before the window becomes visible.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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. If 0 is passed, the specified attribute or attributes are either set or reset for all of the rows in the value set.

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. If 0 is passed, the specified attribute or attributes are either set or reset for all of the columns in the value set.

param2

usItemAttr *(USHORT)*
Item attributes.

Attribute or attributes of the item to be set or reset based on the value of the *fSet* parameter. 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

fSuccess *(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.

Remarks

The application uses this message to either set or reset a specific attribute or attributes of a value set item. This provides customization of a control at the item level, so that applications can provide their own types of items with a value set, as well as perform direct manipulation and other actions.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

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* parameter 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* parameter 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

fSuccess *(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.

Remarks

Upon receiving this message, the value set redraws the control with the new width, height, and spacing specifications for each item. Any items that do not fit within the current window size are clipped.

When the value set control receives a WM_SIZE (in Value Set Controls) message, which is sent when the value set window is resized, the value set control defaults the size of each item by dynamically dividing the window size by the number of rows and columns. It allows enough room for the border, selection cursor, and selection emphasis, and defaults the spacing between items to 0. To override these default settings, the application must resend the VM_SETMETRICS message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

WM_CHAR (in Value Set Controls)

For the cause of this message, see "WM_CHAR" on page 12-24.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-24.

Remarks

The value set control window procedure responds to this message by sending it to its owner if it has not processed the key stroke. This is the most common means by which the focus is switched from one control to another in a value set window.

The keystrokes processed by a value set control are:

Key Name	Action Performed
Down Arrow	Moves the selection cursor down one item. When the selection cursor reaches the bottom, the Down Arrow has no effect.
Up Arrow	Moves the selection cursor up one item. When the selection cursor reaches the top, the Up Arrow has no effect.
Left Arrow	Moves the selection cursor left one item. When the selection cursor reaches the leftmost column, the Left Arrow has no effect.
Right Arrow	Moves the selection cursor right one item. When the selection cursor reaches the rightmost column, the Right Arrow has no effect.
Home	Moves the selection cursor to the leftmost column of the value set control (NLS dependent). Pressing the Home key when the leftmost column is selected has no effect. The row index does not change.
End	Moves the selection cursor to the rightmost column of the value set control (NLS dependent). Pressing the End key when the rightmost column is selected has no effect. The row index does not change.
PgDn	Moves the selection cursor to the bottom row of the value set control. Pressing the Page Down key when the bottom row is selected has no effect. The column index does not change.
PgUp	Moves the selection cursor to the top row of the value set control. Pressing the Page Up key when the top row is selected has no effect. The column index does not change.
Ctrl + Home	Moves the selection cursor to the item in the top row and leftmost column of the value set control (NLS dependent). Pressing the Ctrl + Home keys when the top row and leftmost column is selected has no effect.
Ctrl + End	Moves the selection cursor to the bottom row and rightmost column of the value set control (NLS dependent). Pressing the Ctrl + End keys when the bottom row and rightmost column is selected has no effect.

Enter Sends a VN_ENTER notification code to the owner of the value set with the row and column indices of the selected item.

(Mnemonic) If the VS_TEXT style bit is set for the value set, any mnemonics specified can be used to select an item.

Default Processing

For a description of the default processing, see "WM_CHAR" on page 12-24.

WM_PRESPARAMCHANGED (in Value Set Controls)

For the cause of this message, see "WM_PRESPARAMCHANGED" on page 12-48.

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 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value; must be 0.

Remarks

The application uses this message to notify the value set that a given inherited presentation parameter has changed.

Default Processing

For a description of the default processing, see "WM_PRESPARAMCHANGED" on page 12-48.

WM_QUERYWINDOWPARAMS (in Value Set Controls)

For the cause of this message, see "WM_QUERYWINDOWPARAMS" on page 12-53.

Parameters

param1

wndparams *(PWNDPARAMS)*
Pointer.

Pointer to a WNDPARAMS window parameter structure. See WNDPARAMS on page A-125 for descriptions of the default fields. For a value set, the valid values for the *ulStatus* field are WPM_CBCTLDATA and WPM_CTLDATA.

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

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

result *(BOOL)*
Success indicator.

TRUE Successful operation.
FALSE Error occurred.

Remarks

The value set control window procedure responds to this message by returning the information in the buffer provided. If this message is sent to a value set window of another process, the information in, or identified by, the *wndparams* parameter must be in memory shared by both processes.

Default Processing

For a description of the default processing, see "WM_QUERYWINDOWPARAMS" on page 12-53.

WM_SETWINDOWPARAMS (in Value Set Controls)

For the cause of this message, see "WM_SETWINDOWPARAMS" on page 12-60.

Parameters

param1

wndparams *(PWNDPARAMS)*
Pointer.

Pointer to a WNDPARAMS structure. See WNDPARAMS on page A-125 for descriptions of the fields. For a value set, the valid value of the *ulStatus* field is WPM_CTLDATA.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply

result *(BOOL)*
Success indicator.

TRUE Successful operation.
FALSE Error occurred.

Remarks

If this message is sent to a value set window of another process, the information in, or identified by, the *wndparams* parameter must be in memory shared by both processes.

Default Processing

For a description of the default processing, see "WM_SETWINDOWPARAMS" on page 12-60.

WM_SIZE (in Value Set Controls)

For the cause of this message, see "WM_SIZE" on page 12-61.

Parameters

For a description of the parameters, see "WM_SIZE" on page 12-61.

Remarks

When the value set window is sized, the value set control defaults the size of each item by dynamically dividing the window size by the number of rows and columns. It allows enough room for the border, selection cursor, and selection emphasis, and defaults the spacing between items to 0. To override these default settings, the application must resend the VM_SETMETRICS message.

Default Processing

For a description of the default processing, see "WM_SIZE" on page 12-61.

Chapter 28. Clipboard Messages

Purpose

The clipboard is used by the end-user to transfer data between Presentation Manager* (PM) applications using the following operations.

Cut Remove from a window, leaving a gap in the source, and save for later use.

Copy Copy from a window, leaving the source intact, and save for later use.

Paste Paste the **cut** or **copied** data into the window of an application (the target).

WM_DESTROYCLIPBOARD

This message is sent to the clipboard owner when the clipboard is emptied through a call to WinEmptyClipbrd.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value.

Remarks

If there is any data that has been set with the CFI_OWNERFREE flag, the clipboard owner must release the data at this time.

Default Processing

None.

---

* Trademark of IBM Corporation

WM_DRAWCLIPBOARD

This message is sent to the clipboard viewer window whenever the contents of the clipboard change; that is, as a result of the WinCloseClipbrd function following a call to WinSetClipbrdData.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, 0.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

None.

WM_HSCROLLCLIPBOARD

This message is sent to the clipboard-owner window when the clipboard contains a data handle for the CFI_OWNERDISPLAY format, and there is an event in the clipboard viewer's horizontal scroll bar.

Parameters

param1

hwndhwndViewer *(HWND)*
Handle.

This contains a handle to the clipboard application window.

param2

sposScroll *(SHORT)*
Scroll position.

The position is either:

0 *scodeScroll* is other than SB_SLIDERPOSITION
Other The position of the slider when *scodeScroll* is SB_SLIDERPOSITION.

scodeScroll *(SHORT)*
Scroll-bar code

This is one of the SB_* scroll-bar codes as defined in "WM_HSCROLL (in Horizontal Scroll Bars)" on page 20-3.

SB_LINELEFT	Sent if the operator clicks the left arrow of the scroll bar, or presses the VK_LEFT key.
SB_LINERIGHT	Sent if the operator clicks the right arrow of the scroll bar, or presses the VK_RIGHT key.
SB_PAGELEFT	Sent if the operator clicks the area to the left of the slider, or presses the VK_PAGELEFT key.
SB_PAGERIGHT	Sent if the operator clicks the area to the right of the slider, or presses the VK_PAGERIGHT key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider. *sposScroll* contains the final position of the slider.
SB_SLIDERTRACK	Sent every time the slider position changes if the operator moves the scroll bar slider with the pointer device.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The clipboard owner is responsible for displaying the clipboard contents. The clipboard owner should use WinInvalidateRect or repaint as desired. The scroll-bar position is also reset.

Default Processing

None.

WM_PAINTCLIPBOARD

This message is sent when the clipboard contains a data handle with the CFI_OWNERDISPLAY information flag set.

Parameters

param1

hwndhwndViewer *(HWND)*
Handle.

This is a handle to the clipboard application window.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

As the clipboard owner is responsible for displaying the clipboard contents, this message notifies the clipboard application that its client area needs repainting. The WM_PAINTCLIPBOARD message is sent to the owner of the clipboard to request repainting of all or part of the client area of the clipboard application.

Note: To determine whether the entire client area needs repainting or just a portion of it, the clipboard owner must compare the dimensions of the drawing area to the dimensions given in the most recent WM_SIZECLIPBOARD message.

Default Processing

None.

WM_RENDERALLFMTS

This message is sent to the application that owns the clipboard while the application is being destroyed.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value.

Remarks

The application renders the clipboard data in all formats it is capable of generating and passes a handle to each format to WinSetClipbrdData. This ensures that the data in the clipboard can be rendered even though the application has been destroyed.

Default Processing

None.

WM_RENDERFMT

This message is a request to the clipboard owner to render the data of the format specified in *usfmt*.

Parameters

param1

usfmt *(USHORT)*
Data format.

This is the format of the data to be rendered.

CF_BITMAP	A bit map.
CF_DSPBITMAP	A bit-map representation of a private data format.
CF_DSPMETAFILE	A metafile representation of a private data format.
CF_DSPTEXT	A textual representation of a private data format.
CF_METAFILE	A metafile.
CF_TEXT	An array of text characters.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The data is rendered into a global handle, which is then set into the clipboard with WinSetClipbrdData.

Default Processing

None.

WM_SIZECLIPBOARD

This message is sent when the clipboard contains a data handle for the CFI_OWNERDISPLAY format, and the clipboard application window has changed size.

Parameters

param1

hwndViewer *(HWND)*
Handle of viewer window.

param2

ppaint *(PRECTL)*
Rectangle to be re-painted.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The default window procedure takes no action on this message except to set *flreply* to 0.

WM_VSCROLLCLIPBOARD

This message is sent to the clipboard owner window when the clipboard contains a data handle for the CFI_OWNERDISPLAY format, and there is an event in the clipboard viewer's vertical scroll bar.

Parameters

param1

hwndhwndViewer *(HWND)*
Handle.

This contains a handle to the clipboard application window.

param2

sposScroll *(SHORT)*
Scroll position.

The position is either:

0 *scodeScroll* is other than SB_SLIDERPOSITION
Other The position of the slider when *scodeScroll* is SB_SLIDERPOSITION.

scodeScroll *(SHORT)*
Scroll-bar code.

This is one of the SB_* scroll-bar codes as defined in "WM_HSCROLL (in Horizontal Scroll Bars)" on page 20-3.

SB_LINELEFT	Sent if the operator clicks the left arrow of the scroll bar, or depresses the VK_LEFT key.
SB_LINERIGHT	Sent if the operator clicks the right arrow of the scroll bar, or depresses the VK_RIGHT key.
SB_PAGELEFT	Sent if the operator clicks the area to the left of the slider, or depresses the VK_PAGELEFT key.

SB_PAGERIGHT	Sent if the operator clicks the area to the right of the slider, or depresses the VK_PAGERIGHT key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider. *sposScroll* contains the final position of the slider.
SB_SLIDERTRACK	Sent every time the slider position changes if the operator moves the scroll bar slider with the pointer device.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The clipboard owner is responsible for displaying the clipboard contents. The clipboard owner should use WinInvalidateRect or repaint as desired. The scroll bar position is also reset.

Default Processing

None.

Chapter 29. Direct Manipulation (Drag) Messages

Purpose

This section describes the processing that occurs during a direct manipulation operation when the application sends or receives a direct manipulation (DM_*) message.

DM_DISCARDOBJECT

This message is sent to a source that supports the "DRM_DISCARD" rendering method.

Parameters

param1

pDraginfo *(PDRAGINFO)*
Pointer.

Pointer to the DRAGINFO structure representing the items to be discarded.

param2 *(MPARAM)*
Reserved.

NULL Reserved value.

Returns

reply

ulAction *(ULONG)*
Flag.

Flag giving responsibility for the operation.

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.

Remarks

This message is sent to the source window for the drag action. The source should make a copy of the parameters and return. The source should also create a separate thread to execute the discard action if it responds with DRR_SOURCE.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action on it, other than to set *ulAction* to the default value of NULL.

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

reply *(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.

Remarks

The receiver of this message should return the action that the sender should take.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action other than to return FALSE.

DM_DRAGFILECOMPLETE

This message is sent when a direct manipulation operation on a file or files is complete.

Parameters

param1 *(HSTR)*
File handle.

param2 *(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

reply *(ULONG)*

Reserved.

0 Reserved value, 0.

Remarks

param1 is HSTR for the source file if this message is sent by DrgDragFiles, and is HSTR for the target file if this message is sent by DrgAcceptDroppedFiles.

This message is sent by DrgDragFiles to its caller when the move or copy operation is completed, regardless of success or failure. It is also sent by DrgAcceptDroppedFiles when a file has been successfully dropped on the caller.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action other than to return 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.

Pointer to the DRAGINFO structure for the drag operation.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*

Reserved.

0 Reserved value, 0.

Remarks

This message allows for target emphasis and de-emphasis during the direct manipulation process. This message is not sent when a drop occurs. Use DM_DROP as a signal to remove the target emphasis.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action on it other than to return 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.

Parameters

param1

pDraginfo *(PDRAGINFO)*
Pointer.

Pointer to the DRAGINFO structure representing the object being dragged.

param2

Pointer location.

Pointing device pointer location.

sxDrop *(SHORT)*
X-coordinate.

X-coordinate of the pointing device pointer in desktop coordinates.

syDrop *(SHORT)*
Y-coordinate.

Y-coordinate of the pointing device pointer in desktop coordinates.

Returns

reply

usDrop *(USHORT)*
Drop indicator.

DOR_DROP	Object can be dropped. When this reply is given, *usDefaultOp* must be set to indicate which operation will be performed if the user should drop at this location. This is used to provide visual feedback to the user.
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)*
Default operation.

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.

Remarks

This message is sent to the window that is directly under the hot spot of the mouse pointer during the drag operation when any of the following conditions are met:

- The user moves the mouse.
- A key is pressed.
- A WM_BUTTON1UP, WM_BUTTON2UP, WM_BUTTON3UP, or WM_ENDDRAG message is received, indicating that the direct manipulation operation corresponds to the *vkTerminate* parameter specified by the source on the call to DrgDrag. In this case the message is sent only if the mouse has moved since the last DM_DRAGOVER message was sent.

The receiver can gain access to *pDraginfo* with DrgAccessDraginfo. The acceptability of the dragged objects can be determined by querying the *hstrType* and *hstrRMF* string handles in each of the DRAGITEM structures carried in *pDraginfo*.

The receiver should provide target emphasis for itself if it returns DOR_DROP for this message. The receiver can use DrgSetDragPointer to change the bit map while it is being dragged over. A DM_DRAGLEAVE or DM_DROP message will be sent to the target in the future. Target emphasis should be removed at that time.

If *usOperation* in DRAGINFO is DO_DEFAULT or DO_UNKNOWN and the target returns DOR_DROP for *usDrop*, *usDefaultOp* should be set to reflect what the target defines as the default operation. This information is used to provide the appropriate modification to the drag pointer and the target's default operation will be passed in the *usOperation* field of the DRAGINFO structure specified in the DM_DROP message.

The *usDrop* parameter is treated as DOR_NEVERDROP if all of the following occur:

- The value of the *usOperation* field in the DRAGINFO structure is DO_DEFAULT or DO_UNKNOWN.
- The value of the *usDrop* parameter is DOR_DROP.
- The *usDefaultOp* parameter does not contain one of the defined values.

Otherwise, if the value of the *usOperation* field is not DO_DEFAULT or DO_UNKNOWN, the *usDefaultOp* parameter is ignored.

Default Processing

The WinDefWindowProc function returns DOR_NEVERDROP to the sender of this message.

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.

Parameters

param1

pDraginfo *(PDRAGINFO)*
Pointer.

Pointer to the DRAGINFO structure that represents the object being dragged.

param2
Target's reply.

Target's reply to the DM_DRAGOVER message.

usDrop *(USHORT)*
Drop indicator.

usDefaultOp *(USHORT)*
Default operation.

Target-defined default operation.

Returns

reply *(ULONG)*
Reserved.

Remarks

The source window can use this message to modify its behavior or appearance based on a target's response to the DM_DRAGOVER message.

See "DM_DRAGOVER" on page 29-4 for a description of the target's possible responses.

Default Processing

The WinDefWindowProc function does not expect to receive this message and therefore takes no action on it other than to return NULL.

DM_DROP

This message is sent to the target when the dragged object is dropped.

Parameters

param1

pDraginfo *(PDRAGINFO)*
Pointer.

Pointer to the DRAGINFO structure.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message is sent to the window directly under the hot spot of the mouse pointer at the completion of a direct manipulation operation only if DOR_DROP was returned for the DM_DRAGOVER message sent to the window during the drag.

The receiver can obtain access to *pDraginfo* with DrgAccessDraginfo.

The receiver should immediately remove any target emphasis and post a private message to itself to initiate the data transfer conversations needed to complete the operation.

The receiver should use the *cxOffset*, and *cyOffset*, fields in the DRAGITEM structure to position the dropped object within its window relative to the drop point, so that no movement of the dragged image is perceived by the user when the drop occurs.

When the application receiving the DM_DROP message has finished all data transfer operations, it should free the DRAGINFO structure using DrgFreeDraginfo.

Default Processing

The WinDefWindowProc function calls DrgDeleteDraginfoStrHandles and DrgFreeDraginfo for *pDraginfo* and returns 0.

DM_DROPHELP

This message requests help for the current drag operation.

Parameters

param1

pDraginfo *(PDRAGINFO)*
Pointer.

Pointer to the DRAGINFO structure used in the drag operation.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message is posted to the target of a drop when F1 is pressed during a direct manipulation operation.

The *usOperation* member of *pDraginfo* can be used to provide help information in the context of the drag operation during which it was requested.

Default Processing

The WinDefWindowProc function calls DrgDeleteDraginfoStrHandles and DrgFreeDraginfo for *pDraginfo* and returns 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.

param2 *(USHORT)*
Flags.

TRUE Apply emphasis.
FALSE Remove emphasis.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action other than to return 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 *(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

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message is used to inform a source that the target has completed its part of a rendering operation. It is sent by the target to the source.

The target must send this message under any of the following circumstances:

- The target receives a DM_RENDERCOMPLETE message and will not retry the operation.
- The target completes the rendering operation without involvement from the source.
- The target wants to terminate a rendering operation in progress.
- The target chooses not to render an object that was dropped on it.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action other than to return 0.

DM_FILERENDERED

This message is sent to the window handling the drag conversation for the caller of DrgDragFiles.

Parameters

param1 *(PRENDERFILE)*
Pointer.

Pointer to a RENDERFILE structure.

param2 *(USHORT)*
Flags.

TRUE Operation succeeded.
FALSE Operation failed.

Returns

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

This message is sent when the rendering (moving or copying) of a file is complete. The handle of this window is the *hwndDragFiles* field of the RENDERFILE structure sent on DM_RENDERFILE.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action other than to return 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

pDragItem *(PDRAGINFO)*
Pointer.

Pointer to the DRAGINFO structure representing the objects to be printed.

param2

pPrintDest *(PPRINTDEST)*
Pointer.

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

reply

ulAction *(ULONG)*
Flag.

Flag giving responsibility for the print operation.

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.

Remarks

This message is sent to the source window procedure. The source window procedure is responsible for interpreting the structure given by *param2*. It should make a copy of all the parameters and then return.

The receiver of this message should create a thread in which to dispatch this message in order to facilitate a prompt reply. The thread can then call DevPostDeviceModes and DevOpenDC as appropriate.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action on it, other than to set *ulAction* to the default value of NULL.

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

Dxfer *(DRAGTRANSFER)*
DRAGTRANSFER structure.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

success *(BOOL)*
Success indicator.

TRUE Successful completion.
FALSE Error occurred.

Remarks

The target sends this message to a source window to request a rendering of an object. If the source returns FALSE, it may set flags in the DRAGTRANSFER structure that tell the target how to perform the rendering operation on its own, or how to retry the operation. If no flags are set, the source will not allow a rendering of the object.

If TRUE is returned, the message was processed by the recipient and the requested rendering will take place. The source will post a DM_RENDERCOMPLETE message to the target when the rendering is complete.

If FALSE is returned, either the message was not processed by the recipient, or the recipient could not perform the requested rendering. See *usReply* in DRAGTRANSFER for more information.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action other than to return 0.

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.

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

reply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

If the rendering operation failed for an intermittent reason, the source can allow the target to retry the operation. The source should return to the state it was in when the drop occurred for that object. The target resumes the rendering operation from the beginning.

If the rendering operation encounters a permanent failure, the source should fail the operation and proceed as if the rendering was completed.

If the rendering operation completes successfully, the source should return to the state it was in when the drop occurred for that object. This allows the target to retry the operation if its portion of the rendering failed. The target must post a DM_ENDCONVERSATION message when either of the following occurs:

- It determines that the rendering operation successfully completed
- It chooses not to retry a rendering operation that failed.

Default Processing

The WinDefWindowProc function should send a DM_ENDCONVERSATION message to the window indicated in the *hwndItem* field of the DRAGITEM structure. The message should indicate that the target failed in its part of the rendering operation. Sending the DM_ENDCONVERSATION message allows the source to release the resources it dedicated to the rendering operation.

DM_RENDERFILE

This message is sent to the caller of DrgDragFiles to tell it to render a file.

Parameters

param1 *(PRENDERFILE)*

Pointer.

Pointer to a RENDERFILE structure.

param2 *(ULONG)*

Reserved.

0 Reserved value, 0.

Returns

reply *(BOOL)*

Render handling.

TRUE The receiver handled the rendering.
FALSE DrgDragFiles should render this file.

Remarks

This message is sent when TRUE is specified in DrgDragFiles. The receiver should perform the operation indicated by the TRUE field in the RENDERFILE structure, moving or copying *hstrSource* to *hstrTarget*.

When the operation is complete, a DM_FILERENDERED message should be sent to *hwndDragFiles* window.

The RENDERFILE structure is allocated temporarily for the receiver of this message. The receiver should make a copy if it needs to use the data in this structure after returning.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action other than to return 0.

DM_RENDERPREPARE

This message tells a source to prepare for the rendering of an object.

Parameters

param1

pDxfer *(PDRAGTRANSFER)*
Pointer.

Pointer to a DRAGTRANSFER structure.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

success *(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.

Remarks

This message must be sent when DC_PREPARE is on in the DRAGITEM structure.

This message is used to allow the source to create an invisible window to handle the conversation required for the data transfer.

Default Processing

The WinDefWindowProc function does not expect to receive this message and takes no action other than to return 0.

Chapter 30. Dynamic Data Exchange Messages

Purpose

This section describes the message part of the DDE protocol, which is a set of guidelines that allows two applications to share data freely between one another; not necessarily driven directly by user input.

Note: DDE operates between two specific applications, each of which must be aware of the other, and active.

WinDdeInitiate, WinDdePostMsg, and WinDdeRespond are the functions associated with these messages.

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

hwndhwnd *(HWND)*
Window handle of the sender.

param2

pDdeStruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page A-23.

The acknowledging application modifies the *usStatus* 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

flReply *(ULONG)*
Reserved.

0 Reserved Value.

Default Processing

None.

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

hwndhwnd *(HWND)*
Window handle of the sender.

param2

pDdeStruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page A-23.

Flags in the *usStatus* 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

flReply *(ULONG)*
Reserved.

0 Reserved Value.

Remarks

The receiving application is expected to reply with a positive WM_DDE_ACK message if it can provide the requested data, or with a negative one if it can not.

Default Processing

None.

WM_DDE_DATA

This message notifies a client application of the availability of data. It is always posted.

Parameters

param1

hwndhwnd *(HWND)*
Window handle of the sender.

param2

pDdeStruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page A-23.

Flags in the *usStatus* 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

flReply *(ULONG)*
Reserved.

0 Reserved value, 0.

Default Processing

None.

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

hwndhwnd *(HWND)*
Window handle of the server.

param2

pDdeStruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page A-23.

offabData contains the commands to be executed.

Returns

flReply *(ULONG)*

Reserved.

0 Reserved Value.

Default Processing

None.

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

hwndhwnd *(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

reply

fresult *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

Upon receiving this message, all applications with names matching the application name (where specified), that support the topic identified by the topic name, are expected to acknowledge.

A modal window, for example a message box, must not be invoked during the processing of this message.

Default Processing

None.

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

hwndhwnd *(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

reply

fresult *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

A modal window, such as a message box, must not be posted during the processing of this message.

Default Processing

None.

WM_DDE_POKE

This message requests an application to accept an unsolicited data item. It is always posted.

Parameters

param1

hwndhwnd *(HWND)*
Window handle of the sender.

param2

pDdeStruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page A-23.

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

flReply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The receiving application is expected to reply with a positive WM_DDE_ACK message if it accepts the unsolicited data, or with a negative WM_DDE_ACK if it does not.

Default Processing

None.

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

hwndhwnd *(HWND)*
Window handle of the server.

param2

pDdeStruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page A-23.

offszItemName identifies which data item is being requested.

usFormat identifies in which registered DDE data format the data item is to be rendered.

Returns

flReply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The receiving application is expected to respond with a WM_DDE_DATA message, containing the requested data, if possible. Otherwise, it is expected to respond with a negative WM_DDE_ACK message.

Default Processing

None.

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

hwndhwnd *(HWND)*
Window handle of the sender.

param2

flReserved *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns

flReply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

Upon receiving this message, an application is expected to post a WM_DDE_TERMINATE message in response.

Default Processing

None.

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

hwndhwnd *(HWND)*
Window handle of a sender.

param2

pDdeStruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure (see DDESTRUCT on page A-23). *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

flReply *(ULONG)*
Reserved.

0 Reserved value, 0.

Remarks

The receiving application is expected to reply with a positive WM_DDE_ACK message if it can honor the request, or a negative one if it cannot.

Default Processing

None.

Chapter 31. Help Manager Messages

Purpose

This section describes the processing of messages sent by the Help Manager or applications in response to requests for help by the user.

HM_ACTIONBAR_COMMAND

This message is sent to the current active application window by the help manager to notify the application when the user selects a tailored action bar item.

Parameters

param1

idCommand *(USHORT)*
Identity of the action bar item that was selected.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Default Processing

None.

HM_CONTROL

This message is sent by the help manager to the child of the coverpage window to add a control in the control area of a window.

Parameters

param1

usreserved *(USHORT)*
Reserved.

controlres *(USHORT)*

The res number of the control that was selected. For author-defined push buttons, this is the res identification number that was specified with the push button tag (**:pbutton.**). For default push buttons, this is the res identification number defined in the PMHELP.H file.

param2 *(ULONG)*
Reserved.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

If an application wants to filter any of the controls, it can subclass the child of the coverpage window and intercept this message. If the application does not intercept this message, the help manager adds the control to the control area.

Default Processing

None.

HM_CREATE_HELP_TABLE

This message is sent by the application to give the help manager a new help table.

Parameters

param1

pHELPTABLE *(PHELPTABLE)*
Help table.

This points to a help table structure; see HELPTABLE on page A-63.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

ulreturnValue *(ULONG)*
Return code.

0	The procedure was successfully completed
Other	See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Default Processing

None.

HM_DISMISS_WINDOW

This message tells the help manager to remove the active help window.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

ulreturnValue *(ULONG)*
Return code.

0	The help window was successfully removed
Other	There was no associated help window.

See also the values of the *ulErrorCode* parameter of the HM_ERROR message.

Remarks

If the user requests help from a primary or secondary window, and then interacts with the primary or secondary window without leaving help, the currently displayed help window might not be appropriate for the application window. This message gives the application the ability to remove that help window.

Default Processing

None.

HM_DISPLAY_HELP

This message tells the help manager to display a specific help window.

Parameters

param1

This parameter depends on the value of the *usTypeFlag* parameter.

For a value of the *usTypeFlag* parameter of HM_RESOURCEID.

idHelpPanelId *(USHORT)*
Identity of the help window.

This points to a USHORT data type.

For a value of the *usTypeFlag* parameter of HM_PANELNAME.

pHelpPanelName *(PSTRL)*
Name of the help window.

This points to a PSZ data type.

param2

usTypeFlag *(USHORT)*
Flag indicating how to interpret the first parameter.

HM_RESOURCEID	Indicates the *param1* points to the identity of the help window.
HM_PANELNAME	Indicates the *param1* points to the name of the help window.

Returns

reply

ulreturnValue *(ULONG)*
Return code.

0	The window was successfully displayed
Other	See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Default Processing

None.

HM_ERROR

This message notifies the application of an error caused by a user interaction.

Parameters

param1

ulErrorCode *(ULONG)*

Error code.

A constant describing the type of error that occurred. The application can also receive some of these error constants in the *fIreply* parameter of messages it has sent to the help manager.

The error constants are:

HMERR_LOAD_DLL	The resource DLL was unable to be loaded.
HMERR_NO_FRAME_WND_IN_CHAIN	There is no frame window in the window chain from which to find or set the associated help instance.
HMERR_INVALID_ASSOC_APP_WND	The application window handle specified on the WinAssociateHelpInstance function is not a valid window handle.
HMERR_INVALID_ASSOC_HELP_INST	The help instance handle specified on the WinAssociateHelpInstance function is not a valid window handle.
HMERR_INVALID_DESTROY_HELP_INST	The window handle specified as the help instance to destroy is not of the help instance class.
HMERR_NO_HELP_INST_IN_CHAIN	The parent or owner chain of the application window specified does not have an associated help instance.
HMERR_INVALID_HELP_INSTANCE_HDL	The handle specified to be a help instance does not have the class name of a help manager instance.
HMERR_INVALID_QUERY_APP_WND	The application window specified on a WinQueryHelpInstance function is not a valid window handle.
HMERR_HELP_INST_CALLED_INVALID	The handle of the instance specified on a call to the help manager does not have the class name of a help manager instance.
HMERR_HELPTABLE_UNDEFINE	The application did not provide a help table for context-sensitive help.
HMERR_HELP_INSTANCE_UNDEFINE	The help instance handle specified is invalid.
HMERR_HELPITEM_NOT_FOUND	Context-sensitive help was requested but the ID of the main help item specified was not found in the help table.
HMERR_INVALID_HELPSUBITEM_SIZE	The help subtable item size is less than 2.
HMERR_HELPSUBITEM_NOT_FOUND	Context-sensitive help was requested but the ID of the help item specified was not found in the help subtable.
HMERR_INDEX_NOT_FOUND	The index is not in the library file.
HMERR_CONTENT_NOT_FOUND	The library file does not have any content.
HMERR_OPEN_LIB_FILE	The library file cannot be opened.
HMERR_READ_LIB_FILE	The library file cannot be read.
HMERR_CLOSE_LIB_FILE	The library file cannot be closed.
HMERR_INVALID_LIB_FILE	Improper library file provided.
HMERR_NO_MEMORY	Unable to allocate the requested amount of memory.
HMERR_ALLOCATE_SEGMENT	Unable to allocate a segment of memory for memory allocation requests from the help manager.
HMERR_FREE_MEMORY	Unable to free allocated memory.
HMERR_PANEL_NOT_FOUND	Unable to find the requested help window.
HMERR_DATABASE_NOT_OPEN	Unable to read the unopened database.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

There is no other way to communicate the error to the application since the user initiated communication, not the application. Other errors which occur when the application sends a message to the help manager are returned as the *flreply* parameter of the message.

The help manager does not display any error messages to the user. Instead, the help manager sends or returns all error notifications to the application so that it can display its own messages. This procedure ensures a consistent message interface for all user messages.

Default Processing

None.

HM_EXT_HELP

When the help manager receives this message, it displays the extended help window for the active application panel.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

ulreturnValue *(ULONG)*
Return code.

0 The extended help window was successfully displayed
Other See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Default Processing

None.

HM_EXT_HELP_UNDEFINED

This message is sent to the application by the help manager to notify it that an extended help window has not been defined.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

flreply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

When the extended help window is requested, the help manager searches the help table for its identity. If the extended help window identity associated with the current active window is zero, the help manager sends this message to the application to notify it that an extended help window has not been defined. The application then can:

- Ignore the request for help and not display a help window.
- Display its own window.
- Use the HM_DISPLAY_HELP message to tell the help manager to display a particular window.

Default Processing

None.

HM_GENERAL_HELP

When the help manager receives this message, it displays the general help window for the active application window.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

ulreturnValue *(ULONG)*
Return code.

0 The general help window was successfully displayed.
Other See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Default Processing
None.

HM_GENERAL_HELP_UNDEFINED
This message is sent to the application by the help manager to notify it that a general help window has not been defined.

Parameters
param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, 0.

Returns
fireply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks
When the general help window is requested, the help manager searches the help table for its identity. If the general help window identity associated with the current active window is zero, the help manager sends this message to the application to notify it that a general help window has not been defined. The application can then:

- Ignore the request for help and not display a help window.
- Display its own window.
- Use the HM_DISPLAY_HELP message to tell the help manager to display a particular window.

Default Processing
None.

HM_HELP_CONTENTS
When the help manager receives this message, it displays the help contents window.

Parameters
param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns
reply

ulreturnValue *(ULONG)*
Return code.

0 The help contents window was successfully displayed.
Other See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Default Processing
None.

HM_HELP_INDEX
When the help manager receives this message, it displays the help index window.

Parameters
param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns
reply

ulreturnValue *(ULONG)*
Return code.

0	The help index window was successfully displayed.
Other	See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Default Processing
None.

HM_HELPSUBITEM_NOT_FOUND
The help manager sends this message to the application when the user requests help on a field and it cannot find a related entry in the help subtable.

Parameters
param1

usContext *(USHORT)*
Type of window on which help was requested.

HLPM_WINDOW	An application window
HLPM_FRAME	A frame window
HLPM_MENU	A menu window.

param2

sTopic *(SHORT)*
Topic identifier.

For a value of the *usContext* parameter of HLPM_WINDOW or HLPM_FRAME:

window	Identity of the window containing the field on which help was requested.
menu	Identity of the submenu containing the field on which help was requested.

sSubTopic *(SHORT)*
Subtopic identifier.

For a value of the *usContext* parameter of HLPM_WINDOW or HLPM_FRAME:

control	Control identity of the cursored field and on which help was requested.
-1	No menu item was selected
Other	Menu item identity of the currently selected submenu item on which help was requested.

Returns

reply

Informs the help manager what should be done next.

fAction *(BOOL)*
Action indicator:

For a value of the *usContext* parameter of HLPM_WINDOW or HLPM_FRAME:

FALSE Display the extended help window.
TRUE Do nothing.
For a value of the *usContext* parameter of HLPM_MENU:
FALSE Display the extended help window.

Remarks

If FALSE is returned from this message, the help manager displays the extended help window.

The application has the following options:

- Ignore the notification and not display help for that field or window.
- Display its own window.
- Use the HM_DISPLAY_HELP message to tell the help manager to display a particular window.

Default Processing

None.

HM_INFORM

This message is used by the help manager to notify the application when the user selects a hypertext field that was specified with the **reftype = inform** attribute of the **:link.** tag.

Parameters

param1

idnum *(USHORT)*
Window identity.

The identity that is associated with the hypertext field.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, zero.

Default Processing

None.

HM_INVALIDATE_DDF_DATA

The application sends this message to IPF to indicate that the previous DDF data is no longer valid.

Parameters

param1 *(ULONG)*

rescount The count of DDFs to be invalidated.

param2 *(PUSHORT)*

resarray The pointer to an array of unsigned 16-bit (USHORT) integers that are the *res* numbers of DDFs to be invalidated.

Note: If both param1 and param2 are NULL, then all the DDFs in that page will be invalidated.

Returns

reply

ulreturnvalue *(ULONG)*
Return code

0 The procedure was successfully completed.
Other See the values of the *errorcode* parameter of the HM_ERROR message.

Remarks

When IPF receives this message, it discards the current DDF data and sends a new HM_QUERY_DDF_DATA message to the object communication window.

This message should be sent to the child of the coverpage window handle.

Default Processing

None.

HM_KEYS_HELP

This message is sent by the application and informs the help manager to display the keys help window.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

ulreturnValue *(ULONG)*
Return code.

0 The keys help window was successfully displayed
Other See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Remarks

When the help manager receives this message, it sends a HM_QUERY_KEYS_HELP message to the active application window. The active application window is the window that was specified when the last HM_SET_ACTIVE_WINDOW message was sent. If no HM_SET_ACTIVE_WINDOW message was issued, then the active application window is the window specified in the WinAssociateHelpInstance call.

The application must return one of the following:

- The identity of a keys help window in the *usHelpPanel* parameter of the HM_QUERY_KEYS_HELP message.
- Zero, if no action is to be taken by the help manager for keys help.

Default Processing

None.

HM_LOAD_HELP_TABLE

The application sends this message to give the help manager the module handle that contains the help table, the help subtable, and the identity of the help table.

Parameters

param1

idHelpTable *(USHORT)*
Identity of the help table.

fsidentityflag *(USHORT)*
Help table identity indicator.

X'FFFF' Reserved value.

param2

MODULE *(HMODULE)*
Resource identity.

Handle of the module that contains the help table and help subtable.

Returns

reply

ulreturnValue *(ULONG)*
Return code.

0 The procedure was successfully completed
Other See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Default Processing

None.

HM_NOTIFY

This message is used by the application to sub-class and change the behavior or appearance of the help window.

Parameters

param1

controlres *(USHORT)*

The res number of the control that was selected. For author-defined push buttons, this is the res number that was specified with the push button tag (**:pbutton.**). For default push buttons, this is the res number defined in the PMHELP.H file.

usreserved *(USHORT)*

Reserved for events other than CONTROL_SELECTED and HELP_REQUESTED.

0 Reserved value, zero.

usevent *(USHORT)*

The type of event which has occurred.

CONTROL_SELECTED	A control was selected.
HELP_REQUESTED	Help was requested.
OPEN_COVERPAGE	The coverpage is displayed.
OPEN_PAGE	The child window of the coverpage is opened.
SWAP_PAGE	The child window of the coverpage is swapped.
OPEN_INDEX	The index window is displayed.
OPEN_TOC	The table of contents window is displayed.
OPEN_HISTORY	The history window is displayed.
OPEN_LIBRARY	The new library is opened.
OPEN_SEARCH_HIT_LIST	The search list displayed.

param2 *(ULONG)*

Window handle of relevant window.

Returns

reply

fresult *(BOOL)*

Return code

TRUE	IPF will not format the controls and re-size the window.
FALSE	IPF will process as normal.

Remarks

This message is sent to the application to notify it of events that the application would be interested in controlling.

Default Processing

None.

HM_QUERY

This message is sent to IPF by the application to request IPF-specific information, such as the current Instance handle, the active communication object window, the active window, or the group number of the current window.

Parameters

param1

usreserved *(USHORT)*
Reserved

0 Reserved value, zero.

usmessageId *(USHORT)*
Specifies the type of window to query. The value can be any of the following constants:

HMQW_INDEX	The handle of the index window.
HMQW_TOC	The handle of the Table of Contents window.
HMQW_SEARCH	The handle of the Search Hitlist window.
HMQW_VIEWEDPAGES	The handle of the Viewed Pages window.
HMQW_LIBRARY	The handle of the Library List window.
HMQW_OBJCOM_WINDOW	The handle of the active communication window.
HMQW_INSTANCE	The handle of the help instance.
HMQW_COVERPAGE	The handle of the help manager multiple document interface (MDI) parent window. It is where the secondary windows are contained within the parent window.
HMQW_VIEWPORT	The handle of the viewport window specified in the low-order word of param1 and in param2.
HMQW_GROUP_VIEWPORT	The group number of the window whose handle is specified in param2.
HMQW_RES_VIEWPORT	The res number of the window whose handle is specified in param2.
HMQW_ACTIVEVIEWPORT	The handle of the currently active window.
USERDATA	The previously stored user-data.

usselectionid *(USHORT)*
Specifies whether a res ID, ID number, or group number is being requested. The value can be any of the following constants:

HMQVP_NUMBER	A pointer to a USHORT that holds the res ID of the window.
HMQVP_NAME	A pointer to a null-terminated string that holds the ID of the window.
HMQVP_GROUP	The group number of the window.

param2 *(PVOID)*
Param2 depends on the value of *param1 messageid*:

If *param1 messageid* is HMQW_VIEWPORT, then *param2* is a pointer to the res number, ID, or group ID.

If *param1 messageid* is HMQW_GROUP_VIEWPORT, then *param2* is the handle of the viewport for which the group number is assigned.

If *param1 messageid* is HMQW_RES_VIEWPORT, then *param2* is the handle of the viewport for which the res number is requested.

Returns

reply

ulreturnvalue *(ULONG)*
Return value.

0	The procedure was not successfully completed.
Other	The handle (*HWND*), group number (*USHORT*), or res number (*USHORT*) of the window, or the user data (*USHORT*), depending on the value of *param1 selectionid*.

Default Processing
None.

HM_QUERY_DDF_DATA
This message is sent to the communication object window by IPF when it encounters the dynamic data formatting (**:ddf.**) tag.

Parameters
param1 *(HWND)*

pageclienthwnd The client handle of the page that contains the object communication window.

param2 *(ULONG)*

resId The res ID associated with the DDF tag.

Returns
reply

hddfddfhandle *(HDDF)*
Return code

0 An error has occurred in the application's DDF processing.
Other The DDF handle to be displayed.

Note: Once this handle has been returned, the HDDF handle can no longer be used by the application.

Remarks
Upon receiving this message, the communication object calls DdfInitialize to indicate the start of dynamic data formatting (DDF). Any combination of other DDF calls are then made to describe this data. When this is complete, the communication object finishes processing this message, indicating that the DDF data is complete. After that time, the DDF handle received from DdfInitialize is considered invalid.

Default Processing
None.

HM_QUERY_KEYS_HELP
When the user requests the keys help function, the help manager sends this message to the application.

Parameters
param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns
reply

usHelpPanel *(USHORT)*
The identity of the application-defined keys help window to be displayed.

0 Do nothing
Other Identity of the keys help window to be displayed.

Remarks

The application responds by returning the identity of the requested keys help window. The help manager then displays that help window. Returning 0 in the *usHelpPanel* parameter indicates that the help manager should do nothing for the keys help function.

Default Processing

None.

HM_REPLACE_HELP_FOR_HELP

This message tells the help manager to display the application-defined Help for Help window instead of the help manager Help for Help window.

Parameters

param1

idHelpForHelpPanel *(USHORT)*
Identity of the application-defined Help for Help window.

0 Use the help manager Help for Help window.
Other Identity of the application-defined Help for Help window.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

fireply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks

An application may prefer to provide information that is more specific to itself, rather than the more general help information provided in the help manager Help for Help window.

Default Processing

None.

HM_REPLACE_USING_HELP

This message tells the help manager to display the application-defined Using help window instead of the help manager Using help window.

Parameters

param1

idUsingHelpPanel *(USHORT)*

The identity of the application-defined Using Help window.

0 Use the help manager Using Help window.
Other The identity of the application-defined Using Help window.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

fireply *(ULONG)*

Reserved.

0 Reserved value, zero.

Remarks

An application may prefer to provide information that is more specific to itself, rather than the more general help information that is provided in the help manager Using help window. The guidelines that define the current CUA interface recommend the **Using help** choice be provided in a pull-down menu from the **Help** choice.

Default Processing

None.

HM_SET_ACTIVE_WINDOW

This message allows the application to change the window with which the help manager communicates and the window to which the help window is to be positioned.

Parameters

param1

hwndActiveWindow *(HWND)*

The handle of the window to be made active.

Its window procedure receives all messages from the help manager until the application changes the active window with another HM_SET_ACTIVE_WINDOW message.

param2

hwndRelativeWindow *(HWND)*

The handle of the window next to which the help window is to be positioned.

The handle of the application window next to which the help manager will position a new help window.

HWND_PARENT	This help manager defined constant tells the help manager to trace the parent chain of the window that had the focus when the user requested help.
Other	Handle of the window next to which the help window is to be positioned.

Returns

reply

ulreturnValue *(ULONG)*

Return code.

0	The procedure has been successfully completed.
Other	See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Remarks

Normally the help manager communicates with the application window with which the help manager instance has been associated. The help window is positioned next to this same application window.

If the *hwndActiveWindow* parameter is 0, the *hwndRelativeWindow* parameter is set to 0. That is, if the active window is NULL HANDLE, the relative window is not used.

Default Processing

None.

HM_SET_COVERPAGE_SIZE

This message is sent to IPF by the application to set the size of the coverpage, the window within which all other IPF windows are displayed.

Parameters

param1 *(PRECTL)*

coverpagerectl A PRECTL containing the size of the coverpage.

param2 *(ULONG)*

Reserved.

0 Reserved value, zero.

Returns

reply

ulreturnvalue *(ULONG)*

Return code

0 The procedure was successfully completed.
Other See the values of the *errorcode* parameter of the HM_ERROR message.

Remarks

The default size for the coverpage of a book is the full width of the screen, while the default size for a help file is one-half the width of the screen.

This message takes effect immediately, changing the size of the coverpage. If the coverpage is not currently open, the requested size is saved for the next open.

Default Processing

None.

HM_SET_HELP_LIBRARY_NAME

This message identifies a list of help window library names to the help manager instance.

Parameters

param1

pHelpLibraryName *(PSTRL)*

Library name.

This points to a PSZ data type.

The string contains a list of help window library names that will be searched by the help manager for the requested help window. The names must be separated by a blank.

param2 *(ULONG)*

Reserved.

0 Reserved value, zero.

Returns

reply

ulreturnValue *(ULONG)*

Return code.

0 The newly specified library successfully replaced the current help window library name.
Other See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Remarks

Any subsequent communication to the help manager with this message replaces the current list of names with the newly specified list.

When help is requested, the help manager will search each library in the list for the requested help window.

Default Processing

None.

HM_SET_HELP_WINDOW_TITLE

This message allows the application to change the window text of a help window title.

Parameters

param1

pHelpWindowTitle *(PSTRL)*
Help window title.

This points to a PSZ data type.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

ulreturnValue *(ULONG)*
Return code.

0 The window title was successfully set.
Other See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Default Processing

None.

HM_SET_OBJCOM_WINDOW

This message is sent to IPF by the application to identify the communication object window to which the HM_INFORM and HM_QUERY_DDF_DATA messages will be sent. This message is not necessary if the communication object does not expect to receive either of these messages.

Parameters

param1 *(HWND)*

objcomhwnd The handle of the communication object window to be set.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

hwndprevioushwnd *(HWND)*
The handle of the previous communication object window.

Remarks

HM_INFORM and HM_QUERY_DDF_DATA messages which are not processed must be passed to the previous communication object window which was returned when HM_SET_OBJECT_WINDOW was sent.

Default Processing

None.

HM_SET_SHOW_PANEL_ID

This message tells the help manager to display, hide, or toggle the window identity for each help window displayed.

Parameters

param1

fsShowPanelId *(USHORT)*
The show window identity indicator:

CMIC_HIDE_PANEL_ID	Sets the show option off and the window identity is not displayed.
CMIC_SHOW_PANEL_ID	Sets the show option on and the window identity is displayed.
CMIC_TOGGLE_PANEL_ID	Toggles the display of the window identity.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns

reply

ulreturnValue *(ULONG)*
Return code.

0	The show window identity indicator was successfully changed.
Other	See the values of the *ulErrorCode* parameter of the HM_ERROR message.

Default Processing

None.

HM_SET_USERDATA

The application sends this message to IPF to store data in the IPF data area.

Parameters

param1 *(ULONG)*
Reserved.

0 Reserved value, zero.

param2 *(VOID)*
4-byte user data area.

Returns

reply

ulreturn-value *(ULONG)*
Return code.

TRUE	The user data was successfully stored.
FALSE	The call failed.

Default Processing
None.

HM_TUTORIAL
The help manager sends this message to the application window when the user selects the Tutorial choice from a help window.

Parameters
param1

pTutorialName *(PSTRL)*
Default tutorial name.

This points to a PSZ data type.

The string contains the name of the default tutorial program specified in the help manager initialization structure. A tutorial name specified in the help window definition overrides this default tutorial program.

param2 *(ULONG)*
Reserved.

0 Reserved value, zero.

Returns
fireply *(ULONG)*
Reserved.

0 Reserved value, zero.

Remarks
The application then calls its own tutorial program.

Default Processing
None.

HM_UPDATE_OBJCOM_WINDOW_CHAIN

This message is sent to the currently active communication object by the communication object who wants to withdraw from the communication chain.

Parameters

param1 *(HWND)*

The handle of the object to be withdrawn from the communication chain.

param2 *(HWND)*

Window containing the handle of the object to be replaced.

Returns

flreply *(ULONG)*

Reserved.

0 Reserved value, zero.

Remarks

The object that receives this message should check to see if the object handle returned from HM_SET_OBJCOM_WINDOW is equal to the handle in *param1*. If the handle is equal, then the handle in *param1* should be replaced by the handle in *param2*. If the handle is not equal *and* the handle previously received is not NULL HANDLE, then send HM_UPDATE_OBJCOM_WINDOW_CHAIN to that object.

Default Processing

None.

Chapter 32. Resource Files

This chapter describes the syntax for the resource language using railroad syntax, and describes the formats used.

Resource files are used to build dialog templates, menu templates, accelerator tables, extended attribute association tables, keyboard scancode mapping tables, keyboard names and fonts. The files must be compiled before they can be used by application programs.

How to Read the Syntax Definitions

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ▶▶— symbol indicates the beginning of a statement.

  The —▶ symbol indicates that the statement syntax is continued on the next line.

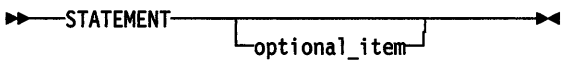
  The ▶— symbol indicates that a statement is continued from the previous line.

  The —▶◀ symbol indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the ▶— symbol and end with the —▶ symbol.

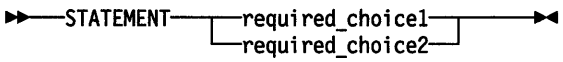
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path.

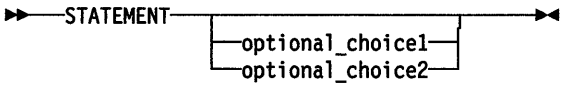


- If a choice can be made from two or more items, they appear vertically, in a stack.

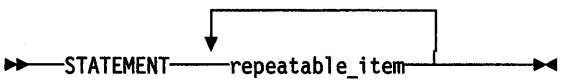
  If one of the items *must* be chosen, one item of the stack appears on the main path.



  If choosing one of the items is optional, the entire stack appears below the main path.



- An arrow returning to the left above the main path indicates an item that can be repeated.



  A repeat arrow above a stack indicates that a choice can be made from the stacked items, or a single choice can be repeated.

- Keywords appear in uppercase (for example, PARM1). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, parmx). They represent user-supplied names or values.

- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, they must be entered as part of the syntax.

Definitions Used in all Resources

Specification of Values

These rules apply to values specified in resources:

- Coordinates must be integers. There must be no space between the sign of the value and the value itself. For example, "–1" is allowed but "– 1" is not.
- Resource identifiers must be positive integers or names that resolve to positive integers.
- Real values, containing a decimal point, cannot be used.

Resource Load and Memory Options

The following options define when each resource is loaded and how memory is allocated for each resource.

LOADOPTION Resource loading options

PRELOAD	Resource is loaded immediately.
LOADONCALL	Resource is loaded when called.

MEMOPTION Resource memory options

FIXED	Resource remains at a fixed memory location.
MOVEABLE	Resource can be moved if necessary to compact memory.
DISCARDABLE	Resource can be discarded if no longer needed.
SEGALIGN	Resources are aligned on 64Kbyte boundaries.

Resource Script File Specification

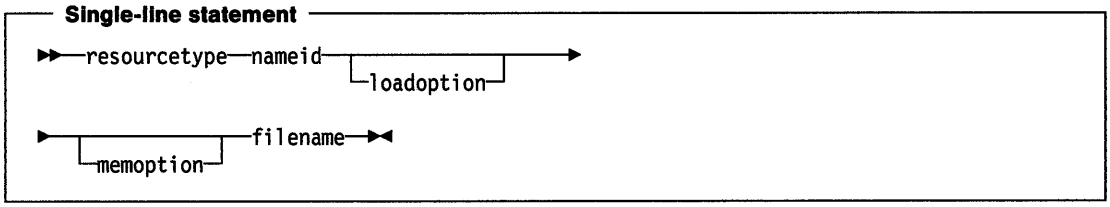
The resource script file defines the names and attributes of the resources to be added to the executable file of the application. The file consists of one or more resource statements that define the resource type and original file, if any. The following is a list of the types of resource statement:

- Single-line statements
- User-defined resources
- Directives
- Multiple-line statements.

The following sections describe these statements in detail.

Single-Line Statements

The general form for all single-line statements is:



resourcetype *(USHORT)*

One of the following keywords, specifying the type of resource to be loaded:

Keyword	Resource type
FONT	A font resource is a file containing a font.

POINTER A pointer resource is a bit map defining the shape of the pointing device pointer on the display screen.

ICON An icon resource is a bit map defining the shape of the icon to be used for a given application.

BITMAP A bit-map resource is a custom bit map that an application intends to use in its screen display or as an item in a menu.

DLGINCLUDE This statement tells the dialog editor which file to use as an include file for the dialogs in the resource file. The **nameid** is not applicable.

nameid *(USHORT)*
is either a unique name or an integer number identifying the resource. For a FONT resource, the **nameid** must be a number; it cannot be a name.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE and DISCARDABLE for POINTER, ICON, and FONT resources. The default for BITMAP resources is MOVEABLE. The FIXED option overrides both MOVEABLE and DISCARDABLE. The SEGALIGN option can be specified independently of other options, if it is not present the default (for all resources) is that the resource is not aligned on a 64KB boundary.

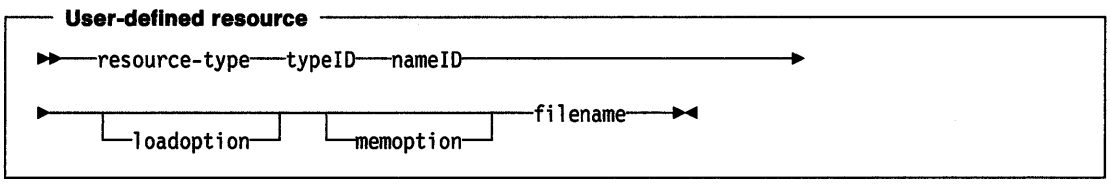
filename *(STR)*
is an ASCII string specifying the OS/2 Version 2.0 name of the file containing the resource. A full path name must be given if the file is not in the current working directory.

Example

```
POINTER pointer point.cur
POINTER pointer DISCARDABLE point.cur
POINTER 10 custom.cur

ICON desk desk.ico
ICON desk DISCARDABLE desk.ico
ICON 11 custom.ico

BITMAP disk disk.bmp
BITMAP disk DISCARDABLE disk.bmp
BITMAP 12 custom.bmp

FONT 5 CMROMAN.FNT
```

User-Defined Resources

An application can also define its own resource. The resource can be any data that the application intends to use. A user-defined resource statement has the form:



typeID
Either a unique name or an integer number identifying the resource type. If a number is given, it must be greater than 255. The type numbers 1 through 255 are reserved for existing and future predefined resource types.

nameID
Either a unique name or an integer number identifying the resource.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE.

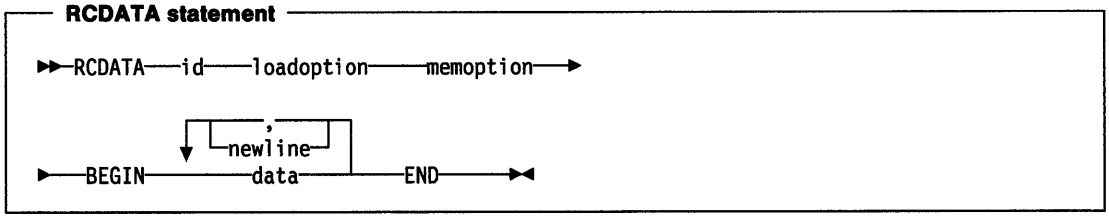
filename
An ASCII string specifying the OS/2* name of the file containing the cursor bit map. A full path name must be given if the file is not in the current working directory.

Example

```
RESOURCE MYRES   array   DATA.RES
RESOURCE 300     14      CUSTOM.RES
```

RCDATA statement

The RCDATA statement is provided to allow an application to define a simple data resource.



id
Either a unique name or an integer number identifying the resource.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE.

data
A number or string.

Example

```
RCDATA 4
BEGIN
 "Sample string."
 "TEST DATA."
 "A message."
END
```

Directives

The resource directives are special statements that define actions to perform on the file before it is compiled. The directives can assign values to names, include the contents of files, and control compilation of the file.

#include filename

rcinclude filename

These directives copy the contents of the file specified by **filename** into the resource before it is compiled. If **rcinclude** is used, the entire file is copied. If **#include** is used, only **#define** statements are copied.

Note: If an **rcinclude** is to be commented out, the open comment (/*) must appear on the same line as the directive.

---

* Trademark of IBM Corporation

Filename is an ASCII string. A full path name must be given if the file is not in the current directory or in the directory specified by the INCLUDE environment variable. The file extensions .I and .TMP must not be used as these are reserved for system use.

The **filename** parameter is handled as a C string, and two back-slashes must be given wherever one is expected in the path name (for example, root\\sub.) Or, a single forward slash (/) can be used instead of double back-slashes (for example, root/sub.)

Example

```
#include "wincalls.h"

MENU PenSelect
BEGIN
       MENUITEM "black pen", BLACK_PEN
END
```

Files included in resource script files constants that use #define statements may not include any casting of those constants that are used in the resource script. The resource compiler does not parse this casting syntax. For example, the following statement may not be included:

```
#define IDBUTTON1  (USHORT) 3
```

If casting is required for C source compilation, you may use two statements such as:

```
#define IDBUTTON1    3
#define CSRC_IDBUTTON1    ((USHORT)IDBUTTON1)
```

#define name value

This directive assigns the given value to **name**. All subsequent occurrences of **name** are replaced by the value.

name is any combination of letters, digits, or punctuation.

value is any integer, character string, or line of text.

Example

```
#define     nonzero        1
#define     USERCLASS   "MyControlClass"
```

#undef name

This directive removes the current definition of **name**. All subsequent occurrences of **name** are processed without replacement.

name is any combination of letters, digits, or punctuation.

Example

```
#undef      nonzero
#undef      USERCLASS
```

#ifdef name

This directive performs a conditional compilation of the resource file by checking the specified name. If the name has been defined using a #define directive, #ifdef directs the resource compiler to continue with the statement immediately after it. If the name has not been defined, #ifdef directs the compiler to skip all statements up to the next #endif directive.

name is the name to be checked by the directive.

Example

```
#ifdef Debug
FONT 4 errfont.fnt
#endif
```

#ifndef name

This directive performs a conditional compilation of the resource file by checking the specified name. If the name has not been defined or if its definition has been removed using the #undef directive, #ifndef directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after the #endif. If the name is defined, #ifndef directs the compiler to skip to the next #endif, #else, or #elif directive.

name is the name to be checked by the directive.

Example

```
#ifndef Optimize
FONT 4 errfont.fnt
#endif
```

#if constant expression

This directive performs a conditional compilation of the resource file by checking the specified constant-expression. If the constant-expression is nonzero, #if directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after the #endif. If the constant-expression is zero, #if directs the compiler to skip to the next #endif, #else, or #elif directive.

constant expression is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

```
#if Version<3
FONT 4 errfont.fnt
#endif
```

#elif constant expression

This directive marks an optional clause of a conditional compilation block defined by an #ifdef, #ifndef, or #if directive. The directive carries out conditional compilation of the resource file by checking the specified constant-expression. If the constant-expression is nonzero, #elif directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after the #endif. If the constant-expression is zero, #elif directs the compiler to skip to the next #endif, #else, or #elif directive. Any number of #elif directives can be used in a conditional block.

constant expression Is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

```
#if Version<3
FONT 4 italic.fnt
#elif Version<7
FONT 4 bold.fnt
#endif
```

#else

This directive marks an optional clause of a conditional compilation block defined by an #ifdef, #ifndef, or #if directive. The #else directive must be the last directive before #endif.

Example

```
#ifdef Debug
FONT 4 italic.fnt
#else
FONT 4 bold.fnt
#endif
```

#endif

This directive marks the end of a conditional compilation block defined by an #ifdef, #ifndef, or #if directive. One #endif is required for each #ifdef, #ifndef, and #if directive.

Multiple-Line Statements

Code Page Flagging

The CODEPAGE statement may be placed within the source, to set the code page used for these resources:

STRINGTABLE
ACCELTABLE
MENU
DIALOGTEMPLATE and WINDOWTEMPLATE.

The CODEPAGE statement cannot be encoded within any other statement. All items following a CODEPAGE statement are assumed to be in that code page. The code page is encoded in the resource, and the data in the resource is assumed to be in the specified code page. However, no checking is performed.

These code pages can be specified:

437
850
860
863
865.

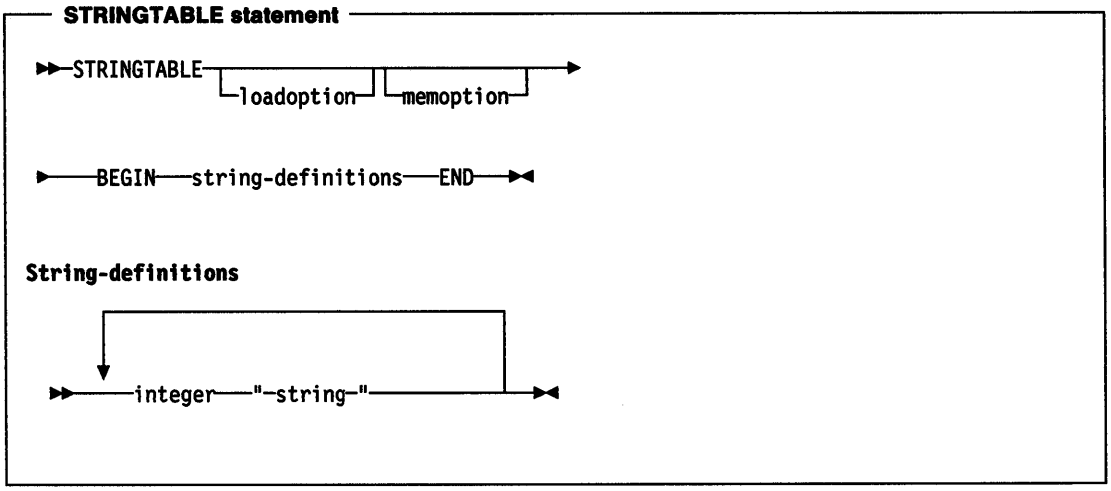
If the code page is not specified, code page 850 is assumed.

STRINGTABLE Statement

The STRINGTABLE statement defines one or more string resources for an application. String resources are null-terminated ASCII strings that can be loaded, when needed, from the executable file, using the WinLoadString function.

Note: The ASCII strings can include no more than 256 characters, including the NULL termination character.

The STRINGTABLE statement has the form:



loadoption *(LDOPT)*
An optional keyword specifying when the resource is to be loaded. It must be one of:

PRELOAD	Resource is loaded immediately.
LOADONCALL	Resource is loaded when called.

The default is LOADONCALL.

memoption *(MEMOPT)*
Consists of the following keyword or keywords, specifying whether the resource is fixed or movable and whether it is discardable:

FIXED	Resource remains at a fixed memory location.
MOVEABLE	Resource can be moved if necessary to compact memory.
DISCARDABLE	Resource can be discarded if no longer needed.

The default is MOVEABLE and DISCARDABLE.

string *(STR)*
A string, enclosed in double quotation marks. To insert a double-quote character (") in the text, use two double-quote characters ("").

Note: A string may be defined on more than one line if each line begins and ends with a double-quote. If newline characters are desired after each line, there should be a double-quote at the beginning of the first line and at the end of the last line only.

The string may contain any ASCII characters. Because (\) is interpreted as an escape character, use (\\) to generate a (\).

The following escape sequences may be used:

Escape Sequence	**Name**
\t	Horizontal tab
\a	Bell (alert)
\nnn	ASCII character (octal)
\xdd	ASCII character (hexadecimal).

The sequences \ddd and \xdd allow any character in the ASCII character set to be inserted in the character string. Thus, the horizontal tab could be entered as \X09, \011 or \t.

Example

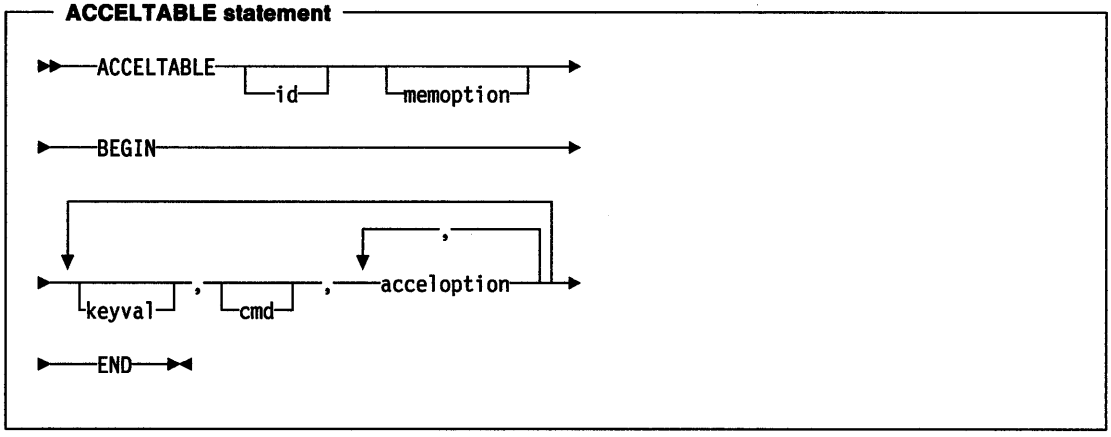
```
#define IDS_STRING1  1
#define IDS_STRING2  2
#define IDS_STRING3  3

STRINGTABLE
BEGIN
  IDS_STRING1, "The first two strings in this table are identical."
  IDS_STRING2, "The first two strings "
               "in this table are identical."
  IDS_STRING3, "This string will contain a newline character
                before it continues on this line."
END
```

ACCELTABLE Statement

The ACCELTABLE statement defines a table of *accelerator* keys for an application.

An accelerator is a keystroke defined by the application to give the user a quick way to perform a task. The WinGetMsg function automatically translates accelerator messages from the application queue into WM_COMMAND, WM_HELP, or WM_SYSCOMMAND messages.

The ACCELTABLE statement has the form:



id *(USHORT)*
The resource identifier.

memoption
Optional. It consists of the following keyword or keywords, specifying whether the resource is fixed or movable, and whether it can be discarded:

FIXED	Resource remains at a fixed memory location.
MOVEABLE	Resource can be moved if necessary to compact memory.
DISCARDABLE	Resource can be discarded if no longer needed.

keyval *(USHORT)*
The accelerator character code. This can be either a constant or a quoted character. If it is a quoted character, the CHAR acceloption is assumed. If the quoted character is preceded with a caret character (^), a control character is specified as if the CONTROL **acceloption** had been used.

cmd *(USHORT)*
The value of the WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message generated from the accelerator for the indicated key.

acceloption *(BIT_16)*
Defines the kind of accelerator.

These options are available:

ALT

CHAR
CONTROL
HELP
LONEKEY
SCANCODE
SHIFT
SYSCOMMAND
VIRTUALKEY.

The VIRTUALKEY, SCANCODE, LONEKEY, and CHAR acceloptions specify the type of message that matches the accelerator. Only one of these options can be specified for each accelerator. For information on the corresponding KC_* values, see "WM_CHAR" on page 12-24.

The **acceloptions** SHIFT, CONTROL, and ALT, cause a match of the accelerator only if the corresponding key is down.

If there are two accelerators that use the same key with different SHIFT, CONTROL, or ALT options, the more restrictive accelerator should be specified first in the table. For example, Shift-Enter should be placed before Enter.

The SYSCOMMAND **acceloption** causes the keystroke to be passed to the application as a WM_SYSCOMMAND message. The HELP **acceloption** causes the keystroke to be passed to the application as a WM_HELP message. If neither is specified, a WM_COMMAND message is used.

Example

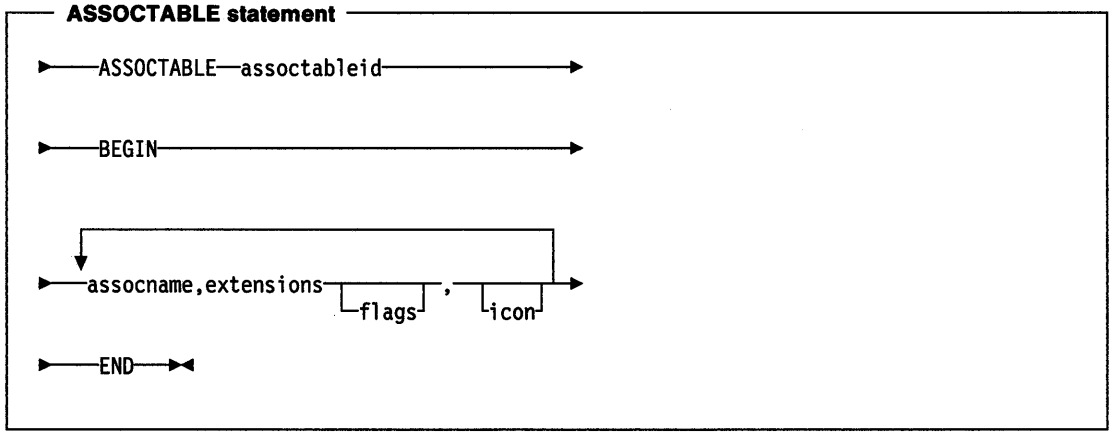
```
ACCELTABLE MainAcc
BEGIN
     VK_F1,101,HELP
     VK_F3,102,SYSCOMMAND
END
```

This generates a WM_HELP with value 101 from VIRTUALKEY accelerator F1 and a WM_SYSCOMMAND with value 102 from VIRTUALKEY accelerator F3.

ASSOCTABLE Statement

The ASSOCTABLE statement defines the extended attributes (EA) for an application.

The ASSOCTABLE statement has the form:



The source for the ASSOCTABLE description is contained in the resource file for a particular project:

```
ASSOCTABLE assoctableid
BEGIN
"association name", "extension", flags, icon filename
"association name", "extension", flags, icon filename
...
END
```

association name Program recognizes data files of this EA TYPE. This is the same name found in the TYPE field of data files.

assoctableid A name or number used to identify the assoctable resource.

extension 3 letter file extension that is used to identify files of this type if they have no EA TYPE entry. (This may be empty.)

flags

EAF_DEFAULTOWNER
The default application for the file.
EAF_UNCHANGEABLE
This flag is set if the entry in the ASSOCTABLE is not to be edited.
EAF_REUSEICON
This flag is specified if a previously defined icon in the ASSOCTABLE is to be reused. Entries with this flag set have no icon data defined. The icon used for this entry is the icon used for the previous entry (see below). Note that EAF_* flags may be ORed together when specified in the ASSOCTABLE.

```
ASSOCTABLE 3000
BEGIN
"Product XYZ Spreadsheet", "xys", EAF_DEFAULTOWNER,
                                          xyzspr.ico
"Product XYZ Chart", "xyc", EAF_DEFAULTOWNER |
                                    EAF_REUSEICON
END
```

icon filename Filename of the icon used to represent this file type. (This may be empty.)

DEFAULTICON Keyword

This keyword installs the filename.ico icon definition under the ICON EA of the program file.

Example

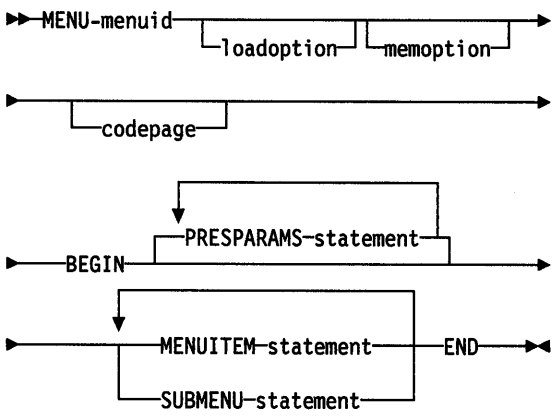
```
DEFAULTICON <filename.ico>
```

MENU Statement

The MENU statement defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu can be used to create an action bar.

The MENU statement has the form:

MENU statement



MENUITEM statement
See page 32-13

SUBMENU statement
See page 32-14

PRESPARAMS statement
See page 32-22

menuid *(USHORT)*
A name or number used to identify the menu resource.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE.

codepage *(USHORT)*
The code page of the text.

MENUITEM
A special resource statement used to define the items in the menu. These are discussed in more detail in "Menu Item Definition Statements" on page 32-13.

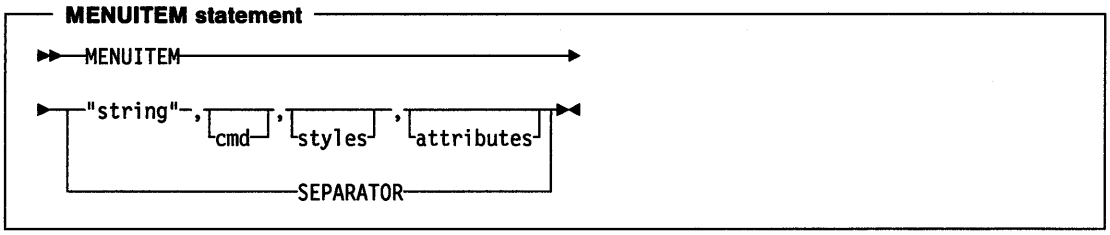
Example: This is an example of a complete MENU statement:

```
MENU sample
BEGIN
 MENUITEM "~Alpha", 100, MIS_TEXT
 SUBMENU  "~Beta", 101, MIS_TEXT
 BEGIN
     MENUITEM "~Green", 200, MIS_TEXT
     MENUITEM "~Blue", 201, MIS_TEXT,MIA_CHECKED
 END
END
```

Menu Item Definition Statements

MENUITEM statements are used in the item-definition section of a MENU statement to define the names and attributes of the actual menu items. Any number of statements can be given; each defines a unique item. The order of the statements defines the order of the menu items.

Note: The MENUITEM statements can only be used within an item-definition section of a MENU statement.



string *(STR)*
A string, enclosed in double quotation marks, specifying the text of the menu item.

To insert a double-quote character (") in the text, use two double-quote characters ("").

If the **styles** parameter does not contain MIS_TEXT, the string is ignored but must still be specified. An empty string ("") should be specified in this instance.

To indicate the mnemonic for each item, insert the tilde character (~) in the string preceding the mnemonic character.

For MENUITEM statements within a SUBMENU (that is, pull-down menus) text may be split into a second column with an alignment substring. To right-align items insert "\a" in the text where alignment should begin. To left-align a second column of text insert "\t" in the text where alignment should begin. For each SUBMENU the longest item in the second column determines the width of that column. Only one alignment substring should be used in a menu item.

cmd *(USHORT)*
The value of the WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message generated by the item when it is selected. It identifies the selection made and should be unique within one menu definition.

styles *(BIT_16)*
One or more menu options defined by the MIS_* constants, ORed together with the | operator. For definitions of the MIS_* constants, see "Menu Item Styles" on page 17-2.

attributes *(BIT_16)*
One or more menu options defined by the MIA_* constants, ORed together with the | operator. For definitions of the MIA_* constants, see page 17-2.

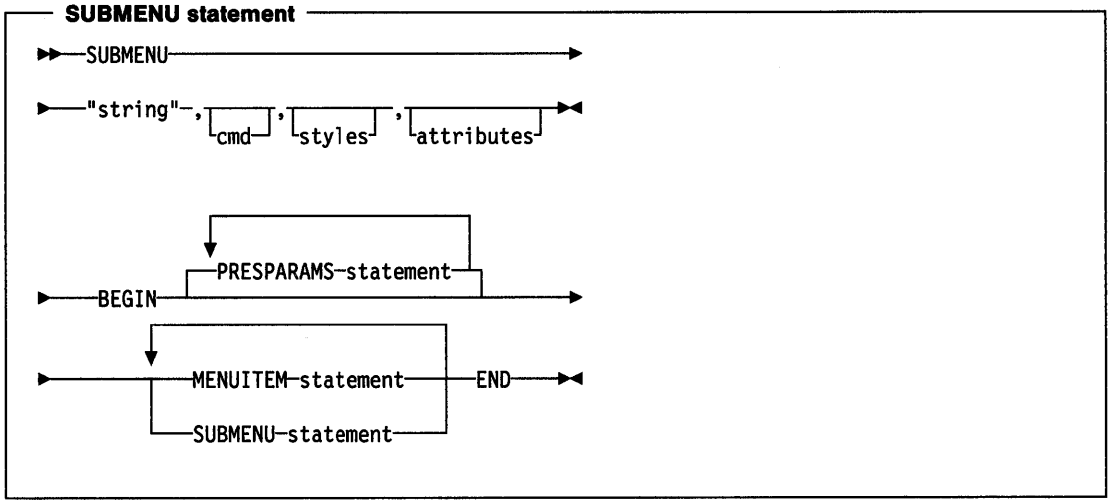
The style MIS_SUBMENU must not be used with this statement. See the SUBMENU statement on page 32-14.

Examples

```
MENUITEM  "Alpha", 1, MIS_TEXT,MIA_ENABLED|MIA_CHECKED,'A'
MENUITEM  "Beta", 2, MIS_TEXT,,'B'
```

Pull-Down Menus or Submenus

In addition to simple items, a menu definition can contain the definition of a submenu. A submenu can itself invoke a lower level submenu.



string *(STR)*

A string, enclosed in double quotation marks, specifying the text of the menu item.

To insert a double-quote character (") in the text, use two double-quote characters ("").

If the **styles** parameter does not contain MIS_TEXT, the string is ignored but must still be specified. An empty string ("") should be specified in this instance.

cmd *(USHORT)*

The value of the WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message generated by the item when it is selected. It identifies the selection made and should be unique within one menu definition.

styles *(BIT_16)*

One or more menu options defined by the MIS_ constants, ORed together with the | operator.

In the SUBMENU statement, the style MIS_SUBMENU is always ORed with the styles given. If no value is supplied, the default value of MIS_TEXT and MIS_SUBMENU is used.

attributes *(BIT_16)*

One or more menu options defined by the MIA_ constants, ORed together with the | operator.

Example

```
MENU chem
BEGIN

SUBMENU  "~Elements", 2, MIS_TEXT
BEGIN
     MENUITEM "~Oxygen", 200, MIS_TEXT
     MENUITEM "~Carbon", 201, MIS_TEXT,MIA_CHECKED
     MENUITEM "~Hydrogen", 202, MIS_TEXT
END

SUBMENU  "~Compounds", 3, MIS_TEXT
BEGIN
     MENUITEM "~Glucose", 301, MIS_TEXT
     MENUITEM "~Sucrose", 302, MIS_TEXT,MIA_CHECKED
     MENUITEM "~Lactose", 303, MIS_TEXT|MIS_BREAK
     MENUITEM "~Fructose", 304, MIS_TEXT
END

END
```

SEPARATOR Menu Item

There is a special form of the MENUITEM statement that is used to create a horizontal dividing bar between two active menu items in a pull-down menu. The SEPARATOR menu item is itself inactive and has no text associated with it nor a **cmd** value.

Example

```
MENUITEM  "~Roman", 206, MIS_TEXT
MENUITEM  SEPARATOR
MENUITEM  "20 ~Point", 301, MIS_TEXT
```

Menu Template

Menu templates are data structures used to define menus. Menu templates can be loaded as resources or created dynamically, or embedded in dialog templates, which in turn can be loaded as resources or created dynamically. Templates loaded as resources cannot contain references to bit maps or owner-drawn items. A menu template consists of a sequence of variable-length records. Each record in a menu template defines a menu item. If a menu item contains a reference to a submenu, the menu template that defines that submenu is placed after the definition of that particular menu item.

Template Format: A menu template has this format:

Length *(USHORT)*
The length of the menu template.

Version *(USHORT)*
The template version. Versions 0 and 1 are valid.

Code page *(USHORT)*
The identifier of the code page used for the text items within the menu (but not any submenus, which each have their own code pages).

Item offset *(USHORT)*
The offset of the items from the start of the template, in bytes.

Count *(USHORT)*
The count of menu items.

Presentation parameters offset *(USHORT)*
Offset of presentation parameters from the start of the template, in bytes. This field is only present for version 1 of the template.

Menu Items
A variable-sized array of menu items as follows:

Style *(USHORT)*
Menu item styles (MIS_*; see page 17-2) combined with the logical-OR operator.

Attributes *(USHORT)*
Menu item attributes (MIA_*; see page 17-2) combined with the logical-OR operator.

Item *(IDENTITY)*
An application-provided identifier for the menu item.

Variable data
Following the identifier is a variable data structure whose format depends upon the value of **Style**:

MIS_TEXT

Text *(STRL)*
Null-terminated text string.

MIS_SUBMENU
A menu template structure.

MIS_BITMAP

Text *(STR)*
Null-terminated text string.

For MIS_BITMAP menu items, the item text string can be used to derive the resource identifier from which a bit map is loaded. There are three instances:

- The first byte is null; that is, no resource is defined and it is assumed that the application subsequently provides a bit-map handle for the item.
- The first byte is X'FF', the second byte is the low byte of the resource identifier, and the third byte is the high byte of the resource identifier.
- The first character is "#," and subsequent characters make up the decimal text representation of the resource identifier.

The resource is assumed to reside in the resource file of the current process.

If the string is empty or does not follow the format above, no resource is loaded.

DLGTEMPLATE and WINDOWTEMPLATE Statements

This section describes how to define dialog and window templates.

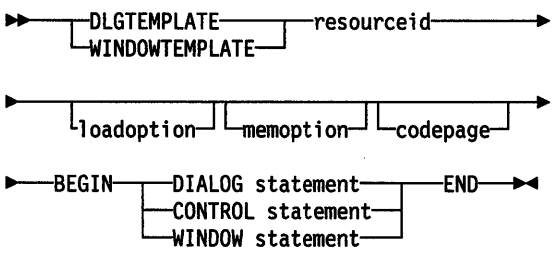
It also describes the control data and presentation parameter structures that the application needs to create windows and define dialog templates.

Data types are shown after each parameter or option. These are the data types that the parameter or option is converted to when it is compiled.

DLGTEMPLATE and WINDOWTEMPLATE statements are used by an application to create predefined window and dialog resource templates.

The DLGTEMPLATE and WINDOWTEMPLATE statements are treated identically by the resource compiler and have this format:

DLG and WINDOW TEMPLATE



The parts of the DLGTEMPLATE and WINDOWTEMPLATE statements are described below.

Purpose
This statement marks the beginning of a window template. It defines the name of the window, and its memory and load options.

resourceId *(USHORT)*
Either a unique name or an integer number identifying the resource.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE.

code page *(USHORT)*
The code page of the text in the template.

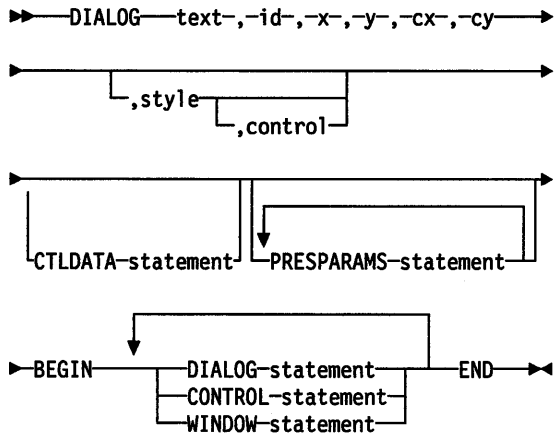
Alternatively, ({{ can be used in place of BEGIN and {}) in place of END.

The DLGTEMPLATE and WINDOWTEMPLATE keywords are synonymous.

The DIALOG statement defines a dialog-box window that can be created by an application.

The DIALOG statement has the format:

DIALOG statement

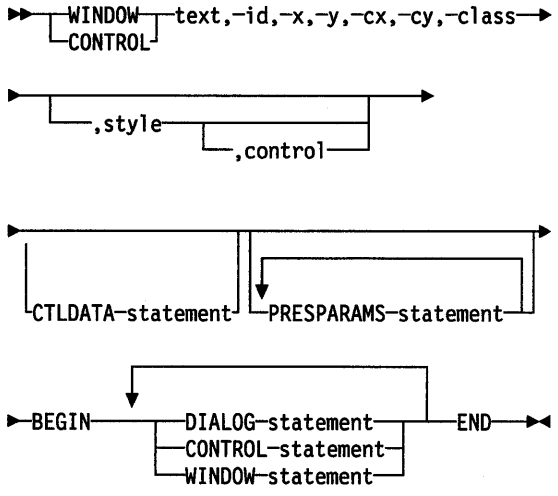


CTLDATA statement
See page 32-22

PRESPARAMS statement
See page 32-22

The WINDOW and CONTROL statements have the format:

WINDOW and CONTROL statements



CTLDATA statement
See page 32-22

PRESPARAMS statement
See page 32-22

Note: The WINDOW and CONTROL keywords are synonymous.

The DIALOG, CONTROL, and WINDOW statements between the BEGIN and END statements are defined as child windows. Presentation parameters always apply to the whole control. They can not be changed for the individual items within the control.

The parameters of these statements are described below.

Purpose.
These statements mark the beginning of a window. They define the starting location on the display screen, its width, its height, and other details such as style.

Note: Not all values may be specified for each statement type. For details, see the call syntax diagrams.

text *(STR)*
A string, enclosed in double quotes, that is displayed in the title-bar control, if it exists. To insert a double-quote character (") in the text, use two double-quote characters ("").

id *(USHORT)*
Item identifier.

x,y *(SHORT)*
Integer numbers specifying the x- and y-coordinates on the display screen of the lower left corner of the dialog. X and y are in dialog coordinates. The exact meaning of the coordinates depends on the style defined by the style argument. For normal dialogs, the coordinates are relative to the origin of the parent window. For FCF_SCREENALIGN style boxes, the coordinates are relative to the origin of the display screen. With FCF_MOUSEALIGN, the coordinates are relative to the position of the pointer at the time the dialog is created.

cx,cy *(SHORT)*
Integer numbers specifying the width and height of the window.

class *(STR)*
The class of the window or control to be created.

Note: For a DIALOG statement the class is fixed as WC_FRAME and cannot be specified.

style *(BIT32)*
Any additional window style, frame style, or other class-specific style.

The default style is WS_SYNCPAINT | WS_CLIPSIBLINGS | WS_SAVEBITS | FS_DLGBORDER . If the FS_DLGBORDER or WS_SAVEBITS styles are not required, they should be preceded by the keyword 'NOT'. For example:

- NOT FS_DLGBORDER | FS_BORDER | NOT WS_SAVEBITS

replaces the FS_DLGBORDER default style by the FS_BORDER style and removes the WS_SAVEBITS style. Note that the logic of the NOT keyword is different from the corresponding operator in the C language.

It is not possible to remove the default WS_SYNCPAINT and WS_CLIPSIBLINGS styles.

control *(BIT32)*
Frame Creation Flags (FCF_*; see page 15-1) for the window

This data is placed in the control data field in the correct format for a window of class WC_FRAME.

Note: FCF_SHELLPOSITION has no effect if specified in a template.

Keyboard Resources

RT_FKALONG (=17), is defined in BSEDOS.H, and the resource compiler (RC.EXE) recognizes **FKALONG**. This type identifies a 256-byte table, that can be used for either primary or secondary scan-code mapping.

The resource ID contains three bytes, the least significant byte identifying the type of scan-code mapping table as follows:

0	Primary scan-code mapping
1	Secondary scan-code mapping.

The other two bytes are 0 for the primary mapping table, and the keyboard ID (as defined in PMWINP.H) for secondary mapping tables. This is to enable simple support to be provided for future keyboards with conflicting scan codes.

The primary scan-code mapping table in the interrupt handler is stored as a resource of this type. The secondary scan-code mapping table in the interrupt handler is also stored as a resource of this type.

Depending on which keyboard is attached, the resources are loaded when the system is initialized, and transferred to RING-0 byte arrays, where they can be accessed by the interrupt handler as necessary. A default primary scan-code mapping table is transferred if the resource cannot be loaded.

Templates, Control Data, and Presentation Parameters

Dialog Template

A dialog template is a data structure used to define a dialog box. Dialog templates can be loaded from resources or created dynamically in memory. Dialog templates define windows of any window class that contain child windows of any class. For standard dialog windows, the dialog window itself is created with the WC_FRAME class, and its children are any of the preregistered control classes.

The dialog template specifies all the information required to create a dialog box and its children.

Dialog Coordinates

Coordinates in a dialog template are specified in *dialog coordinates*. These are based on the default character cell size; a unit in the horizontal direction is 1/4 the default character-cell width, and a unit in the vertical direction is 1/8 the default character-cell height. The origin is the bottom left-hand corner of the dialog box.

Dialog Template Format and Contents

A dialog template has these sections:

Header Defines the type of template format and contains information about the location of the other sections of the template. It also contains a summary of the status of the individual controls contained within the dialog box.

Items Defines each of the controls that comprise the dialog box.

Data area Contains the data values associated with each control. Each control defined in the item section contains pointers to the data area section. The data area also contains presentation parameter definitions. The data area is not necessarily a contiguous portion of the template. User data can be placed anywhere in the template if it does not interfere with other defined information.

The sections of a dialog template are illustrated in Figure 32-1 on page 32-20.

Notes:

1. Throughout the dialog template all lengths are in bytes. String lengths do not include any null terminator that may be present. When strings are passed to the Presentation Interface, the length specifications are used and any null terminators are ignored. When strings are returned by the Presentation Interface, length specifications and null terminators are both supplied; therefore, space must be allowed for a null terminator.
2. All offsets are in bytes from the start of the dialog template structure.

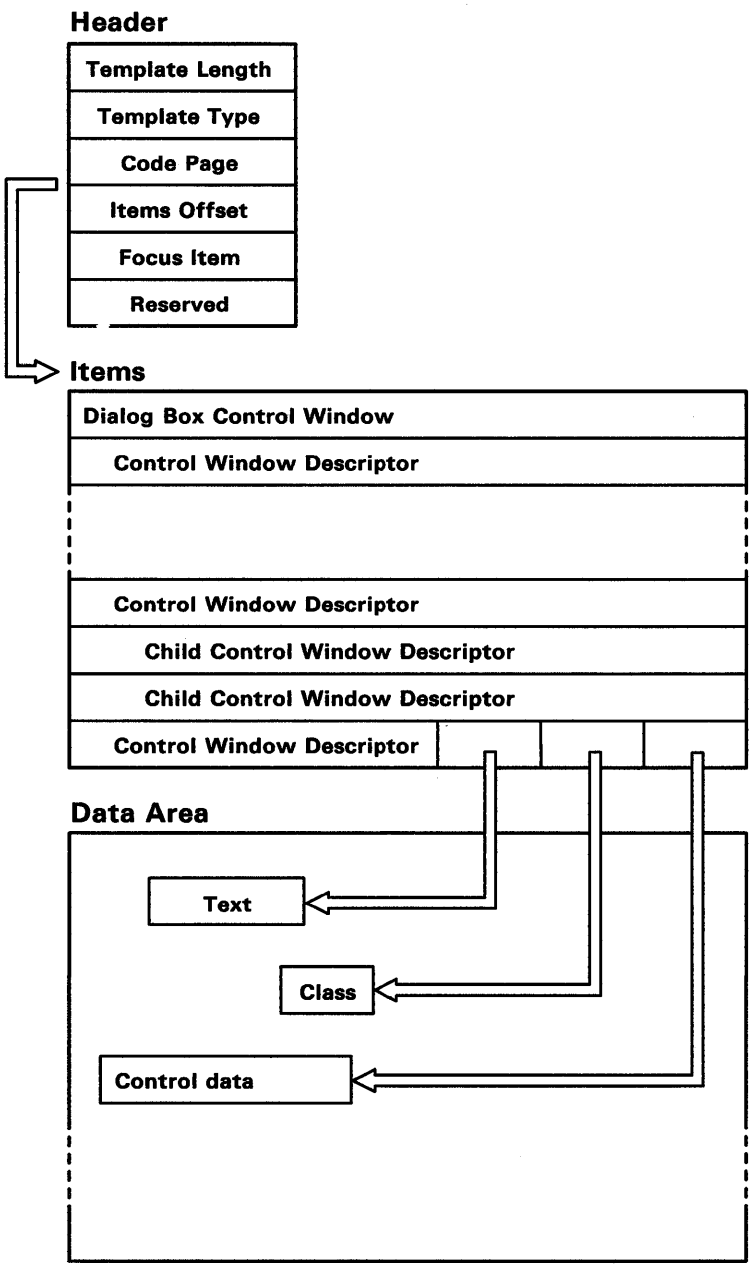


Figure 32-1. Dialog Template

Header

The dialog template header consists of:

Template length *(USHORT)*
The overall length of the dialog template.

Template type *(USHORT)*
The dialog template format type. The format defined is type 0.

Code page *(USHORT)*
The code page of the text in the dialog template.

Items offset *(USHORT)*
The offset of the array of dialog items.

Reserved *(BIT16)*
Must be 0.

Focus item *(USHORT)*
The index in the array of dialog items of the control to receive the focus. If this value is 0, or if the identified control cannot receive the focus, for example because it is a static control, the focus is passed to the first item within the template that can receive the focus.

Reserved *(BIT16)*
Must be 0.

Items

The dialog template items are specified as elements of an array that also defines the hierarchy of the control windows of the dialog box. Each element of the array is a control window descriptor and defines some control or a child of some control, so that every control within the dialog box is described by this array. The first descriptor is the specification of the dialog box itself.

The dialog template items consist of:

Reserved *(BIT16) (16_bit BOOL)*
Must be 0.

Children *(USHORT)*
The number of dialog item child windows that are owned by this dialog item.

This is the number of elements following in the array that are created as child windows of this window. Each window can have any number of child windows, which allows for a tree-structured arrangement.

For example, in Figure 32-1 on page 32-20, assuming that there are no more dialog items than are shown, the first item, the dialog box control window descriptor, has three children. The second item has no children, the third item has two children, and the remaining three items have no children.

Class name length *(USHORT)*
The length of the window class name string.

Class name offset *(USHORT)*
The offset of the window class name string.

Text length *(USHORT)*
The length of the text string.

For controls that allow input of text, this is the current text length, not the maximum text length, and so this value changes when text is put into the control.

Text offset *(USHORT)*
The offset of the text string.

Style *(BIT32) (32_bit BOOL)*
The window style of the control.

The standard style bits are 16 bits. The use of the remaining 16 bits depends on the class of the control.

x *(SHORT)*
y *(SHORT)*
The position of the origin of the dialog item. This is specified in dialog coordinates, with x and y relative to the origin of the parent window.

cx *(SHORT)*
cy *(SHORT)*
The size of the dialog item in dialog coordinates; it must be greater than 0.

Identifier *(USHORT)*
An application-defined identifier for the dialog item.

Reserved *(USHORT)*
Must be zero.

Control data offset *(USHORT)*
The offset of the control-specific data for this dialog item. A value of 0 indicates that there is no control data for this dialog item.

Data Area

The dialog template data area contains the following different types of objects: **text**, **class name**, **presentation parameters**, and **control data**. These objects can be placed anywhere within the data area. They do not have to be in contiguous storage, and so an application can place data for its own use between these objects.

The dialog template data area contains:

Text *(STR)*
The textual data associated with a dialog item.

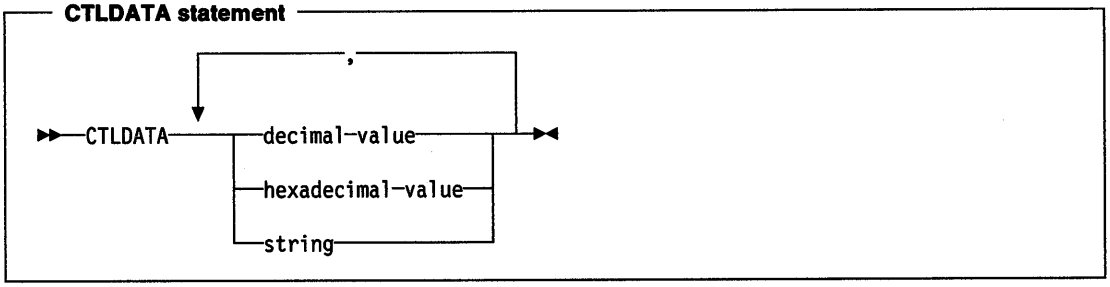
Class name *(STR)*
The name of the window class.

Presentation parameters *(PRESPARAMS)*
Presentation parameters are defined in "Presentation Parameters."

Control data *(CTLDATA)*
For more information, see "Control Data."

Control Data

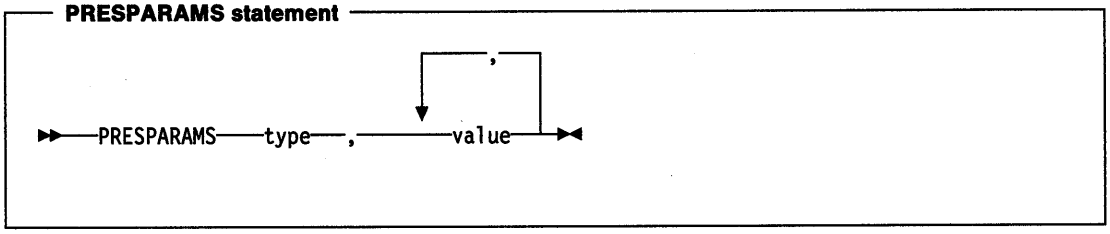
The optional CTLDATA statement is used to define control data for the control. Hexadecimal or decimal word constants follow the CTLDATA statement, separated with commas.



In addition to hexadecimal or decimal data, the CTLDATA statement can be followed by the MENU keyword, followed by a menu template in a BEGIN/END block. This creates a menu template as the control data of the window.

Presentation Parameters

The optional PRESPARAMS statement is used to define presentation parameters. The syntax of the PRESPARAMS statement is as follows.



A presentation parameter consists of:

type *(ULONG)*
The presentation parameter attribute type. See the PARAM data type for a description of valid types.

A string can be used to specify the type for a user type. If this is done, the string type is converted into a string atom when the dialog template is read into memory. Thereafter this presentation parameter is referred to by this string atom. The application can use the atom manager API to match the string and the string atom.

value *(LONG or STRL)*
One or more values depending upon the attribute type.

If the value is enclosed in quotes it is a zero-terminated string. Otherwise, it is converted to a LONG. There may be more than one value, depending upon the type. See PARAM data type for a description of the values required for system-defined presentation parameters.

Examples: The following are examples of PRESPARAMS statements:

```
PRESPARAMS PP_BORDERCOLOR,  0x00ff00ffL
PRESPARAMS PP_FONTNAMESIZE, "12.Helv"
PRESPARAMS "my color",      0x00ff00ffL
PRESPARAMS "my param",      0, 1, 2, 3, "Hi there"
```

Parent/Child/Owner Relationship

The format of the DLGTEMPLATE and WINDOWTEMPLATE resources is very general to allow tree-structured relationships within the resource format. The general layout of the templates is:

```
WINDOWTEMPLATE id
BEGIN
    WINDOW winTop            the top-level window
    BEGIN
        WINDOW wind1
        WINDOW wind2
        WINDOW wind3
        BEGIN
            WINDOW wind4
        END
        WINDOW wind5
    END
END
```

In this example, the top-level window is identified by **winTop**. It has four child windows: **wind1**, **wind2**, **wind3**, and **wind5**. **wind3** has one child window, **wind4**. When each of these windows is created, the parent and the owner are set to be the same.

The only time when the parent and owner windows are not the same is when frame controls are automatically created by a frame window.

Note that the WINDOW statements in the example above could also have been CONTROL or DIALOG statements.

Predefined Window Classes

The CONTROL statement can be used to define a window control of any class. Window classes may be user defined of one of a predefined set provided by the operating system. The following classes are provided in OS/2 Version 2.0.

WC_FRAME Application frame control.
WC_STATIC Text and group boxes.
WC_BUTTON Push button, check box or radio button.
WC_COMBOBOX Combination of an entry field and list box.
WC_ENTRYFIELD Single line entry field.
WC_MLE Multiple line entry field.
WC_LISTBOX List box.
WC_MENU Application action bar, menus and popup menus.
WC_SCROLLBAR Horizontal or vertical scroll bar.
WC_TITLEBAR Application title bar.
WC_SPINBUTTON Spin button entry field.
WC_CONTAINER Container list.
WC_SLIDER Horizontal or vertical slider bar.
WC_VALUESET Value set control.
WC_NOTEBOOK Notebook control.

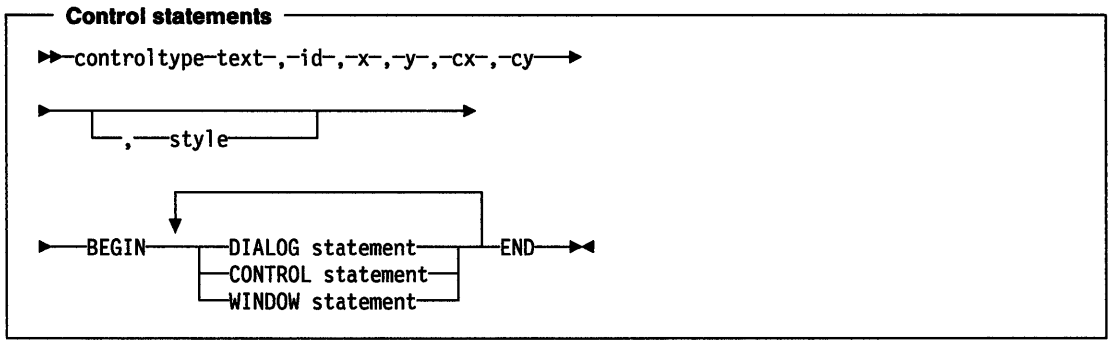
These controls make up the standard user interface components for applications. The following example shows a simple listbox control.

```
CONTROL "", 1, 10, 20, 60, 40, WC_LISTBOX, WS_VISIBLE
```

Predefined Control Statements

In addition to the general form of the CONTROL statement, there are special control statements for commonly used controls. These statements define the attributes of the child control windows that appear in the window.

Control statements have this general form:



The LISTBOX control statement is an exception to this general form because it does not have a text field.

controltype
is one of the keywords described below, defining the type of the control.

text *(STR)*
is a string specifying the text to be displayed. The string must be enclosed in double quotation marks. The manner in which the text is displayed depends on the particular control, as detailed below.

To indicate the mnemonic for each item, insert the tilde character (~) in the string preceding the mnemonic character.

The double quotation marks are required for the COMBOBOX title even if no title is used.

id *(USHORT)*
is a unique integer number identifying the control.

x,y *(SHORT)*
are integer numbers specifying the x- and y-coordinates of the lower left corner of the control, in dialog coordinates. The coordinates are relative to the origin of the dialog.

cx,cy *(SHORT)*
are integer numbers specifying the width and height of the control.

The x, y, cx, and cy fields can use addition and subtraction operators (+ and –). For example, 15 + 6 can be used for the x-field.

Styles can be combined using the (|) operator.

The control type keywords are shown below, with their classes and default styles:

FRAME

Class	WC_FRAME
Default style	WS_VISIBLE

LTEXT

Class	WC_STATIC
Default style	SS_TEXT, DT_LEFT, WS_GROUP, WS_VISIBLE

RTEXT

Class WC_STATIC
Default style SS_TEXT, DT_RIGHT, WS_GROUP, WS_VISIBLE

CTEXT

Class WC_STATIC
Default style SS_TEXT, DT_CENTER, WS_GROUP, WS_VISIBLE

CHECKBOX

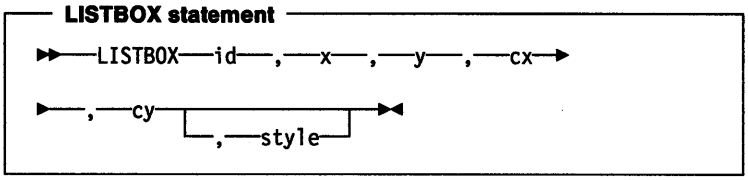
Class WC_BUTTON
Default style BS_CHECKBOX, WS_TABSTOP, WS_VISIBLE

PUSHBUTTON

Class WC_BUTTON
Default style BS_PUSHBUTTON, WS_TABSTOP, WS_VISIBLE

LISTBOX

Format The LISTBOX control statement does not contain a text field, so its form is:



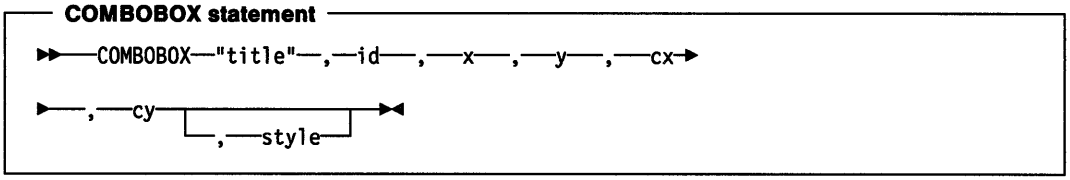
The fields have the same meaning as in the other control statements.

Class WC_LISTBOX
Default style LBS_NOTIFY, LBS_SORT, WS_VSCROLL, WS_BORDER, WS_VISIBLE

COMBOBOX

Format The form of the COMBOBOX control statement is shown below.

The fields have the same meaning as in the other control statements.



Class WC_COMBOBOX
Default style CBS_SIMPLE, WS_TABSTOP, WS_VISIBLE

GROUPBOX

Class WC_STATIC
Default style SS_GROUPBOX, WS_TABSTOP, WS_VISIBLE

DEFPUSHBUTTON

Class WC_BUTTON
Default style BS_DEFAULT, BS_PUSHBUTTON, WS_TABSTOP, WS_VISIBLE

RADIOBUTTON

Class WC_BUTTON
Default style BS_RADIOBUTTON, WS_TABSTOP, WS_VISIBLE

AUTORADIOBUTTON

Class WC_BUTTON
Default style BS_AUTORADIOBUTTON, WS_TABSTOP, WS_VISIBLE

ENTRYFIELD

Class	WC_ENTRYFIELD
Default style	WS_TABSTOP, ES_LEFT, WS_VISIBLE

ICON

Class	WC_STATIC
Default style	SS_ICON, WS_VISIBLE

Examples: The following is a complete example of a DIALOG statement:

```
DLGTEMPLATE errmess
BEGIN
    DIALOG  "Disk Error", 100, 10, 10, 300, 110
    BEGIN
        CTEXT "Select One:", 1, 10, 80, 280, 12
        RADIOBUTTON "Retry", 2, 75, 50, 60, 12
        RADIOBUTTON "Abort", 3, 75, 30, 60, 12
        RADIOBUTTON "Ignore", 4, 75, 10, 60, 12
    END
END
```

This is an example of a WINDOWTEMPLATE statement that is used to define a specific kind of window frame. Calling Load Dialog with this resource automatically creates the frame window, the frame controls, and the client window (of class MyClientClass).

```
WINDOWTEMPLATE wind1
BEGIN
 FRAME "My Window", 1, 10, 10, 320, 130, WS_VISIBLE,
       FCF_STANDARD | FCF_VERTSCROLL
 BEGIN
  WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass",
                                                 style
 END
END
```

This example creates a resource template for a parallel dialog identified by the constant **parallel1**. It includes a frame with a title bar, a system menu, and a dialog-style border. The parallel dialog has three auto radio buttons in it.

```
DLGTEMPLATE parallel1
BEGIN
    DIALOG "Parallel Dialog", 1, 50, 50, 180, 110
    CTLDATA FCF_TITLEBAR | FCF_SYSMENU | FCF_DLGBORDER
    BEGIN
        AUTORADIOBUTTON "Retry", 2, 75, 80, 60, 12
        AUTORADIOBUTTON "Abort", 3, 75, 50, 60, 12
        AUTORADIOBUTTON "Ignore", 4, 75, 30, 60, 12
    END
END
```

Resource (.RES) File Specification

The format for the .RES file is:

```
(/TYPE NAME FLAGS SIZE BYTES/)+
```

Where:

TYPE is either a null-terminated string or an ordinal, in which instance the first byte is X'FF' followed by an INT that is the ordinal.

```
/* Predefined resource types */
#define RT_POINTER            1
#define RT_BITMAP             2
#define RT_MENU               4
#define RT_DIALOG             5
#define RT_STRING             6
#define RT_FONTDIR            7
#define RT_FONT               8
#define RT_ACCELTABLE         9
#define RT_DLGINCLUDE        11
#define RT_FKALONG           17
```

NAME is the same format as TYPE. There are no predefined names.

FLAGS is an unsigned value containing the memory manager flags:

```
#define NSTYPE      X'0007'   /* Segment type mask              */
#define NSCODE      X'0000'   /* Code segment                   */
#define NSDATA      X'0001'   /* Data segment                   */
#define NSITER      X'0008'   /* Iterated segment flag          */
#define NSMOVE      X'0010'   /* Moveable segment flag          */
#define NSPURE      X'0020'   /* Pure segment flag              */
#define NSPRELOAD   X'0040'   /* Preload segment flag           */
#define NSEXRD      X'0080'   /* Execute-only (code segment),   */
                              /* or read-only (data segment)    */
#define NSRELOC     X'0100'   /* Segment has relocations        */
#define NSCONFORM   X'0200'   /* Segment has debug info         */
#define NSDPL       X'0C00'   /* 286 DPL bits                   */
#define NSDISCARD   X'1000'   /* Discard bit for segment        */
#define NS32BIT     X'2000'   /* 32-BIT code segment            */
#define NSHUGE      X'4000'   /* Huge memory segment            */
```

SIZE is a LONG value defining how many bytes follow in the resource.

BYTES is the stream of bytes that makes up the resource.

Any number of resources can appear one after another in the .RES file.

Chapter 33. Graphics Orders

This chapter describes the format of the graphics orders.

Graphics orders are used in the following circumstances:

- Using GpiGetData or GpiPutData functions for bulk transfer of part or all of graphics segment data (unless this is simply being copied without being changed).
- Editing segments with GpiQueryElement and GpiElement.
- Generating metafiles (other than through the Presentation Manager API), or examining their contents. The data part of Graphics Data structured fields within the metafile (see "Metafile Data Format" on page G-2) consists of graphics orders.

When primitive or attribute functions (plus certain other functions) are specified at the programming interface, and the drawing mode (see GpiSetDrawingMode) is set to **drawandretain**, graphics orders are constructed and placed in the current graphics segment. One API call often causes a single order to be generated. Sometimes, however, several orders are necessary: an example of this is where a GpiPolyLine call is issued, which specifies more strokes than there is room for, in a single order.

In either case, the order or orders generated by a single API call comprise a single **element**, unless the application specifically starts an element using the GpiBeginElement function. In this case the element consists of all of the orders generated between this and the following GpiEndElement function. A GpiQueryElement function returns the orders that comprise an element; the application may edit these, and return them to the segment with GpiElement. The Begin Element – End Element orders that surround a multi-order element in the segment are never passed between the application and the system on GpiQueryElement and GpiElement functions.

No double word or word alignment can be assumed for orders either within segments or during editing.

Data Types

All data types are in Intel** format, unless noted otherwise.

GBIT1	1-bit field.
GBIT16	16-bit field.
GBIT2	2-bit field.
GBIT32	32-bit field.
GBIT4	4-bit field.
GBIT5	5-bit field.
GBIT6	6-bit field.
GBIT7	7-bit field.
GBIT8	8-bit field.
GCHAR	Signed 1-byte integer value.
GDELPOINT	Offset point structure. **dx** *(GCHAR)* x coordinate offset.

** Trademark of Intel Corporation

dy *(GCHAR)*
y coordinate offset.

GFIXED
Signed integer fraction (16:16). (This can be treated as a GLONG where the value has been multiplied by 65536.)

GFIXEDS
Signed integer fraction (8:8), which can be treated as a GSHORT data type, where the value has been multiplied by 256.

integer *(GCHAR)*
Integral component.

fraction *(GUCHAR)*
Fractional component.

GHBITMAP
Bit-map handle, which is the same as GULONG.

GINDATT
Individual attribute value. For the attribute types color and background color, this is the same as GINDEX3. For the attribute types mix and background color, this is the same as GUCHAR.

GINDEX3
Unsigned 3-byte integer value.

GLENGTH1
1-byte length.

GLENGTH2
2-byte length, in S/370 format; that is, the high-order byte precedes the low-order byte in storage.

GLONG
Signed 4-byte integer value.

GPOINT
Point structure.

x *(GROSOL)*
x coordinate.

y *(GROSOL)*
y coordinate.

GPOINTB
Point in bit-map structure.

x *(GLONG)*
x coordinate.

y *(GLONG)*
y coordinate.

GPOLYS
Array of Polygons. Each element of the array is a 16 bit count of the number of vertices, followed by the vertex coordinates.

GREAL
Real (single precision floating point).

This data type is in Intel format.

GROF
Number representation which is the same as the GFIXED data type.

GROFUFS
Number representation which is either GFIXED, GUFIXEDS or GREAL data type, depending on the presentation-space format.

GROUFS
Number representation which is either the GUFIXEDS or GREAL data type, depending on the presentation-space format.

GROL
Number representation, which is the same as the GLONG data type.

GROSOL
Number representation which is either the GSHORT or the GLONG data type, depending on the presentation-space format; see PS_FORMAT in the *flOptions* parameter of the GpiCreatePS function.

GROUL
Number representation, which is the same as the GULONG data type.

GSHORT
Signed 2-byte integer value.

GSHORT370
Signed 2-byte integer value, in S/370 format (that is, the high-order byte precedes the low-order byte in storage).

GSTR
String with an explicit length count.

GUCHAR
Unsigned 1-byte integer value.

GULONG	Unsigned 4-byte integer value.
GULONG370	Unsigned 4-byte integer value, in S/370 format (that is, the high-order byte first, the low-order byte last in storage).
GUFIXEDS	Unsigned integer fraction (8:8) which can be treated as a GUSHORT data type, where the value has been multiplied by 256.
GUNDF	Undefined string of 8-bit bytes.
GUNDF1	Undefined 8-bit byte.
GUSHORT370	Unsigned 2-byte integer value, in S/370 format; that is, the high-order byte precedes the low-order byte in storage.
GUSHORT	Unsigned 2-byte integer value.

Arc at a Given Position / Arc at Current Position

This order constructs an arc starting at a given position.

Arc at a Given Position (GARC)
X'C6'(len, p_0, p_1, p_2)

Arc at Current Position (GCARC)
X'86'(len, p_1, p_2)

Parameters

len *(GLENGTH1)*
Length of following data.

p_0 *(GPOINT)*
Coordinate data of start point.

This parameter is only present in a Arc at a Given Position Order.

p_1 *(GPOINT)*
Coordinate data of intermediate point.

p_2 *(GPOINT)*
Coordinate data of end point.

Begin Area

This order indicates the start of a set of primitives that define an area boundary.

Begin Area (GBAR)
X'68'(flags)

Parameters

flags
Internal flags.

res1 *(GBIT1)*
Reserved for migration:

1 Only valid value.

boundary *(GBIT1)*
Boundary-line draw indicator:

0 Do not draw boundary lines
1 Draw boundary lines.

inside *(GBIT1)*
Mode shading:

0 Alternate mode
1 Winding mode.

res2 *(GBIT5)*
Reserved.

00000 Only valid value.

Begin Element

This order indicates the beginning of a set of primitives that define an element.

Begin Element (GBEL)
X'D2'(len, type, descr)

Parameters

len *(GLENGTH1)*
Length of following data.

type *(GLONG)*
Element type code.

Values are:

X'0000FD01'	Line bundle
X'0000FD02'	Character bundle
X'0000FD03'	Marker bundle
X'0000FD04'	Area bundle
X'0000FD05'	Image bundle
X'00000007'	Call segment
X'00000081'	Polyline
X'00000085'	Polyfillet
X'000000A4'	Polyfillet sharp
X'000000A5'	Polyspline
X'00000082'	Polymarker
X'00000087'	Full arc
X'00000091'	Image
X'000000B1'	Character string at current position
X'000000F1'	Character string at given position
X'81xxxxxx' – X'FFxxxxxx'	Indicates user defined elements
Other	Reserved values.

descr *(GUNDF)*
Element description data.

This is optional.

Begin Image at Given Position / Begin Image at Current Position

These orders identify the start of an image definition at a given position or at the current position.

Begin Image at Given Position (GBIMG)
X'D1'(len, pe, format, res, width, height)

Begin Image at Current Position (GCBIMG)
X'91'(len, format, res, width, height)

Parameters

len *(GLENGTH1)*
Length of following data.

X'06' Only valid value.

pe *(GPOINT)*
Point at which the image is to be placed.

This parameter is only present in a Begin Image at Given Position order.

format *(GBIT8)*
Format of the image data.

X'00' One bit in the data represents one image point on the usable area.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

width *(GUSHORT370)*
Width of the image data.

This is the width in pels

X'00' – X'07' Valid range of values.

height *(GUSHORT370)*
Height of the image data.

This is the height in pels

Begin Path

This order sets the drawing process into path state.

Begin Path (GBPTH)
X'D0'(len, res, pthid)

Parameters

len *(GLENGTH1)*
Length of following data.

X'06' Only valid value.

res *(GBIT16)*
Reserved.

X'0000' Only valid value.

pthid *(GLONG)*
Path identifier.

X'00000001' – X'FFFFFFFF'

Bezier Curve at Given Position / Bezier Curve at Current Poition

This order generates a curve that starts at a given position.

Bezier Curve at Given Position (GBEZ)
X'E5'(len, p_0, p_1, p_2, p_3, p_4, p_5, p_6, pn-2, pn-1, pn)

Bezier Curve at Current Poition (GCBEZ)
X'A5'(len, p_1, p_2, p_3, p_4, p_5, p_6, pn-2, pn-1, pn)

Parameters

len *(GLENGTH1)*
Length of following data.

p_0 *(GPOINT)*
Coordinate data of first curve start.

This parameter is only present in a Bezier Curve at Given Position.

p_1 *(GPOINT)*
Coordinate data of first curve, first control point.

p_2 *(GPOINT)*
Coordinate data of first curve, second control point.

p_3 *(GPOINT)*
Coordinate data of first curve end.

p_4 *(GPOINT)*
Coordinate data of second curve, first control point

p_5 *(GPOINT)*
Coordinate data of second curve, second control point

p_6 *(GPOINT)*
Coordinate data of second curve end.

pn-2 *(GPOINT)*
Coordinate data of final curve, first control point

pn-1 *(GPOINT)*
Coordinate data of final curve, second control point

pn *(GPOINT)*
Coordinate data of final curve end.

BitbIt

This order copies a rectangle of a bit map into DOCS.

BitbIt (GBBLT)
X'D6'(len, flags, mix, bmid, trans, p1, p2, source1x, source1y, source2x, source2y)

Parameters

len *(GLENGTH1)*
Length of following data.

flags *(GBIT16)*
Reserved.

X'0000'	Only valid value.

mix *(GBIT16)*
Mix mode.

Values are:

X'00CC'	Source.
X'00C0'	Source and pattern.
X'00CA'	Source where pattern1
X'000C'	Source where pattern0
X'00E2'	Pattern where source1
X'00B8'	Pattern where source0
other	Reserved values.

bmid *(GHBITMAP)*
Bit-map identifier.

trans *(GBIT32)*
Transfer mode.

Values are:

X'00000000'	OR
X'01000000'	AND
X'02000000'	Ignore
other	Reserved values.

p1 *(GPOINT)*
Target rectangle bottom-left corner.

p2 *(GPOINT)*
Target rectangle top-right corner.

source1x *(GLONG)*
Source rectangle bottom-left corner, x coordinate.

source1y *(GLONG)*
Source rectangle bottom-left corner, y coordinate.

source2x *(GLONG)*
Source rectangle top-right corner, x coordinate.

source2y *(GLONG)*
Source rectangle top-right corner, y coordinate.

Box at Given Position / Box at Current Position

This order defines a box with square or round corners, drawn with its first corner at a given position.

Box at Given Position (GBOX)
X'C0'(len, control, res, p_0, p_1, haxis, vaxis)

Box at Current Position (GCBOX)
X'80'(len, control, res, p_1, haxis, vaxis)

Parameters

len *(GLENGTH1)*
Length of following data.

control
Internal flags.

res1 *(GBIT1)*
Reserved.

0 Only valid value.

fill *(GBIT1)*
Values:

0 No fill
1 Fill.

boundary *(GBIT1)*
Values:

0 No boundary
1 Boundary.

res2 *(GBIT5)*
Reserved.

00000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

p_0 *(GPOINT)*
Coordinate data of box origin.

This parameter is only present in a Box at Given Position order.

p_1 *(GPOINT)*
Coordinate data of box corner.

haxis *(GROSOL)*
Length of horizontal axis of ellipse.

vaxis *(GROSOL)*
Length of vertical axis of ellipse.

Call Segment

This order calls one segment from another.

Call Segment (GCALLS)
X'07'(len, res, segname)

Parameters

len *(GLENGTH1)*
Length of following data.

X'06' Only valid value.

res *(GBIT16)*
Reserved value.

X'0000' Only valid value.

segname *(GLONG)*
Name of segment that is to be called.

The name cannot be 0.

Character String at Given Position / Character String at Current Position

These orders draw a character string at a given position or at the current position.

Character String at Given Position (GCHST)
X'C3'(len, p_e, cp)

Character String at Current Position (GCCHST)
X'83'(len, cp)

Parameters

len *(GLENGTH1)*
Length of following data.

p_e *(GPOINT)*
Point at which the character string is to be placed.

This parameter is only present in a Character String at Given Position order.

cp *(GSTR)*
Code points of each character in the string.

Character String Extended at Given Position / Character String Extended at Current Position

This order defines a character string to be drawn at a given position.

Character String Extended at Given Position (GCHSTE)
X'FEF0'(len1, p_0, flags, res, p_1, p_2, len2, cp, pad, vect)

Character String Extended at Current Position (GCCHSTE)
X'FEB0'(len1, flags, res, p_1, p_2, len2, cp, pad, vect)

Parameters

len1 *(GLENGTH2)*
Length of following data.

p_0 *(GPOINT)*
Point at which the character string is to be placed.

This parameter is only present in a Character String Extended at Given Position order.

flags
Extra functions:

rect *(GBIT1)*
Values:

0 Do not draw background rectangle
1 Draw background rectangle.

clip *(GBIT1)*
Values:

0 Do not clip to rectangle
1 Clip to rectangle.

res1 *(GBIT1)*
Reserved.

0 Only valid value.

lvcp *(GBIT1)*
Values:

0 Move current position
1 Leave current position.

res2 *(GBIT4)*
Reserved.

0000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

p_1 *(GPOINT)*
Coordinate data of rectangle corner.

p2 *(GPOINT)*
Coordinate data of rectangle corner.

len2 *(GLENGTH2)*
Length of code-point data.

cp *(GSTR)*
Code-point data.

pad *(GBIT8)*
Pad byte.

Only needs to be included if *cp* is an odd number of bytes.

vect *(GROSOL*n)*
Vector of character increments.

n is the number of code points present in the *cp* parameter.

Character String Move at Given Position / Character String Move at Current Position

This order draws a character string starting from a given position and moves the current position to the end of the string.

Character String Move at Given Position (GCHSTM)
X'F1'(len, pe, cp)

Character String Move at Current Position (GCCHSTM)
X'B1'(len, cp)

Parameters

len *(GLENGTH1)*
Length of following data.

pe *(GPOINT)*
Point at which the character string is to be placed.

This parameter is only present in a Character String Move at Given Position order.

cp *(GSTR)*
Code points of each character in the string.

Close Figure

This order delimits the end of a closed figure.

Close Figure (GCLFIG)
X'7D'(res)

Parameters

res *(GBIT8)*
Reserved.

X'00' Only valid value.

Comment

This order enables data to be stored within a segment.

Comment (GCOMT)
X'01'(len, data)

Parameters

len *(GLENGTH1)*
Length of following data.

data *(GBIT8*len)*
Comment data.

Remarks

This order is treated as a no-operation.

End Area

This order indicates the end of a set of primitives that define an area boundary.

End Area (GEAR)
X'60'(len, data)

Parameters

len *(GLENGTH1)*
Length of following data. It is normally 0.

data *(GBIT8*len)*
Reserved.

X'00...' Only valid value.

End Element

This order identifies the end of a set of primitives that define an element.

End Element (GEEL)
X'49'(res)

Parameters

res *(GBIT8)*
Reserved.

X'00' Only valid value.

End Image

This order identifies the end of an image definition.

End Image (GEIMG)
X'93'(len, data)

Parameters

len *(GLENGTH1)*
Length of following data. It is normally 0.

data *(GBIT8*len)*
Reserved.

X'00...' Only valid value.

End of Symbol Definition

This order indicates the end of a set of orders defining a graphics symbol.

End of Symbol Definition (GESD) **X'FF'**

Remarks

This order is only valid in the context of symbol definitions.

End Path

This order ends the definition of a path.

End Path (GEPTH) **X'7F'(res)**

Parameters

res *(GBIT8)*

Reserved.

X'00' Only valid value.

End Prolog

This order indicates the end of the prolog of a segment.

End Prolog (GEPROL) **X'3E'(res)**

Parameters

res *(GBIT8)*

Reserved.

X'00' Only valid value.

Escape

This order provides facilities for registered and unregistered escape functions.

Escape (GESCP)
X'D5'(len, type, rid, parms)

Parameters

len *(GLENGTH1)*
Length of following data.

type *(GBIT8)*
Type identifier:

80 Registered value
Other All other values are unregistered.

rid *(GBIT8)*
Registered identifier:

01 Set pel.
02 BITBLT function.
03 Flood fill function.
04 Draw bits function.

parms *(GSTR)*
Parameters of escape.

Extended Escape

This order provides facilities for registered and unregistered escape functions.

Extended Escape (GEESCP)
X'FED5'(len, type, rid, parms)

Parameters

len *(GLENGTH2)*
Length of following data.

type *(GBIT8)*
Type identifier:

X'80' Registered value
Other All other values are unregistered.

rid *(GUCHAR)*
Registered identifier.

No registered extended escapes are used by OS/2 Version 2.0

parms *(GSTR)*
Parameters of escape.

Fill Path

This order fills the interior of the specified path.

Fill Path (GFPTH)
X'D7'(len, flags, res, pthId)

Parameters

len *(GLENGTH1)*
Length of following data.

X'06' Only valid value.

flags
Extra functions:

res1 *(GBIT1)*
Reserved.

0 Only valid value.

inside *(GBIT1)*
Values:

0 Alternate mode
1 Winding mode.

mod *(GBIT1)*
Values:

0 Do not modify before filling
1 Modify path before filling.

res2 *(GBIT5)*
Reserved.

00000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pthId *(GLONG)*
Path identifier.

X'00000001' – X'FFFFFFFF' Valid path identifiers.

Fillet at Given Position / Fillet at Current Position

These orders draw a curved line tangential to a specified set of straight lines, at the given position or at the current position.

Fillet at Given Position (GFLT)
X'C5'(len, p_0, p_1, p_2, pn)

Fillet at Current Position (GCFLT)
X'85'(len, p_1, p_2, pn)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of line start.

This parameter is only present in a Fillet at Given Position order.

p1 *(GPOINT)*
Coordinate data of first line end.

p2 *(GPOINT)*
Coordinate data of second line end.

pn *(GPOINT)*
Coordinate data of final line end.

Full Arc at Given Position / Full Arc at Current Position

This order constructs a full circle or an ellipse, with the center at a given position.

Full Arc at Given Position (GFARC)
X'C7'(len, p0, m)

Full Arc at Current Position (GCFARC)
X'87'(len, m)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of the center of the circle/ellipse.

This parameter is only present in a Full Arc at Given Position order.

m *(GROFUFS)*
Multiplier.

Image Data

This order provides bit data for an image.

Image Data (GIMD)
X'92'(len, data)

Parameters

len *(GLENGTH1)*
Length of following data.

data *(GBIT8*len)*
Image data.

Label

This order is used to label an element within a segment.

Label (GLBL)
X'D3'(len, ldata)

Parameters

len *(GLENGTH1)*
Length of following data.

X'04' Only valid value.

ldata *(GLONG)*
Label value.

Line at Given Position / Line at Current Position

This order defines one or more connected straight lines, drawn from the given position.

Line at Given Position (GLINE)
X'C1'(len, p_0, p_1, pn)

Line at Current Position (GCLINE)
X'81'(len, p_1, pn)

Parameters

len *(GLENGTH1)*
Length of following data.

p_0 *(GPOINT)*
Coordinate data of line start.

This parameter is only present in a Line at Given Position order.

p_1 *(GPOINT)*
Coordinate data of first line end.

pn *(GPOINT)*
Coordinate data of final line end.

Marker at Given Position / Marker at Current Position

This order draws the current marker symbol at one or more positions starting from a given position.

Marker at Given Position (GMRK)
X'C2'(len, p_0, p_1, pn)

Marker at Current Position (GCMRK)
X'82'(len, p_1, pn)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of first marker.

p1 *(GPOINT)*
Coordinate data of second marker.

pn *(GPOINT)*
Coordinate data of final marker.

Modify Path

This order modifies the path according to the value of the mode.

Modify Path (GMPTH)
X'D8'(len, mode, res, pthid)

Parameters

len *(GLENGTH1)*
Length of following data.

X'06' Only valid value.

mode *(GBIT8)*
Mode of path modification:

X'06' Stroke the path
Other All other values are reserved.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pthid *(GLONG)*
Path identifier.

X'00000001' – X'FFFFFFFF' Valid path identifiers.

No-Operation

This order is a no-operation.

No-Operation (GNOP1)
X'00'

Outline Path

This order draws the outline of the specified path.

Outline Path (GOPTH)
X'D4'(len, flags, res, pthid)

Parameters

len *(GLENGTH1)*
Length of following data.

flags *(GBIT8)*
Function flags:

X'00' Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pthid *(GLONG)*
Path identifier.

1 Only valid value.

Partial Arc at Given Position / Partial Arc at Current Position

This order draws a line from a given position to the start of an arc, and then draws the arc.

Partial Arc at Given Position (GPARC)
X'E3'(len, p_0, p_1, m, start, sweep)

Partial Arc at Current Position (GCPARC)
X'A3'(len, p_1, m, start, sweep)

Parameters

len *(GLENGTH1)*
Length of following data.

p_0 *(GPOINT)*
Coordinate data of start of line.

This parameter is only present in a Partial Arc at Given Position order.

p_1 *(GPOINT)*
Coordinate data of center of arc.

m *(GROFUFS)*
Multiplier.

start *(GROF)*
Start angle.

sweep *(GROF)*
Sweep angle.

Polygons

This order defines a set of polygons, which are optionally filled.

Polygons (GPOLYS)
X'F3'(len, flags., count, polys)

Parameters

len *(GLENGTH2)*
Length of following data.

flags.
Internal flags.

inside *(GBIT1)*
Mode shading:

0 Alternate mode.
1 Winding mode.

model *(GBIT1)*
Drawing model:

0 The fill is inclusive of bottom right.
1 The fill is exclusive of bottom right.

res2 *(GBIT6)*
Reserved.

000000 Only valid value.

count *(GUSHORT)*
Number of polygons

polys *(GPOLYS)*
Array of polygons

Remarks

This order draws a set of polygons. For the first polygon the current position is the first point. For all subsequent polygons all points which define the polygon are given explicitly. The polygons are automatically closed if necessary.

The current position is set to the last point specified.

Pop

This order enables data to be popped from the Segment Call Stack.

Pop (GPOP)
X'3F'(res)

Parameters

res *(GBIT8)*
Reserved.

X'00' Only valid value.

Remarks

The data is placed into an attribute or Drawing Process Control.

Relative Line at Given Position / Relative Line at Current Position

These orders define one or more connected straight lines, at the given position or at the current position.

Relative Line at Given Position (GRLINE)
X'E1'(len, p_0, off_0, off_1, offn)

Relative Line at Current Position (GCRLINE)
X'A1'(len, off_0, off_1, offn)

Parameters

len *(GLENGTH1)*
Length of following data.

p_0 *(GPOINT)*
Coordinate data of line start.

This parameter is only present in a Relative Line at Given Position order.

off_0 *(GDELPOINT)*
Offset data for first point.

This offset is to the first line end, relative to its start point.

off_1 *(GDELPOINT)*
Offset data for second point.

This offset is to the second line end, relative to the first line end.

offn *(GDELPOINT)*
Offset data for final point.

This offset is to the nth line end, relative to the n-1th line end.

Remarks

The end point of each line is given as an offset from the start of the line, rather than as absolute coordinates.

Segment Characteristics

This order provides the facility to set architected or user-defined characteristics for a segment.

Segment Characteristics (GSGCH)
X'04'(len, cbit8, parms)

Parameters

len *(GLENGTH1)*
Length of following data.

cbit8 *(GUCHAR)*
Identification code for characteristics:

X'00' – X'7F' Reserved for architected characteristics.
X'80' – X'FF' Reserved for user-defined characteristics.

parms *(GSTR)*
Parameters of characteristics.

Remarks

The order is only valid in a root-segment prolog.

Set Arc Parameters / Push and Set Arc Parameters

These orders set, or push and set, the values of the current arc parameters.

Set Arc Parameters (GSAP)
X'22'(len, p, q, r, s)

Push and Set Arc Parameters (GPSAP)
X'62'(len, p, q, r, s)

Parameters

len *(GLENGTH1)*
Length of following data.

p *(GROSOL)*
P-parameter of arc transform.

q *(GROSOL)*
Q-parameter of arc transform.

r *(GROSOL)*
R-parameter of arc transform.

s *(GROSOL)*
S-parameter of arc transform.

Remarks

The values of the current arc parameters are pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the current arc parameters to the values specified in the order.

The value of these parameters determines the shape of subsequent orders drawn using Arc at a Given Position / Arc at Current Position or Full Arc at Given Position / Full Arc at Current Position or Partial Arc at Given Position / Partial Arc at Current Position.

Set Background Color / Push and Set Background Color

These orders set, or push and set, the value of the current background color attribute.

Set Background Color (GSBCOL)
X'25'(len, color)

Push and Set Background Color (GPSBCOL)
X'65'(len, color)

Parameters

len *(GLENGTH1)*
Length of following data.

X'02' Only valid value.

color *(GBIT16)*
Color-tabie index:

Except for the special values, the values X'0000' through X'nnnn' are allowed color indexes; that is, as many values as are allowed by the size of the LCT.

Special Values

X'0000' Drawing default
X'0007' White
X'0008' Black
X'FF00' Drawing default
X'FF0x' Color indexes X'000x', where x is in the range 1 through 7.
X'FF08' Color index 0 (reset color).

Set Background Indexed Color / Push and Set Background Indexed Color

These orders set, or push and set, the value of the current background color attribute.

Set Background Indexed Color (GSBICOL)
X'A7'(len, flags, index)

Push and Set Background Indexed Color (GPSBICOL)
X'E7'(len, flags, index)

Parameters

len *(GLENGTH1)*
Length of following data.

X'04' Only valid value.

flags
Values:

default *(GBIT1)*
Options:

0 Use specified *index*
1 Use drawing default color.

spec *(GBIT1)*
Options:

0 Use index directly
1 Special value.

res *(GBIT6)*
Reserved.

000000 Only valid value.

index *(GINDEX3)*
Value for color index.

The value is a direct index into the current color table or a special value.

The special values are:

1 Black
2 White
4 All ones
5 All zeros.

Remarks

The value of the current background color attribute is pushed on to the stack by the Push and Set order only. Both orders then set the current background color attribute to the value specified in the order.

Set Background Mix / Push and Set Background Mix

These orders set, or push and set, the value of the current background mix attribute.

Set Background Mix (GSBMX)
X'0D'(mode)

Push and Set Background Mix (GPSBMX)
X'4D'(mode)

Parameters

mode *(GBIT8)*

Mix-mode value:

X'00' Drawing default
X'01' OR
X'02' Overpaint
X'03' Reserved
X'04' Exclusive-OR
X'05' Leave Alone
X'06' AND
X'07' Subtract
X'08' Source AND (inverse destination)
X'09' All zeros
X'0A' Inverse (source OR destination)
X'0B' Inverse (source XOR destination)
X'0C' Inverse destination
X'0D' Source OR (inverse destination)
X'0E' Inverse source
X'0F' (Inverse source) OR destination
X'10' Inverse (source AND destination)
X'11' All ones.

Remarks

The value of the current background mix attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the current background mix attribute to the value specified in the order.

Set Character Angle / Push and Set Character Angle

These orders set, or push and set, the value of the current character angle attribute.

Set Character Angle (GSCA)
X'34'(len, ax, ay)

Push and Set Character Angle (GPSCA)
X'74'(len, ax, ay)

Parameters

len *(GLENGTH1)*
Length of following data.

ax *(GROSOL)*
X coordinate of point.

This point defines the angle of the character string.

ay *(GROSOL)*
Y coordinate of point.

This point defines the angle of the character string.

Remarks

The value of the current character angle attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character angle to the value specified in the order.

Set Character Break Extra / Push and Set Character Break Extra

These orders set, or push and set, the value of the current character break extra attribute.

Set Character Break Extra (GSCBE)
X'05'(len, flags, res2, inc)

Push and Set Character Break Extra (GPSCBE)
X'45'(len, flags, res2, inc)

Parameters

len *(GLENGTH1)*
Length of following data.

flags
Values as follows:

default *(GBIT1)*
Values as follows:

B'0' Set to specified value.
B'1' Set to drawing default.

res1 *(GBIT7)*
Reserved.

B'0000000' Only valid value.

res2 *(GUNDF1)*
Reserved.

X'00' Only valid value.

inc *(GROF)*
Increment.

Remarks

The value of the current character break extra attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character break extra attribute to the value specified in the order.

Set Character Cell / Push and Set Character Cell

These orders set, or push and set, the value of the current character cell-size attribute.

Set Character Cell (GSCC)
X'33'(len, cellx, celly, cellxf, cellyf, flags, res)

Push and Set Character Cell (GPSCC)
X'03'(len, cellx, celly, cellxf, cellyf, flags, res)

Parameters

len *(GLENGTH1)*
Length of following data.

cellx *(GROSOL)*
X part of character cell-size attribute.

celly *(GROSOL)*
Y part of character cell-size attribute.

cellxf *(GUSHORT)*
Fractional X part of character cell-size attribute.

This parameter is optional.

cellyf *(GUSHORT)*
Fractional Y part of character cell-size attribute.

This parameter must be present if *cellxf* parameter is present.

flags
Internal flags.

This parameter is optional.

notdefit *(GBIT1)*
Values:

0 A cell size of zero sets drawing default
1 A cell size of zero sets to zero.

res *(GBIT7)*
Reserved.

0000000 Only valid value.

res *(GBIT8)*
Reserved value.

This parameter must be present if *flags* parameter is present.

X'00' Only valid value.

Remarks

The value of the current character cell-size attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character cell-size attribute to the value in the order.

Set Character Direction / Push and Set Character Direction

These orders set, or push and set, the value of the current character direction attribute.

Set Character Direction (GSCD)
X'3A'(direction)

Push and Set Character Direction (GPSCD)
X'7A'(direction)

Parameters

direction *(GBIT8)*
Value for character direction:

All other values are reserved.

X'00' Drawing default
X'01' Left to right
X'02' Top to bottom
X'03' Right to left
X'04' Bottom to top.

Remarks

The value of the current character direction attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character direction attribute to the value in the order.

Set Character Extra / Push and Set Character Extra

These orders set, or push and set, the value of the current character extra attribute.

Set Character Extra (GSCE)
X'17'(len, flags, res2, inc)

Push and Set Character Extra (GPSCE)
X'57'(len, flags, res2, inc)

Parameters

len *(GLENGTH1)*
Length of following data.

flags
Values as follows:

default *(GBIT1)*
Values as follows:

B'0' Set to specified value.
B'1' Set to drawing default.

res1 *(GBIT7)*
Reserved.

B'0000000' Only valid value.

res2 *(GUNDF1)*
Reserved.

X'00' Only valid value.

inc *(GROF)*
Increment.

Remarks

The value of the current character extra attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders set the value of the current character extra attribute to the value specified in the order.

Set Character Precision / Push and Set Character Precision

These orders set, or push and set, the value of the current character precision attribute.

Set Character Precision (GSCR)
X'39'(prec)

Push and Set Character Precision (GPSCR)
X'79'(prec)

Parameters

prec *(GBIT8)*
Value for character-precision attribute:

All other values are reserved.

X'00' Drawing default
X'01' String precision
X'02' Character precision
X'03' Stroke precision

Remarks

The value of the current character precision attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character precision attribute to the value in the order.

Set Character Set / Push and Set Character Set

These orders set, or push and set, the value of the current character-set attribute.

Set Character Set (GSCS)
X'38'(lcid)

Push and Set Character Set (GPSCS)
X'78'(lcid)

Parameters

lcid *(GUCHAR)*
Local identifier (LCID) for the character set:

X'00'	Drawing default
X'01'–X'FE'	Lcid for the symbol set
X'FF'	Special character set.

Remarks

The value of the current character-set attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character-set attribute to the value in the order.

Set Character Shear / Push and Set Character Shear

These orders set, or push and set, the value of the current character shear attribute.

Set Character Shear (GSCH)
X'35'(len, hx, hy)

Push and Set Character Shear (GPSCH)
X'75'(len, hx, hy)

Parameters

len *(GLENGTH1)*
Length of following data.

hx *(GROSOL)*
Dividend of shear ratio.

hy *(GROSOL)*
Divisor of shear ratio.

Remarks

When *hx* and *hy* are both 0, the drawing default is set. The value of the current character shear attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character shear attribute to the value in the order.

Set Clip Path

This order sets the current clip path.

Set Clip Path (GSCPTH)
X'B4'(len, flags, res, pthid)

Parameters

len *(GLENGTH1)*
Length of following data.

flags
Extra functions:

res *(GBIT1)*
Reserved.

0 Only valid value.

fill *(GBIT1)*
Values:

0 Alternate mode
1 Winding mode.

inter *(GBIT1)*
Values:

0 Set to specified path
1 Set to intersection of specified and current clip path.

res2 *(GBIT5)*
Reserved.

B'00000' Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pthid *(GLONG)*
Path identifier.

X'00000000'	No clipping.
X'00000001' – X'FFFFFFFF'	Path identifier.

Set Color / Push and Set Color

These orders set, or push and set, the value of the current color attribute.

Set Color (GSCOL)
X'0A'(col)

Push and Set Color (GPSCOL)
X'4A'(col)

Parameters

col *(GBIT8)*

Value for color attribute:

X'00' – X'08'	These one-byte values are converted to two-byte values by preceding the value with X'FF'. The resultant is then treated as a two-byte value as defined by the Set Extended Color / Push and Set Extended Color order.
Other	Reserved values.

Remarks

The value of the current color attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current color attribute to the value in the order.

Set Current Position / Push and Set Current Position

These orders set, or push and set, the value of the current position.

Set Current Position (GSCP)
X'21'(len, p)

Push and Set Current Position (GPSCP)
X'61'(len, p)

Parameters

len *(GLENGTH1)*

Length of following data.

p *(GPOINT)*

Coordinate data.

Remarks

The value of the current position is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current position to the value in the order.

Set Extended Color / Push and Set Extended Color

These orders set, or push and set, the value of the current color attribute.

Set Extended Color (GSECOL)
X'26'(len, color)

Push and Set Extended Color (GPSECOL)
X'66'(len, color)

Parameters

len *(GLENGTH1)*
Length of following data.

X'02' Only valid value.

color *(GBIT16)*
Color-table index.

Except for the special values, the values X'0000' through X'nnnn' are allowed color indexes; that is, as many values as are allowed by the size of the LCT.

Special Values

X'0000' Drawing default
X'0007' White
X'0008' Black
X'FF00' Drawing default
X'FF0x' Color indexes X'000x', where x is in the range 1 through 7.
X'FF08' Color index 0 (reset color).

Remarks

The value of the current extended color attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current extended color attribute to the value in the order.

Set Fractional Line Width / Push and Set Fractional Line Width

These orders set, or push and set, the value of the current line-width attribute.

Set Fractional Line Width (GSFLW)
X'11'(len, line width)

Push and Set Fractional Line Width (GPSFLW)
X'51'(len, line width)

Parameters

len *(GLENGTH1)*
Length of following data.

X'02' Only valid value.

line width *(GROUFS)*
Value for the line-width attribute.

The nonzero value is an integral and fractional multiplier of the normal line width:

X'0000' Drawing default
X'0001' – X'FFFF' Multiplier of normal line width.

Remarks

The value of the current line-width attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current line-width attribute to the value in the order.

Set Indexed Color / Push and Set Indexed Color

These orders set, or push and set, the value of the current color attribute.

Set Indexed Color (GSICOL)
X'A6'(len, flags, index)

Push and Set Indexed Color (GPSICOL)
X'E6'(len, flags, index)

Parameters

len *(GLENGTH1)*
Length of following data.

X'04' Only valid value.

flags
Values:

default *(GBIT1)*
Options:

0 Use specified index
1 Use drawing default color.

spec *(GBIT1)*
Options:

0 Use index directly
1 Special value.

res *(GBIT6)*
Reserved.

000000 Only valid value.

index *(GINDEX3)*
Value for color index.

The value is a direct index into the current color table or a special value.

The table can be the standard table, or one loaded by the user.

The special values are:

1 Black
2 White
4 All ones
5 All zeros.

Remarks

The value of the current color attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current color attribute to the value in the order.

Set Individual Attribute / Push and Set Individual Attribute

These orders set, or push and set, the value of the color, background color, mix, or background mix attribute for the line character, marker, pattern, or image primitive type.

Set Individual Attribute (GSIA)
X'14'(len, atype, ptype, flag1, val)

Push and Set Individual Attribute (GPSIA)
X'54'(len, atype, ptype, flag1, val)

Parameters

len *(GLENGTH1)*
Length of following data.

atype *(GBIT8)*
Attribute type:

X'1'	Color
X'2'	Background color
X'3'	Mix
X'4'	Background Mix
Other	All other values are reserved.

ptype *(GBIT8)*
Primitive type:

X'1'	Line
X'2'	Character
X'3'	Marker
X'4'	Pattern
X'5'	Image
Other	All other values are reserved.

flag1
Values:

default *(GBIT1)*
Options:

0	Use specified value
1	Use drawing default color.

spec *(GBIT1)*
Options:

0	Use value directly
1	Special Value.

res *(GBIT6)*
Reserved.

000000	Only valid value.

val *(GINDATT)*
Color index value.

For colors, the value is a direct index into the current color table or a special value.

The table can be the standard table, or one loaded by the user.

The special values are:

1 Black
2 White
4 All ones
5 All zeros.

Remarks

The value of the current attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the individual attribute to the value in the order.

Set Line End / Push and Set Line End

These orders set, or push and set, the value of the current line-end attribute.

Set Line End (GSLE)
X'1A'(lineend)

Push and Set Line End (GPSLE)
X'5A'(lineend)

Parameters

lineend *(GBIT8)*
Value for the line-end attribute:

X'00' Drawing default
X'01' Flat
X'02' Square
X'03' Round
Other Reserved values.

Remarks

The value of the current line-end attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current line-end attribute to the value in the order.

Set Line Join / Push and Set Line Join

These orders set the value of the current line-join attribute.

Set Line Join (GSLJ)
X'1B'(linejoin)

Push and Set Line Join (GPSLJ)
X'5B'(linejoin)

Parameters

linejoin *(GBIT8)*

Value for line-join attribute:

X'00'	Drawing default
X'01'	Bevel
X'02'	Round
X'03'	Miter
Other	Reserved values.

Remarks

The value of the current line-join attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current line-join attribute to the value in the order.

Set Line Type / Push and Set Line Type

These orders set, or push and set, the value of the current line-type attribute.

Set Line Type (GSLT)
X'18'(linetype)

Push and Set Line Type (GPSLT)
X'58'(linetype)

Parameters

linetype *(GBIT8)*

Value for line-type attribute.

The value is an index into a notational line-type table:

X'00'	Drawing default
X'01'	Dotted line
X'02'	Short dashed line
X'03'	Dash-dot line
X'04'	Double dotted line
X'05'	Long dashed line
X'06'	Dash-double-dot line
X'07'	Solid line
X'08'	Invisible line
Other	Reserved values.

Remarks

The value of the current line-type attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current line-type attribute to the value in the order.

Set Line Width / Push and Set Line Width

These orders set, or push and set, the value of the current line-width attribute to the value specified in the order.

Set Line Width (GSLW)
X'19'(linewidth)

Push and Set Line Width (GPSLW)
X'59'(linewidth)

Parameters

linewidth *(GBIT8)*
Value for line-width attribute:

X'00'	Drawing default
X'01' – X'FF'	Integral multiplier of normal line width.

Remarks

The value of the current line-width attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current line-width attribute to the value in the order.

Set Marker Cell / Push and Set Marker Cell

These orders set, or push and set, the value of the current marker cell-size attribute.

Set Marker Cell (GSMC)
X'37'(len, cellx, celly, flags, res)

Push and Set Marker Cell (GPSMC)
X'77'(len, cellx, celly, flags, res)

Parameters

len *(GLENGTH1)*
Length of following data.

cellx *(GROSOL)*
X part of marker cell-size attribute.

celly *(GROSOL)*
Y part of marker cell-size attribute.

flags
This is an optional extension.

Values:

notdefilt *(GBIT1)*
Options:

0 A cell size of zero sets drawing default
1 A cell size of zero sets to zero.

res *(GBIT7)*
Reserved.

0000000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

Remarks

The value of the current marker cell-size attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current marker cell-size attribute to the value in the order.

Set Marker Precision / Push and Set Marker Precision

These orders set, or push and set, the value of the current marker-precision attribute.

Set Marker Precision (GSMP)
X'3B'(prec)

Push and Set Marker Precision (GPSMP)
X'7B'(prec)

Parameters

prec *(GBIT8)*
Value for marker-precision attribute:

X'00'	Drawing default
X'01'	String precision
X'02'	Character precision
X'03'	Stroke precision
Other	Reserved values.

Remarks

The value of the current marker-precision attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current-marker precision attribute to the value in the order.

Set Marker Set / Push and Set Marker Set

These orders set, or push and set, the value of the current marker symbol-set attribute.

Set Marker Set (GSMS)
X'3C'(lcid)

Push and Set Marker Set (GPSMS)
X'7C'(lcid)

Parameters

lcid *(GUCHAR)*
Local identifier (LCID) for the marker set:

X'00'	Drawing default
X'01' – X'FE'	LCID for the coded font
X'FF'	Special marker set.

Remarks

The value of the current marker symbol-set attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current marker symbol-set attribute to the value in the order.

Set Marker Symbol / Push and Set Marker Symbol

These orders set, or push and set, the value of the current marker symbol attribute.

Set Marker Symbol (GSMT)
X'29'(n)

Push and Set Marker Symbol (GPSMT)
X'69'(n)

Parameters

n *(GBIT8)*
Value of marker symbol code point.

Special marker set

When this is selected (**lcid** = X'FF'), the values are:

X'00'	Drawing default
X'01'	Cross
X'02'	Plus
X'03'	Diamond
X'04'	Square
X'05'	6-point star
X'06'	8-point star
X'07'	Filled diamond
X'08'	Filled square
X'09'	Dot
X'0A'	Small circle
X'40'	Blank
Other	Reserved values.

Marker set

Values are as follows for any other set:

X'00'	Drawing default
X'01'–X'FF'	These are the code points into the current marker set.

Remarks

The value of the current marker symbol attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current marker symbol attribute to the value in the order.

Set Mix / Push and Set Mix

These orders set, or push and set, the value of the current mix attribute.

Set Mix (GSMX)
X'0C'(mode)

Push and Set Mix (GPSMX)
X'4C'(mode)

Parameters

mode *(GBIT8)*

Mix-mode value:

X'00'	Drawing default
X'01'	OR
X'02'	Overpaint
X'03'	Reserved
X'04'	Exclusive-OR
X'05'	Leave alone
X'06'	AND
X'07'	Subtract
X'08'	Source AND (inverse destination)
X'09'	All zeros
X'0A'	Inverse (source OR destination)
X'0B'	Inverse (source XOR destination)
X'0C'	Inverse destination
X'0D'	Source OR (inverse destination)
X'0E'	Inverse source
X'0F'	(Inverse source) OR destination
X'10'	Inverse (source AND destination)
X'11'	All ones.
Other	Reserved values.

Remarks

The value of the current mix attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current mix attribute to the value in the order.

Set Model Transform / Push and Set Model Transform

These orders set, or push and set, values in the current model transform.

Set Model Transform (GSTM)
X'24'(len, res, flags, mask, mx)

Push and Set Model Transform (GPSTM)
X'64'(len, res, flags, mask, mx)

Parameters

len *(GLENGTH1)*

Length of following data.

res *(GBIT8)*

Reserved.

X'00' Only valid value.

flags

Values:

res *(GBIT6)*

Reserved.

B'000000' Only valid value.

cm *(GBIT2)*

Matrix control bits:

B'00'	Unity matrix
B'01'	Concatenate after

B'10' Concatenate before
B'11' Overwrite.

mask *(GBIT16)*
Load mask.

mx *(GROSOL*number of bits set on in mask)*
Matrix values.

Remarks

The value of the current model transform is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set values in the current model transform as specified in the order.

Set Pattern Reference Point / Push and Set Pattern Reference Point

These orders set, or push and set, the value of the current pattern reference-point attribute.

Set Pattern Reference Point (GSPRP)
X'A0'(len, flags, res, pref)

Push and Set Pattern Reference Point (GPSPRP)
X'E0'(len, flags, res, pref)

Parameters

len *(GLENGTH1)*
Length of following data.

flags
Values:

default *(GBIT1)*
Options:

0 Set to specified value
1 Set to the drawing default.

res *(GBIT7)*
Reserved

0000000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pref *(GPOINT)*
Coordinate data of the pattern-reference point.

Remarks

The value of the current pattern reference-point attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current reference-point attribute to the value in the order.

Set Pattern Set / Push and Set Pattern Set

These orders set, or push and set, the value of the current pattern symbol-set attribute.

Set Pattern Set (GSPS)
X'08'(lcid)

Push and Set Pattern Set (GPSPS)
X'48'(lcid)

Parameters

lcid *(GUCHAR)*
Local identifier (LCID) for the pattern set:

X'00'	Drawing default
X'01'–X'FE'	LCID for the symbol set
X'FF'	Special pattern set.

Remarks

The value of the current pattern symbol-set attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current pattern symbol-set attribute to the value in the order.

Set Pattern Symbol / Push and Set Pattern Symbol

These orders set, or push and set, the value of the current pattern-symbol attribute.

Set Pattern Symbol (GSPT)
X'28'(patt)

Push and Set Pattern Symbol (GPSPT)
X'09'(patt)

Parameters

patt *(GBIT8)*
Value for pattern-symbol attribute.

Special pattern set

When this is selected (**lcid** = X'FF'), the values are:

X'00'	Drawing default
X'01'–X'08'	Density one through density eight (decreasing)
X'09'	Vertical lines
X'0A'	Horizontal lines
X'0B'	Diagonal lines 1 (bottom-left to top-right)
X'0C'	Diagonal lines 2 (bottom-left to top-right)
X'0D'	Diagonal lines 1 (top-left to bottom-right)
X'0E'	Diagonal lines 2 (top-left to bottom-right)
X'0F'	No shading
X'10'	Solid shading
X'40'	Blank.

Other Reserved values.

Pattern set

Values are as follows for any other set:
X'00' Drawing default
X'01'–X'FF' These are the code points into the current pattern set.

Remarks

The value of the current pattern-symbol attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current pattern-symbol attribute to the value in the order.

Set Pick Identifier / Push and Set Pick Identifier

These orders set, or push and set, the value of the current pick identifier.

Set Pick Identifier (GSPIK)
X'43'(len, pkid)

Push and Set Pick Identifier (GPSPIK)
X'23'(len, pkid)

Parameters

len *(GLENGTH1)*
Length of following data.

pkid *(GLONG)*
Pick identifier.

Remarks

The value of the current pick identifier is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the value of the current pick identifier to the value in the order.

Set Segment Boundary

This order defines the maximum extent of the boundaries of the associated root segment. It is valid *only* in a root segment prolog.

Set Segment Boundary (GSSB)
X'32'(len, res, mask, bb)

Parameters

len *(GLENGTH1)*
Length of following data.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

mask
Values:

res1 *(GBIT2)*
Reserved.

00 Only valid value.

xl *(GBIT1)*
X left limit.

0 Not included in list of *bb* values
1 Is included in list of *bb* values.

xr *(GBIT1)*
X right limit.

0 Not included in list of *bb* values
1 Is included in list of *bb* values.

yb *(GBIT1)*
Y bottom limit.

0 Not included in list of *bb* values
1 Is included in list of *bb* values.

yt *(GBIT1)*
Y top limit.

0 Not included in list of *bb* values
1 Is included in list of *bb* values.

res2 *(GBIT2)*
Reserved.

00 Only valid value.

bb *(GROSOL*number of bits set on in mask)*
Boundary values.

Remarks
The order is only valid in a root-segment prolog.

Set Stroke Line Width / Push and Set Stroke Line Width

These orders set the current stroke line-width attribute.

Set Stroke Line Width (GSSLW)
X'15'(len, flags, res, strwidth)

Push and Set Stroke Line Width (GPSSLW)
X'55'(len, flags, res, strwidth)

Parameters
len *(GLENGTH1)*
Length of following data.

flags

defit *(GBIT1)*
Values:

0 Set to value
1 Set to drawing default.

res *(GBIT7)*
Reserved.

B'0000000' Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

strwidth *(GROSOL)*
Value for stroke width.

Set Text Alignment / Push and Set Text Alignment

These orders set, or push and set, the value of the current text alignment attribute.

Set Text Alignment (GSTA)
X'36'(horiz, vert)

Push and Set Text Alignment (GPSTA)
X'76'(horiz, vert)

Parameters

horiz *(GUCHAR)*
Horizontal alignment as follows:

X'01' Normal alignment. The alignment assumed depends on the current character direction:

Left to right	Left alignment.
Top to bottom	Center alignment.
Right to left	Right alignment.
Bottom to top	Center alignment.

X'02' Left alignment. The string is aligned on the left edge of its leftmost character.
X'03' Center alignment. The string is aligned on the arithmetic mean of left and right.
X'04' Right alignment. The string is aligned on the right edge of its rightmost character.
X'05' Standard alignment. The alignment assumed depends on the current character direction:

Left to right	Left alignment.
Top to bottom	Left alignment.
Right to left	Right alignment.
Bottom to top	Left alignment.

vert *(GUCHAR)*
Vertical alignment as follows:

X'01' Normal alignment. The alignment assumed depends on the current character direction:

Left to right	Base alignment.
Top to bottom	Top alignment.
Right to left	Base alignment.
Bottom to top	Bottom alignment.

X'02' Top Alignment. The string is aligned on the top edge of its topmost character.
X'03' Halfr alignment. The string is aligned on the arithmetic mean of top and bottom.
X'04' Base alignment. The string is aligned on the base of its bottom character.
X'05' Bottom Alignment. The string is aligned on the bottom edge of its bottom character.
X'06' Standard alignment. The alignment assumed depends on the current character direction:

Left to right	Bottom alignment.
Top to bottom	Top alignment.
Right to left	Bottom alignment.
Bottom to top	Bottom alignment.

Remarks

The value of the current text alignment attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders set the value of the current text alignment attribute to the value specified in the order.

Set Viewing Transform

This order sets the current viewing transform.

Set Viewing Transform (GSTV)
X'31'(len, res, flags, mask, mx)

Parameters

len *(GLENGTH1)*
Length of following data.

res *(GBIT8)*
Reserved.

X'0' Only valid value.

flags
Values:

res1 *(GBIT5)*
Reserved.

00000 Only valid value.

control *(GBIT1)*
Values:

0 Concatenate before drawing default
1 Concatenate before the current viewing transform.

res2 *(GBIT2)*
Reserved.

00 Only valid value.

mask *(GBIT16)*
Load mask.

mx *(GROSOL*number of bits set on in mask)*
Matrix values.

Set Viewing Window / Push and Set Viewing Window

These orders set, or push and set, the current viewing window.

Set Viewing Window (GSVW)
X'27'(len, flag, mask, ww)

Push and Set Viewing Window (GPSVW)
X'67'(len, flag, mask, ww)

Parameters

len *(GLENGTH1)*
Length of following data.

flag
Values:

replace *(GBIT1)*
Values:

0 Intersect with current window
1 Replace current with new window.

res *(GBIT7)*
Reserved.

0000000 Only valid value.

mask
Values:

res1 *(GBIT2)*
Reserved.

00 Only valid value.

xl *(GBIT1)*
X left limit.

0 Not included in list of *ww* values
1 Is included in list of *ww* values

xr *(GBIT1)*
X right limit.

0 Not included in list of *ww* values
1 Is included in list of *ww* values

yb *(GBIT1)*
Y bottom limit.

0 Not included in list of *ww* values
1 Is included in list of *ww* values

yt *(GBIT1)*
Y top limit.

0 Not included in list of *ww* values
1 Is included in list of *ww* values

res2 *(GBIT2)*
Reserved value.

00 Only valid value.

ww *(GROSOL*number of bits set on in mask)*
Window values.

Remarks

The value of the current viewing window is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the current viewing window using the values in the order.

Sharp Fillet at Given Position / Sharp Fillet at Current Position

This order generates a curve that starts at a given position, and uses points P_1 and P_2, together with the sharpness specification S_1.

Sharp Fillet at Given Position (GSFLT)
X'E4'(len, p_0, p_1, p_2, p_3, p_4, pn-1, pn, s_1, s_2, sn/2)

Sharp Fillet at Current Position (GCSFLT)
X'A4'(len, p_1, p_2, p_3, p_4, pn-1, pn, s_1, s_2, sn/2)

Parameters

len *(GLENGTH1)*
Length of following data.

p_0 *(GPOINT)*
Coordinate data of first curve start.

This parameter is only present in a Sharp Fillet at Given Position order.

p_1 *(GPOINT)*
Coordinate data of first curve control point.

p_2 *(GPOINT)*
Coordinate data of first curve end.

p_3 *(GPOINT)*
Coordinate data of second curve control point.

p_4 *(GPOINT)*
Coordinate data of second curve end.

pn-1 *(GPOINT)*
Coordinate data of last curve control point.

pn *(GPOINT)*
Coordinate data of last curve end.

s_1 *(GROF)*
Sharpness specification of first curve.

s_2 *(GROF)*
Sharpness specification of second curve.

sn/2 *(GROF)*
Sharpness specification of last curve.

Remarks

Further points are used in groups of two to form a polycurve.

Chapter 34. Code Pages

The initialization file contains country information relating to date, time, and numeric formats. It does not contain code-page information; this is obtained from the CONFIG.SYS file.

Applications start with the default code page. The default code page is set when the operating system is installed. It can be changed subsequently either by reinstalling the operating system or by editing the COUNTRY statement in the CONFIG.SYS file.

A GPI presentation space inherits the code page of the process that created it. The code page changes only when the process issues a GpiSetCp function.

Windowed PM Applications

Windowed PM applications allow the code-page calls to use any of the supported ASCII code pages. These are:

	Char. Set	Code Page
Canadian-French	993	863
Desktop Publishing	1146	1004
Iceland	991	861
Latin 1 Multilingual	980	850
Latin 2 Multilingual	982	852
Nordic	995	865
Portuguese	990	860
Turkey	987	857
U.S. (IBM PC)	919	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

	Char. Set	Code Page
Austrian/German	697	273
Belgian	697	500
Czechoslovakia	959	870
Danish/Norwegian	697	277
Finnish/Swedish	697	278
French	697	297
Hungary	959	870
Iceland	697	871
International	697	500
Italian	697	280
Poland	959	870
Portuguese	697	037
Spanish	697	284
Turkey	1152	1026
U.K.-English	697	285
U.S.-English	697	037
Yugoslavia	959	870

Note: Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.

** Trademark of Microsoft Corporation

The operating system provides the following additional code-page setting and query calls for the supported ASCII and EBCDIC code pages. These calls work independently of the CONFIG.SYS file.

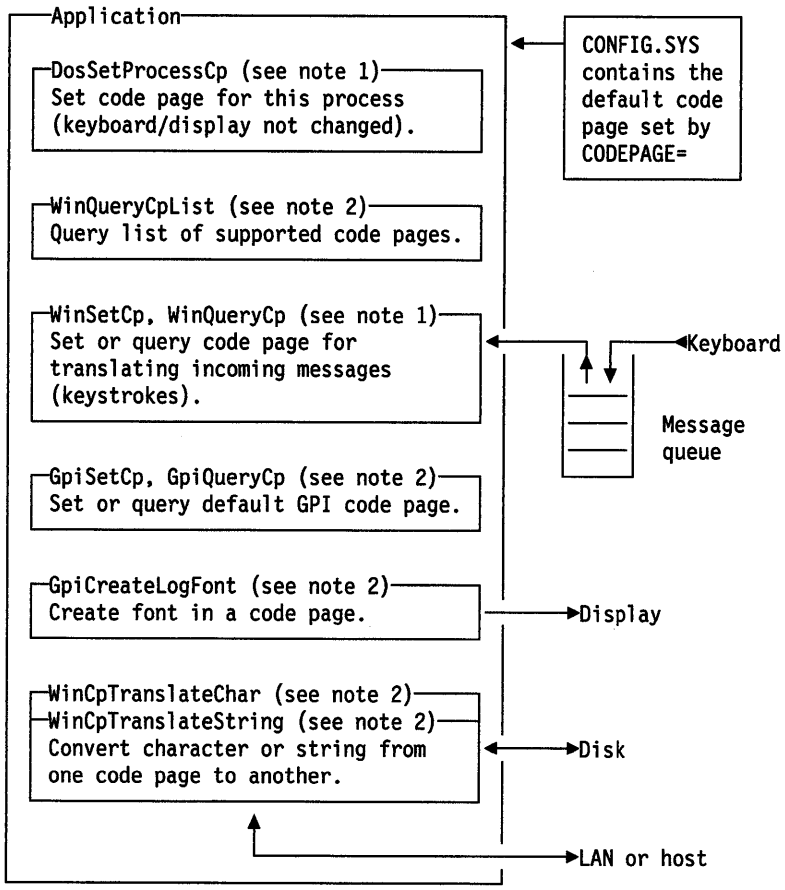
GpiSetCp	Sets the code page for GPI.
GpiQueryCp	Queries the code page for GPI.
GpiCreateLogFont	Creates fonts in a code page.
WinSetCp	Sets the code page for a message queue.
WinQueryCp	Queries the code page for a message queue.

WinQueryCpList creates a list of code pages supported by the operating system.

Text entered in a dialog box is supplied to the application in the code page of the queue ('queue code page'). If possible, the code page of a resource (for example, a menu or dialog box) should match the code page of the queue. In general, code page 850 is the best choice for both an application and its resources.

Applications should be able to process data from a variety of sources. Because code page 850 contains most of the characters in other supported code pages, this is usually the best choice for the queue code page.

OS/2 Code Page Options for PM Applications



Note 1: Either of the two ASCII code pages specified in CONFIG.SYS.
Code page 1004 is also supported.
Note 2: Any supported ASCII or EBCDIC code page as reported by
WinQueryCpList.
Code page 1004 is also supported.

Figure 34-1. OS/2 Code Page Options for PM Applications

OS/2 Font Support for Multiple Code Pages

The operating system supports multiple code pages for text input and output. A single font resource is used to support all the code pages. This section describes the font resource format.

Font Code-Page Functions

Many of the characters required by each code page are common; for example, the first 128 characters of all the ASCII code pages are identical. This set of characters is called the Universal Glyph List (UGL). A code page is simply a set of pointers into the UGL.

As the characters in every font are in the same order, only one set of code-page translation tables is necessary.

Note: The fonts of Microsoft Windows support only code page 1004.

Font Layout

The following table lists the full character set in the order in which the characters occur in the multi-code-page font. Characters are listed in order of their universal glyph list (UGL) number; the graphic character global identifier (GCGID) and a description of each character are also given.

UGL	GCGID	Description
1	SS000000	Smiling face
2	SS010000	Smiling face, reverse image
3	SS020000	Heart suit symbol
4	SS030000	Diamond suit symbol
5	SS040000	Club suit symbol
6	SS050000	Spade suit symbol
7	SM570000	Bullet
8	SM570001	Bullet, reverse image
9	SM750000	Open circle
10	SM750002	Open circle, reverse image
11	SM280000	Male symbol
12	SM290000	Female symbol
13	SM930000	Musical note
14	SM910000	Two musical notes
15	SM690000	Sun symbol
16	SM590000	Forward arrow indicator
17	SM630000	Back arrow indicator
18	SM760000	Up-down arrow
19	SP330000	Double exclamation point
20	SM250000	Paragraph symbol (USA)
21	SM240000	Section symbol (USA), paragraph (Europe)
22	SM700000	Solid horizontal rectangle
23	SM770000	Up-down arrow, perpendicular
24	SM320000	Up arrow
25	SM330000	Down arrow
26	SM310000	Right arrow
27	SM300000	Left arrow
28	SA420000	Right angle symbol
29	SM780000	Left-right arrow
30	SM600000	Solid triangle
31	SV040000	Solid triangle, inverted
32	SP010000	Space
33	SP020000	Exclamation point
34	SP040000	Quotation marks
35	SM010000	Number sign
36	SC030000	Dollar sign
37	SM020000	Percent sign
38	SM030000	Ampersand
39	SP050000	Apostrophe
40	SP060000	Left parenthesis
41	SP070000	Right parenthesis

UGL	GCGID	Description
42	SM040000	Asterisk
43	SA010000	Plus sign
44	SP080000	Comma
45	SP100000	Hyphen/minus sign
46	SP110000	Period/full stop
47	SP120000	Slash
48	ND100000	Zero
49	ND010000	One
50	ND020000	Two
51	ND030000	Three
52	ND040000	Four
53	ND050000	Five
54	ND060000	Six
55	ND070000	Seven
56	ND080000	Eight
57	ND090000	Nine
58	SP130000	Colon
59	SP140000	Semicolon
60	SA030000	Less than sign/greater than (arabic)
61	SA040000	Equal Sign
62	SA050000	Greater than sign/less than (arabic)
63	SP150000	Question mark
64	SM050000	At sign
65	LA020000	A capital
66	LB020000	B capital
67	LC020000	C capital
68	LD020000	D capital
69	LE020000	E capital
70	LF020000	F capital
71	LG020000	G capital
72	LH020000	H capital
73	LI020000	I capital
74	LJ020000	J capital
75	LK020000	K capital
76	LL020000	L capital
77	LM020000	M capital
78	LN020000	N capital
79	LO020000	O capital
80	LP020000	P capital
81	LQ020000	Q capital
82	LR020000	R capital
83	LS020000	S capital
84	LT020000	T capital
85	LU020000	U capital
86	LV020000	V capital
87	LW020000	W capital
88	LX020000	X capital
89	LY020000	Y capital
90	LZ020000	Z capital
91	SM060000	Left bracket
92	SM070000	Backslash
93	SM080000	Right bracket
94	SD150000	Circumflex Accent
95	SP090000	Underline, continuous underscore
96	SD130000	Grave accent
97	LA010000	a small
98	LB010000	b small
99	LC010000	c small
100	LD010000	d small
101	LE010000	e small
102	LF010000	f small
103	LG010000	g small

UGL	GCGID	Description
104	LH010000	h small
105	LI010000	i small
106	LJ010000	j small
107	LK010000	k small
108	LL010000	l small
109	LM010000	m small
110	LN010000	n small
111	LO010000	o small
112	LP010000	p small
113	LQ010000	q small
114	LR010000	r small
115	LS010000	s small
116	LT010000	t small
117	LU010000	u small
118	LV010000	v small
119	LW010000	w small
120	LX010000	x small
121	LY010000	y small
122	LZ010000	z small
123	SM110000	Left brace
124	SM130000	Vertical line, logical OR
125	SM140000	Right brace
126	SD190000	Tilde
127	SM790000	House
128	LC420000	C cedilla capital
129	LU170000	U diaeresis small
130	LE110000	E acute small
131	LA150000	A circumflex small
132	LA170000	A diaeresis small
133	LA130000	A grave small
134	LA270000	A overcircle small
135	LC410000	C cedilla small
136	LE150000	E circumflex small
137	LE170000	E diaeresis small
138	LE130000	E grave small
139	LI170000	I diaeresis small
140	LI150000	I circumflex small
141	LI130000	I grave small
142	LA180000	A diaeresis capital
143	LA280000	A overcircle capital
144	LE120000	E acute capital
145	LA510000	AE diphthong small
146	LA520000	AE diphthong capital
147	LO150000	O circumflex small
148	LO170000	O diaeresis small
149	LO130000	O grave small
150	LU150000	U circumflex small
151	LU130000	U grave small
152	LY170000	Y diaeresis small
153	LO180000	O diaeresis capital
154	LU180000	U diaeresis capital
155	LO610000	O slash small
156	SC020000	Pound sterling sign
157	LO620000	O slash capital
158	SA070000	Multiply sign
159	SC070000	Florin sign
160	LA110000	A acute small
161	LI110000	I acute small
162	LO110000	O acute small
163	LU110000	U acute small
164	LN190000	N tilde small
165	LN200000	N tilde capital

UGL	GCGID	Description
166	SM210000	Ordinal indicator, feminine
167	SM200000	Ordinal indicator, masculine
168	SP160000	Question mark, inverted
169	SM530000	Registered trademark symbol
170	SM660000	Logical NOT, end of line symbol
171	NF010000	One-half
172	NF040000	One-quarter
173	SP030000	Exclamation point, inverted
174	SP170000	Left angled quotes
175	SP180000	Right angled quotes
176	SF140000	Fill character, light
177	SF150000	Fill character, medium
178	SF160000	Fill character, heavy
179	SF110000	Center box bar vertical
180	SF090000	Right middle box side
181	LA120000	A acute capital
182	LA160000	A circumflex capital
183	LA140000	A grave capital
184	SM520000	Copyright symbol
185	SF230000	Right box side double
186	SF240000	Center box bar vertical double
187	SF250000	Upper right box corner double
188	SF260000	Lower right box corner double
189	SC040000	Cent sign
190	SC050000	Yen sign
191	SF030000	Upper right box corner
192	SF020000	Lower left box corner
193	SF070000	Middle box bottom
194	SF060000	Middle box top
195	SF080000	Left middle box side
196	SF100000	Center box bar horizontal
197	SF050000	Box intersection
198	LA190000	A tilde small
199	LA200000	A tilde capital
200	SF380000	Lower left box corner double
201	SF390000	Upper left box corner double
202	SF400000	Middle box bottom double
203	SF410000	Middle box top double
204	SF420000	Left box side double
205	SF430000	Center box bar horizontal double
206	SF440000	Box intersection double
207	SC010000	International currency symbol
208	LD630000	eth Icelandic small
209	LD620000	D stroke capital, Eth Icelandic capital
210	LE160000	E circumflex capital
211	LE180000	E diaeresis capital
212	LE140000	E grave capital
213	LI610000	I dotless small
214	LI120000	I acute capital
215	LI160000	I circumflex capital
216	LI180000	I diaeresis capital
217	SF040000	Lower right box corner
218	SF010000	Upper left box corner
219	SF610000	Solid fill character
220	SF570000	Solid fill character, bottom half
221	SM650000	Vertical line, broken
222	LI140000	I grave capital
223	SF600000	Solid fill character, top half
224	LO120000	O acute capital
225	LS610000	Sharp s small
226	LO160000	O circumflex capital
227	LO140000	O grave capital

UGL	GCGID	Description
228	LO190000	O tilde small
229	LO200000	O tilde capital
230	SM170000	Micro symbol
231	LT630000	Thorn Icelandic small
232	LT640000	Thorn Icelandic capital
233	LU120000	U acute capital
234	LU160000	U circumflex capital
235	LU140000	U grave capital
236	LY110000	y acute small
237	LY120000	Y acute capital
238	SM150000	Overline
239	SD110000	Acute accent
240	SP320000	Syllable hyphen
241	SA020000	Plus or minus sign
242	SM100000	Double underscore
243	NF050000	Three-quarters
244	SM250000	Paragraph symbol (USA)
245	SM240000	Section symbol (USA), paragraph (Europe)
246	SA060000	Divide sign
247	SD410000	Cedilla (or sedila) accent
248	SM190000	Degree symbol
249	SD170000	Diaeresis, umlaut accent
250	SD630000	Middle dot
251	ND011000	One superscript
252	ND031000	Three superscript
253	ND021000	Two superscript
254	SM470000	Solid square, histogram, square bullet
255	SP300000	Required space
256	SC060000	Peseta sign
257	SM680000	Start of line symbol
258	SF190000	Right box side double to single
259	SF200000	Right box side single to double
260	SF210000	Upper right box corner single to double
261	SF220000	Upper right box corner double to single
262	SF270000	Lower right box corner single to double
263	SF280000	Lower right box corner double to single
264	SF360000	Left box side single to double
265	SF370000	Left box side double to single
266	SF450000	Middle box bottom single to double
267	SF460000	Middle box bottom double to single
268	SF470000	Middle box top double to single
269	SF480000	Middle box top single to double
270	SF490000	Lower left box corner double to single
271	SF500000	Lower left box corner single to double
272	SF510000	Upper left box corner single to double
273	SF520000	Upper left box corner double to single
274	SF530000	Box intersection single to double
275	SF540000	Box intersection double to single
276	SF580000	Solid fill character, left half
277	SF590000	Solid fill character, right half
278	GA010000	Alpha small
279	GG020000	Gamma capital
280	GP010000	Pi small
281	GS020000	Sigma capital
282	GS010000	Sigma small
283	GT010000	Tau small
284	GF020000	Phi capital
285	GT620000	Theta capital
286	GO320000	Omega capital
287	GD010000	Delta small
288	SA450000	Infinity symbol
289	GF010000	Phi small

UGL	GCGID	Description
290	GE010000	Epsilon small
291	SA380000	Intersection, logical product
292	SA480000	Indentity symbol, almost equal
293	SA530000	Greater than or equal sign
294	SA520000	Less than or equal sign
295	SS260000	Upper integral symbol section
296	SS270000	Lower integral symbol section
297	SA700000	Nearly equals symbol
298	SA790000	Product dot
299	SA800000	Radical symbol
300	LN011000	N small superscript
301	SD310000	Macron accent
302	SD230000	Breve accent
303	SD290000	Overdot accent (over small Alpha)
304	SD270000	Overcircle accent
305	SD250000	Double acute accent
306	SD430000	Ogonek accent
307	SD210000	Caron accent
308	SP190000	Left single quote
309	SP200000	Right single quote
310	SP210000	Left double quotes
311	SP220000	Right double quotes
312	SS680000	Endash
313	SM900000	Emdash
314	SD150000	Circumflex accent
315	SD190000	Tilde accent
316	SP260000	Single quote on baseline (German lower)
317	SP230000	Left lower double quotes
318	SV520000	Ellipsis
319	SM340000	Dagger footnote indicator
320	SM350000	Double dagger footnote indicator
321	SD150100	Circumflex accent (over small alpha)
322	SM560000	Permille symbol
323	LS220000	S caron capital
324	SP270000	French single open quote
325	LO520000	OE ligature capital
326	SD190100	Tilde accent (over small alpha)
327	SM540000	Trademark symbol
328	LS210000	s caron small
329	SP280000	French single close quote
330	LO510000	oe ligature small
331	LY180000	Y diaeresis capital
333	LG230000	g Breve Small
334	LG240000	G Breve Capital
335	LI130000	i Grave Small
336	LI300000	I Overdot Capital
337	LS410000	s Cedilla Small
338	LS420000	S Cedilla Capital
339	LA230000	a Breve Small
340	LA240000	A Breve Capital
341	LA430000	a Ogonek Small
342	LA440000	A Ogonek Capital
343	LC110000	c Acute Small
344	LC120000	C Acute Capital
345	LC210000	c Caron Small
346	LC220000	C Caron Capital
347	LD210000	d Caron Small
348	LD220000	D Caron Capital
349	LD610000	d Stroke Small
350	LE210000	e Caron Small
351	LE220000	E Caron Capital
352	LE430000	e Ogenek Small

UGL	GCGID	Description
353	LE440000	E Ogonek Capital
354	LL110000	l Acute Small
355	LL120000	L Acute Capital
356	LL210000	l Caron Small
357	LL220000	L Caron Capital
358	LL610000	l Stroke Small
359	LL620000	L Stroke Capital
360	LN110000	n Acute Small
361	LN120000	N Acute Capital
362	LN210000	n Caron Small
363	LN220000	N Caron Capital
364	LO250000	o Double Acute Small
365	LO260000	O Double Acute Capital
366	LR110000	r Acute Small
367	LR120000	R Acute Capital
368	LR210000	r Caron Small
369	LR220000	R Caron Capital
370	LS110000	s Acute Small
371	LS120000	S Acute Capital
	LS210000	+s Caron Small
	LS220000	+S Caron Capital
	LS410000	*s Cedilla Small
	LS420000	*S Cedilla Capital
372	LT210000	t Caron Small
373	LT220000	T Caron Capital
374	LT410000	t Cedilla Small
375	LT420000	T Cedilla Capital
376	LU250000	u Double Acute Small
377	LU260000	U Double Acute Capital
378	LU270000	u Overcircle Small
379	LU280000	u Overcircle Capital
380	LZ110000	z Acute Small
381	LZ120000	Z Acute Capital
382	LZ210000	z Caron Small
383	LZ220000	Z Caron Capital
384	LZ290000	z Overdot Small
385	LZ300000	Z Overdot Capital

ASCII Code Pages

1		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	2 A↓B	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
0	-0		►		0	@	P	`	p	Ç	É	á	▒	└	╨	α	≡
1	-1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
2	-2	☻	↕	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
3	-3	♥	!!	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	-4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5	-5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	-6	♠	▬	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
7	-7	•	↨	'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	-8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
9	-9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	•
10	-A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
11	-B	♂	←	+	;	K	[	k	{	ï	¢	½	╗	╦	■	δ	√
12	-C	♀	∟	,	<	L	\	l	|	î	£	¼	╝	╠	▬	∞	ⁿ
13	-D	♪	↔	-	=	M	]	m	}	ì	¥	¡	╜	═	▌	ϕ	²
14	-E	♫	▲	.	>	N	^	n	~	Ä	Pt	«	╛	╬	▐	ε	■
15	-F	☼	▼	/	?	O	_	o	⌂	Å	ƒ	»	┐	╧	▀	∩	

Figure 34-2. US-English: ASCII Code Page 437

1		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	2 A↓B	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
0	-0		►		0	@	P	`	p	Ç	É	á	▒	└	ð	Ó	
1	-1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	Ð	β	±
2	-2	☻	↕	"	2	B	R	b	r	é	Æ	ó	▓	┬	Ê	Ô	=
3	-3	♥	!!	#	3	C	S	c	s	â	ô	ú	│	├	Ë	Ò	¾
4	-4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	È	õ	¶
5	-5	♣	§	%	5	E	U	e	u	à	ò	Ñ	Á	┼	ı	Õ	§
6	-6	♠	▬	&	6	F	V	f	v	å	û	ª	Â	ã	Í	µ	÷
7	-7	•	↨	'	7	G	W	g	w	ç	ù	º	À	Ã	Î	þ	¸
8	-8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	©	╚	Ï	Þ	°
9	-9	○	↓	)	9	I	Y	i	y	ë	Ö	®	╣	╔	┘	Ú	¨
10	-A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Û	·
11	-B	♂	←	+	;	K	[	k	{	ï	ø	½	╗	╦	■	Ù	¹
12	-C	♀	∟	,	<	L	\	l	|	î	£	¼	╝	╠	▬	ý	³
13	-D	♪	↔	-	=	M	]	m	}	ì	Ø	¡	¢	═	¦	Ý	²
14	-E	♫	▲	.	>	N	^	n	~	Ä	×	«	¥	╬	Ì	¯	■
15	-F	☼	▼	/	?	O	_	o	⌂	Å	ƒ	»	┐	¤	▀	´	

Figure 34-3. Latin 1 Multilingual: ASCII Code Page 850

1		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	2 A/B	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
0	-0		►		0	@	P	`	p	Ç	É	á	▒	└	đ	Ó	–
1	-1	☺	◄	!	1	A	Q	a	q	ü	Ĺ	í	▒	┴	Đ	β	˝
2	-2	☻	↕	″	2	B	R	b	r	é	ĺ	ó	▓	┬	Ď	Ô	˛
3	-3	♥	!!	#	3	C	S	c	s	â	ô	ú	│	├	Ë	Ń	ˇ
4	-4	♦	¶	$	4	D	T	d	t	ä	ö	Ą	┤	─	ď	ń	˘
5	-5	♣	§	%	5	E	U	e	u	ů	Ľ	ą	Á	┼	Ň	ň	§
6	-6	♠	▬	&	6	F	V	f	v	ć	ľ	Ž	Â	Ă	Í	Š	÷
7	-7	•	↨	′	7	G	W	g	w	ç	Ś	ž	Ě	ă	Î	š	¸
8	-8	◘	↑	(	8	H	X	h	x	ł	ś	Ę	Ş	╚	ě	Ř	°
9	-9	○	↓	)	9	I	Y	i	y	ë	Ö	ę	╣	╔	┘	Ú	¨
10	-A	◙	→	*	:	J	Z	j	z	Ő	Ü	¬	║	╩	┌	ŕ	•
11	-B	♂	←	+	;	K	[	k	{	ő	Ť	ź	╗	╦	■	Ű	ű
12	-C	♀	∟	,	<	L	\	l	|	î	ť	Č	╝	╠	▬	ý	Ř
13	-D	♪	↔	-	=	M	]	m	}	Ź	Ł	ş	Ż	═	Ţ	Ý	ř
14	-E	♫	▲	.	>	N	^	n	~	Ä	×	«	ż	╬	Ů	ţ	■
15	-F	☼	▼	/	?	O	_	o	⌂	Ć	č	»	┐	¤	▬	´	

Figure 34-4. Latin 2 Multilingual: ASCII Code Page 852

1		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	2 A/B	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
0	-0		►		0	@	P	`	p	Ç	É	á	▒	└	º	Ó	–
1	-1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	ª	β	±
2	-2	☻	↕	″	2	B	R	b	r	é	Æ	ó	▓	┬	Ê	Ô	
3	-3	♥	!!	#	3	C	S	c	s	â	ô	ú	│	├	Ë	Ò	¾
4	-4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	È	õ	¶
5	-5	♣	§	%	5	E	U	e	u	à	ò	Ñ	Á	┼		Õ	§
6	-6	♠	▬	&	6	F	V	f	v	å	û	Ğ	Â	ã	Í	µ	÷
7	-7	•	↨	′	7	G	W	g	w	ç	ù	ğ	À	Ã	Î		¸
8	-8	◘	↑	(	8	H	X	h	x	ê	İ	¿	©	╚	Ï	×	°
9	-9	○	↓	)	9	I	Y	i	y	ë	Ö	®	╣	╔	┘	Ú	¨
10	-A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Û	•
11	-B	♂	←	+	;	K	[	k	{	ï	ø	½	╗	╦	■	Ù	¹
12	-C	♀	∟	,	<	L	\	l	|	î	£	¼	╝	╠	▬	ì	³
13	-D	♪	↔	-	=	M	]	m	}	ı	Ø	¡	¢	═	¦	ÿ	²
14	-E	♫	▲	.	>	N	^	n	~	Ä	Ş	«	¥	╬	Ì	¯	■
15	-F	☼	▼	/	?	O	_	o	⌂	Å	ş	»	┐	¤	▬	´	

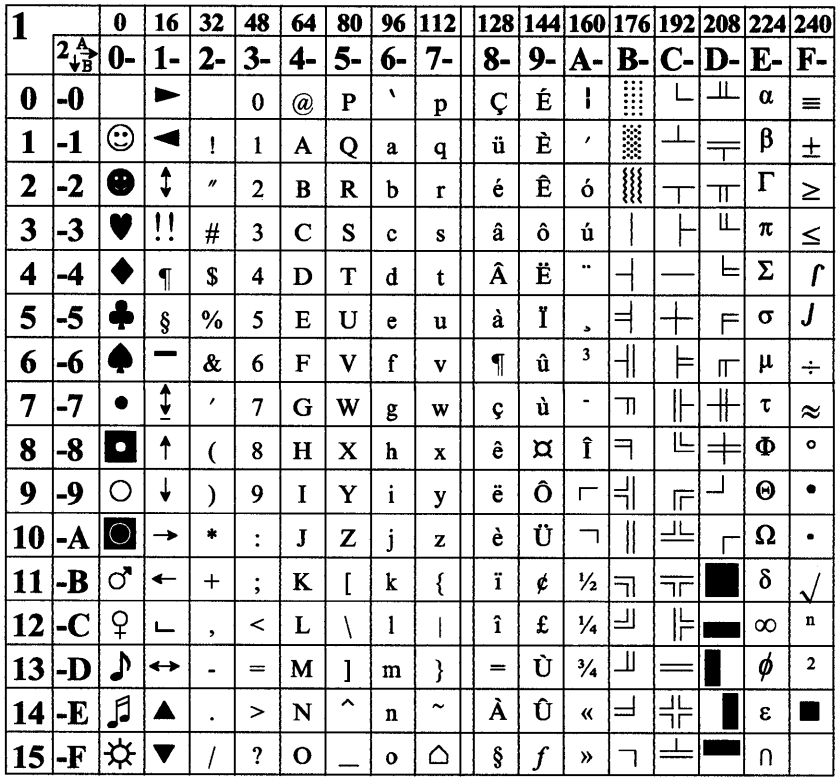
Figure 34-5. Turkey: ASCII Code Page 857

1		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	2 A / B	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
0	-0		►		0	@	P	`	p	Ç	É	á	⣿	└	╨	α	≡
1	-1	☺	◄	!	1	A	Q	a	q	ü	À	í	▒	┴	╤	β	±
2	-2	☻	↕	″	2	B	R	b	r	é	È	ó	⣿	┬	╥	Γ	≥
3	-3	♥	!!	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	-4	♦	¶	$	4	D	T	d	t	ã	õ	ñ	┤	─	╘	Σ	⌠
5	-5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	-6	♠	‾	&	6	F	V	f	v	Á	Ú	ª	╢	╞	╓	μ	÷
7	-7	•	↨	′	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	-8	◘	↑	(	8	H	X	h	x	ê	Ì	¿	╕	╚	╪	Φ	°
9	-9	○	↓	)	9	I	Y	i	y	Ê	Õ	Ò	╣	╔	┘	Θ	•
10	-A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
11	-B	♂	←	+	;	K	[	k	{	Í	¢	½	╗	╦	■	δ	√
12	-C	♀	∟	,	<	L	\	l	|	Ô	£	¼	╝	╠	▬	∞	n
13	-D	♪	↔	-	=	M	]	m	}	ì	Ù	¡	╜	═	▌	ϕ	2
14	-E	♫	▲	.	>	N	^	n	~	Ã	Pts	«	╛	╬	▐	ε	■
15	-F	☼	▼	/	?	O	_	o	⌂	Â	Ó	»	┐	╧	▀	∩	

Figure 34-6. Portuguese: ASCII Code Page 860

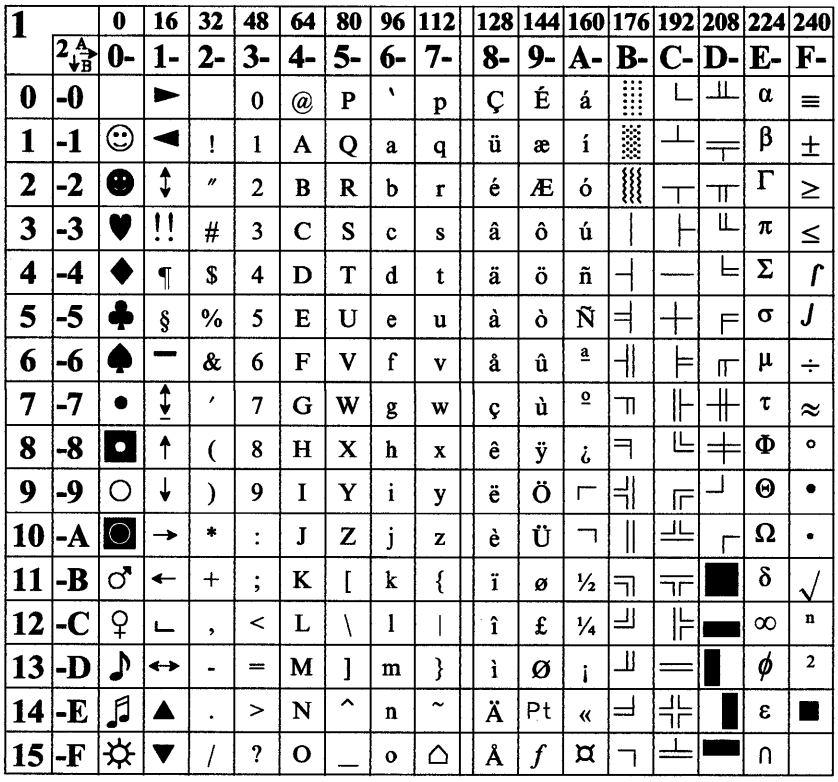
1		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	2 A / B	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
0	-0		►		0	@	P	`	p	Ç	É	á	⣿	└	╨	α	≡
1	-1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
2	-2	☻	↕	″	2	B	R	b	r	é	Æ	ó	⣿	┬	╥	Γ	≥
3	-3	♥	!!	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	-4	♦	¶	$	4	D	T	d	t	ä	ö	Á	┤	─	╘	Σ	⌠
5	-5	♣	§	%	5	E	U	e	u	à	þ	Í	╡	┼	╒	σ	⌡
6	-6	♠	‾	&	6	F	V	f	v	å	û	Ó	╢	╞	╓	μ	÷
7	-7	•	↨	′	7	G	W	g	w	ç	Ý	Ú	╖	╟	╫	τ	≈
8	-8	◘	↑	(	8	H	X	h	x	ê	ý	¿	╕	╚	╪	Φ	°
9	-9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	•
10	-A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
11	-B	♂	←	+	;	K	[	k	{	Đ	ø	½	╗	╦	■	δ	√
12	-C	♀	∟	,	<	L	\	l	|	ð	£	¼	╝	╠	▬	∞	n
13	-D	♪	↔	-	=	M	]	m	}	Þ	Ø	¡	╜	═	▌	ϕ	2
14	-E	♫	▲	.	>	N	^	n	~	Ä	Pts	«	╛	╬	▐	ε	■
15	-F	☼	▼	/	?	O	_	o	⌂	Å	ƒ	»	┐	╧	▀	∩	

Figure 34-7. Iceland: ASCII Code Page 861



1		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	2 A B	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
0	-0		►		0	@	P	`	p	Ç	É	¦	⣿	└	╨	α	≡
1	-1	☺	◄	!	1	A	Q	a	q	ü	È	´	▒	┴	╤	β	±
2	-2	☻	↕	″	2	B	R	b	r	é	Ê	ó	▓	┬	╥	Γ	≥
3	-3	♥	!!	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	-4	♦	¶	$	4	D	T	d	t	Â	Ë	¨	┤	─	╘	Σ	⌠
5	-5	♣	§	%	5	E	U	e	u	à	Ì	¸	╡	┼	╒	σ	⌡
6	-6	♠	‾	&	6	F	V	f	v	¶	û	³	╢	╞	╓	µ	÷
7	-7	•	↨	′	7	G	W	g	w	ç	ù	¯	╖	╟	╫	τ	≈
8	-8	◘	↑	(	8	H	X	h	x	ê	¤	Î	╕	╚	╪	Φ	°
9	-9	○	↓	)	9	I	Y	i	y	ë	Ô	⌐	╣	╔	┘	Θ	•
10	-A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
11	-B	♂	←	+	;	K	[	k	{	ï	¢	½	╗	╦	▉	δ	√
12	-C	♀	∟	,	<	L	\	l	|	î	£	¼	╝	╠	▬	∞	n
13	-D	♪	↔	-	=	M	]	m	}	=	Ù	¾	╜	═	▊	ϕ	2
14	-E	♫	▲	.	>	N	^	n	~	À	Û	«	╛	╬	▋	ε	■
15	-F	☼	▼	/	?	O	_	o	⌂	§	ƒ	»	┐	╧	▄	∩	

Figure 34-8. Canadian-French: ASCII Code Page 863



1		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	2 A B	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
0	-0		►		0	@	P	`	p	Ç	É	á	⣿	└	╨	α	≡
1	-1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
2	-2	☻	↕	″	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
3	-3	♥	!!	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	-4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5	-5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	-6	♠	‾	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
7	-7	•	↨	′	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	-8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
9	-9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	•
10	-A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
11	-B	♂	←	+	;	K	[	k	{	ï	ø	½	╗	╦	▉	δ	√
12	-C	♀	∟	,	<	L	\	l	|	î	£	¼	╝	╠	▬	∞	n
13	-D	♪	↔	-	=	M	]	m	}	ì	Ø	¡	╜	═	▊	ϕ	2
14	-E	♫	▲	.	>	N	^	n	~	Ä	Pt	«	╛	╬	▋	ε	■
15	-F	☼	▼	/	?	O	_	o	⌂	Å	ƒ	¤	┐	╧	▄	∩	

Figure 34-9. Norwegian: ASCII Code Page 865

↓ →	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0			(SP)	0	@	P	`	p			(RSP)	°	À	Đ	à	ð
-1			!	1	A	Q	a	q		‘	¡	þ	Á	Ñ	á	ñ
-2			"	2	B	R	b	r	‚	’	¢	²	Â	Ò	â	ò
-3			#	3	C	S	c	s		“	£	³	Ã	Ó	ã	ó
-4	ˉ		$	4	D	T	d	t	„	”	¤	´	Ä	Ô	ä	ô
-5	˘		%	5	E	U	e	u	…	•	¥	µ	Å	Õ	å	õ
-6	˙		&	6	F	V	f	v	†	–	¦	¶	Æ	Ö	æ	ö
-7			'	7	G	W	g	w	‡	—	§	·	Ç	×	ç	÷
-8	°		(	8	H	X	h	x	ˆ	˜	¨	¸	È	Ø	è	ø
-9			)	9	I	Y	i	y	‰	™	©	¹	É	Ù	é	ù
-A	″		*	:	J	Z	j	z	Š	š	ª	º	Ê	Ú	ê	ú
-B	˛		+	;	K	[	k	{	<	>	«	»	Ë	Û	ë	û
-C	ˇ		,	<	L	\	l	|	Œ	œ	¬	¼	Ì	Ü	ì	ü
-D			-	=	M	]	m	}			(SHY)	½	Í	Ý	í	ý
-E			.	>	N	^	n	~			®	¾	Î	Þ	î	þ
-F			/	?	O	_	o			Ÿ	¯	¿	Ï	ß	ï	ÿ

Figure 34-10. Desktop Publishing: ASCII Code Page 1004

EBCDIC Code Pages

↓ →	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	{	}	\	0
-1						é	/	É	a	j	¯	[	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	ˋ	i	r	z	¾	I	R	Z	9
-A					$	!	’	:	«	ª	¡	^	–	1	2	3
-B					.	£	,	#	»	º	¿	]	ô	û	Ô	Û
-C					<	*	%	@	ð	æ	Ð	~	ö	ü	Ö	Ü
-D					(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F					|	¬	?	"	±	¤	®	ˇ	õ	ÿ	Õ	EO

Figure 34-11. US-English: EBCDIC Code Page 037

↓→	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	ä	ü	Ö	0
-1						é	/	É	a	j	ß	£	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					{	ë	[	Ë	c	l	t	•	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	@	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	~	Ñ	'	i	r	z	¾	I	R	Z	9
-A					Ä	Ü	ö	:	«	ª	¡	¬	–	1	2	3
-B					.	$	,	#	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	§	ð	æ	Ð	—	'	}	\	]
-D					(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F					!	^	?	"	±	¤	®	ˇ	õ	ÿ	Õ	EO

Figure 34-12. Austrian/German: EBCDIC Code Page 273

↓→	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	é	è	ç	0
-1						{	/	É	a	j	¨	£	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					@	}	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					\	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	'	i	r	z	¾	I	R	Z	9
-A					[	[	ù	:	«	ª	¡	¬	–	1	2	3
-B					.	$	,	#	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	à	ð	æ	Ð	—	ö	ü	Ö	Ü
-D					(	)	_	'	ý	¸	Ý	~	ò	'	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F					!	^	?	"	±	¤	®	ˇ	õ	ÿ	Õ	EO

Figure 34-13. Belgian: EBCDIC Code Page 274 (supported for migration purposes)

↓	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	'	@	°	µ	¢	æ	å	\	0
-1						é	/	É	a	j	ü	£	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					}	ï	$	Ï	g	p	x	¼	G	P	X	7
-8					ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A					#	¤	ø	:	«	ª	¡	¬	–	1	2	3
-B					.	Å	,	Æ	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	Ø	ð	{	Ð	—	ö	~	Ö	Ü
-D					(	)	_	'	ý	˛	Ý	¨	ò	ù	Ò	Ù
-E					+	;	>	=	þ	[	Þ	´	ó	ú	Ó	Ú
-F					!	^	?	"	±	]	®	ˇ	õ	ÿ	Õ	EO

Figure 34-14. Danish/Norwegian: EBCDIC Code Page 277

↓	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	ä	å	É	0
-1						'	/	\	a	j	ü	£	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					{	ë	#	Ë	c	l	t	•	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	[	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					}	ï	$	Ï	g	p	x	¼	G	P	X	7
-8					ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	é	i	r	z	¾	I	R	Z	9
-A					§	¤	ö	:	«	ª	¡	¬	–	1	2	3
-B					.	Å	,	Ä	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	Ö	ð	æ	Ð	—	'	~	@	Ü
-D					(	)	_	'	ý	˛	Ý	¨	ò	ù	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	\	ó	ú	Ó	Ú
-F					!	^	?	"	±	]	®	ˇ	õ	ÿ	Õ	EO

Figure 34-15. Finnish/Swedish: EBCDIC Code Page 278

↓→	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	[	µ	¢	à	é	ç	0
-1						]	/	É	a	j	ì	#	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					{	}	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	@	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					\	~	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	ù	i	r	z	¾	I	R	Z	9
-A					°	é	ò	:	«	ª	¡	¬	_	1	2	3
-B					.	$	,	£	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	§	ð	æ	Ð	—	ö	ü	Ö	Ü
-D					(	)	_	'	ý	¸	Ý	¨	’	‘	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F					!	^	?	"	±	¤	®	ˇ	õ	ÿ	Õ	EO

Figure 34-16. Italian: EBCDIC Code Page 280

↓→	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	ã	′	Ç	0
-1						é	/	É	a	j	ç	£	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					{	î	#	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					~	ì	\	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	‘	i	r	z	¾	I	R	Z	9
-A					[	]	õ	:	«	ª	¡	¬	_	1	2	3
-B					.	$	,	Ã	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	Õ	ð	æ	Ð	—	ö	ü	Ö	Ü
-D					(	)	_	'	ý	¸	Ý	¨	ò	’	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	}	ó	ú	Ó	Ú
-F					!	^	?	"	±	¤	®	ˇ	’	ÿ	@	EO

Figure 34-17. Portuguese: EBCDIC Code Page 282 (supported for migration purposes)

↓ →	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	{	}	\	0
-1						é	/	É	a	j	¨	£	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					'	ß	#	`	i	r	z	¾	I	R	Z	9
-A					[	]	ñ	:	«	ª	¡	^	–	1	2	3
-B					.	$	,	Ñ	»	º	¿	!	ô	û	Ô	Û
-C					<	*	%	@	ð	æ	Ð	—	ö	ü	Ö	Ü
-D					(	)	_	'	ý	¸	Ý	~	ò	ù	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F					|	¬	?	"	±	¤	®	ˇ	õ	ÿ	Õ	EO

Figure 34-18. Spanish: EBCDIC Code Page 284

↓ →	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	{	}	\	0
-1						é	/	É	a	j	¯	[	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A					$	!	'	:	«	ª	¡	^	–	1	2	3
-B					.	£	,	#	»	º	¿	]	ô	û	Ô	Û
-C					<	*	%	@	ð	æ	Ð	~	ö	ü	Ö	Ü
-D					(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F					|	¬	?	"	±	¤	®	ˇ	õ	ÿ	Õ	EO

Figure 34-19. UK-English: EBCDIC Code Page 285

↓ →	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	[	'	¢	é	è	ç	0
-1						{	/	É	a	j	¨	#	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					@	}	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	]	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					\	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	µ	i	r	z	¾	I	R	Z	9
-A					°	§	ù	:	«	ª	¡	¬	–	1	2	3
-B					.	$	,	£	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	à	ð	æ	Ð	—	ö	ü	Ö	Ü
-D					(	)	_	'	ý	‗	Ý	~	ò	’	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F					!	^	?	"	±	¤	®	ˇ	õ	ÿ	Õ	EO

Figure 34-20. French: EBCDIC Code Page 297

↓ →	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	{	}	\	0
-1						é	/	É	a	j	~	£	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	'	i	r	z	¾	I	R	Z	9
-A					[	]	’	:	«	ª	¡	¬	–	1	2	3
-B					.	$	,	#	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	@	ð	æ	Ð	—	ö	ü	Ö	Ü
-D					(	)	_	'	ý	‗	Ý	¨	ò	ù	Ò	Ù
-E					+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F					!	^	?	"	±	¤	®	ˇ	õ	ÿ	Õ	EO

Figure 34-21. International: EBCDIC Code Page 500

↓→	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	˘	˘	°	a̧	˙	{	}	\	0
-1						é	/	É	a	j	~	Ą	A	J	÷	1
-2					â	ę	Â	Ę	b	k	s	ż	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	Ţ	C	L	T	3
-4					ţ	ů	″	Ů	d	m	u	Ż	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					ă	î	Ă	Î	f	o	w	ž	F	O	W	6
-7					č	ˇ	Č	Ľ	g	p	x	ź	G	P	X	7
-8					ç	Í	Ç	Ĺ	h	q	y	Ž	H	Q	Y	8
-9					ć	ß	Ć	`	i	r	z	Ź	I	R	Z	9
-A					[	]	|	:	ś	ł	Ś	Ł	–	Ě	ď	Ď
-B					.	$	,	#	ň	ń	Ň	Ń	ô	ű	Ô	Ű
-C					<	*	%	@	đ	š	Đ	Š	ö	ü	Ö	Ü
-D					(	)	_	'	ý	˛	Ý	¨	ŕ	ť	Ŕ	Ť
-E					+	;	>	=	ř	¸	Ř	´	ó	ú	Ó	Ú
-F					!	^	?	"	ş	¤	Ş	×	ő	ě	Ő	

Figure 34-22. Czechoslovakia/Hungary/Poland/Yugoslovia: EBCDIC Code Page 870

↓→	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	þ	æ	´	0
-1						é	/	É	a	j	ö	£	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	·	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	ð	i	r	z	¾	I	R	Z	9
-A					Þ	Æ	¦	:	«	ª	¡	¬	–	1	2	3
-B					.	$	,	#	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	Ð	`	}	@	—	~	ü	^	]
-D					(	)	_	'	ý	˛	Ý	¨	ò	ù	Ò	Ù
-E					+	;	>	=	{	]	[	\	ó	ú	Ó	Ú
-F					!	Ö	?	"	±	¤	®	x	õ	ÿ	Õ	

Figure 34-23. Iceland: EBCDIC Code Page 871

↓→	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0						&	-	ø	Ø	°	µ	¢	ç	ğ	ü	0
-1						é	/	É	a	j	ö	£	A	J	÷	1
-2					â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3					ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4					à	è	À	È	d	m	u	©	D	M	U	4
-5					á	í	Á	Í	e	n	v	§	E	N	V	5
-6					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7					å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8					{	ì	[	Ì	h	q	y	½	H	Q	Y	8
-9					ñ	ß	Ñ	ı	i	r	z	¾	I	R	Z	9
-A					Ç	Ğ	ş	:	«	ª	¡	¬	–	¹	²	³
-B					.	İ	,	Ö	»	º	¿	|	ô	û	Ô	Û
-C					<	*	%	Ş	}	æ	]	—	~	\	#	"
-D					(	)	_	'	`	ª	$	¨	ò	ù	Ò	Ù
-E					+	;	>	=	¦	Æ	@	´	ó	ú	Ó	Ú
-F					!	^	?	Ü	±	¤	®	x	õ	ÿ	Õ	

Figure 34-24. Turkey: EBCDIC Code Page 1026

DBCS Support

The Presentation Interface supports double-byte character sets (DBCS) by means of three kinds of character-encoding schemes:

SBCS only Single-byte code pages; for example, U.S.-English.

Both ASCII and EBCDIC SBCS code pages have similar representations.

DBCS only Double-byte code pages; for example, Kanji.

Both ASCII and EBCDIC DBCS code pages have similar representations.

MIXED Code pages that incorporate a combination of single-byte and double-byte characters.

The internal representations of EBCDIC MIXED and ASCII MIXED code pages differ:

- ASCII MIXED: the encoding scheme allows single-byte characters to be distinguished from double-byte characters algorithmically. With this scheme the number of characters entered or displayed is the same as the number of characters in a field.
- EBCDIC MIXED: the encoding scheme requires that control characters within the string switch from single to double byte encoding (and from double to single byte encoding). These control characters are the shift-out (SO) and shift-in (SI) characters.

  With this encoding scheme there may be many more characters in the input or data field than characters displayed or printed.

All MIXED strings are displayed without a space between sequences of single-byte and double-byte characters (unless spaces are explicitly included in these positions within the string).

For graphics, selection of a local identifier (lcid) identifies the code page in force, and therefore whether subsequent character strings are to be interpreted as SBCS, DBCS, ASCII MIXED, or EBCDIC MIXED.

Appendix A. Data Types

This chapter describes data types in C language.

ACCEL

Accelerator structure.

```
typedef struct _ACCEL {
USHORT    fs;
USHORT    key;
USHORT    cmd;
 } ACCEL;
```

fs (USHORT)
Options.

key (USHORT)
Key.

cmd (USHORT)
Command code.

The value to be placed in the *uscmd* parameter of a WM_HELP, a WM_COMMAND, or a WM_SYSCOMMAND.

ACCELTABLE

Accelerator-table structure.

```
typedef struct _ACCELTABLE {
USHORT    cAccel;
USHORT    codepage;
ACCEL     aaccel[1];
 } ACCELTABLE;
```

cAccel (USHORT)
Number of accelerator entries.

codepage (USHORT)
Code page for accelerator entries.

aaccel[1] (ACCEL)
Accelerator entries.

The default accelerator table has the following 16 entries:

Options		Key	Command
HELP	VIRTUALKEY	VK_F1	0
SYSCOMMAND ALT	VIRTUALKEY	VK_F4	SC_CLOSE
SYSCOMMAND ALT	VIRTUALKEY	VK_ENTER	SC_RESTORE
SYSCOMMAND ALT	VIRTUALKEY	VK_NEWLINE	SC_RESTORE
SYSCOMMAND ALT	VIRTUALKEY	VK_F5	SC_RESTORE
SYSCOMMAND ALT	VIRTUALKEY	VK_F6	SC_NEXTFRAME
SYSCOMMAND ALT	VIRTUALKEY	VK_F7	SC_MOVE
SYSCOMMAND ALT	VIRTUALKEY	VK_F8	SC_SIZE
SYSCOMMAND ALT	VIRTUALKEY	VK_F9	SC_MINIMIZE
SYSCOMMAND ALT	VIRTUALKEY	VK_F10	SC_MAXIMIZE
SYSCOMMAND	VIRTUALKEY	VK_F10	SC_APPMENU
SYSCOMMAND LONEKEY	VIRTUALKEY	VK_ALT	SC_APPMENU
SYSCOMMAND LONEKEY	VIRTUALKEY	VK_ALTGRAF	SC_APPMENU
SYSCOMMAND ALT	VIRTUALKEY	VK_SPACE	SC_SYSMENU
SYSCOMMAND SHIFT	VIRTUALKEY	VK_ESC	SC_SYSMENU
SYSCOMMAND CONTROL	VIRTUALKEY	VK_ESC	SC_TASKMANAGER

ARCPARAMS

Arc-parameters structure.

```
typedef struct _ARCPARAMS {
LONG    lP;
LONG    lQ;
LONG    lR;
LONG    lS;
 } ARCPARAMS;
```

lP (LONG)
P coefficient.

lQ (LONG)
Q coefficient.

lR (LONG)
R coefficient.

lS (LONG)
S coefficient.

AREABUNDLE

Area-attributes bundle structure.

```
typedef struct _AREABUNDLE {
LONG      lColor;
LONG      lBackColor;
USHORT    usMixMode;
USHORT    usBackMixMode;
USHORT    usSet;
USHORT    usSymbol;
POINTL    ptlRefPoint;
 } AREABUNDLE;
```

lColor (LONG)
Area foreground color.

lBackColor (LONG)
Area background color.

usMixMode (USHORT)
Area foreground-mix mode.

usBackMixMode (USHORT)
Area background-mix mode.

usSet (USHORT)
Pattern set.

usSymbol (USHORT)
Pattern symbol.

ptlRefPoint (POINTL)
Pattern reference point.

ATOM

Atom identity.

```
typedef USHORT ATOM;
```

BANDRECT

Rectangle structure, used for the coordinates of an output band (see DevEscape).

An empty rectangle is one for which *lxLeft* is greater than *lxRight*, or *lyBottom* is greater than *lyTop*.

```
typedef struct _BANDRECT {
LONG    lxLeft;
LONG    lyBottom;
LONG    lxRight;
LONG    lyTop;
 } BANDRECT;
```

lxLeft (LONG)
x-coordinate of left edge of rectangle.

lyBottom (LONG)
y-coordinate of bottom edge of rectangle.

lxRight (LONG)
x-coordinate of right edge of rectangle.

lyTop (LONG)
y-coordinate of top edge of rectangle.

BITMAPINFO

Bit-map information structure.

Each bit plane logically contains (*cx* cy* cBitCount*) bits, although the actual length can be greater because of padding.

See also BITMAPINFO2, which is preferred.

```
typedef struct _BITMAPINFO {
ULONG     cbFix;
USHORT    cx;
USHORT    cy;
USHORT    cPlanes;
USHORT    cBitCount;
RGB       argbColor[1];
 } BITMAPINFO;
```

cbFix (ULONG)
Length of fixed portion of structure.

cx (USHORT)
Bit-map width in pels.

cy (USHORT)
Bit-map height in pels.

cPlanes (USHORT)
Number of bit planes.

cBitCount (USHORT)
Number of bits per pel within a plane.

argbColor[1] (RGB)
Array of RGB values.

This is a packed array of 24-bit RGB values. If there are N bits per pel (N = *cPlanes* cBitCount*), the array contains 2**N RGB values. However, if N = 24 the bit map does not need the *color* array because the standard-format bit map, with 24 bits per pel, is assumed to contain RGB values.

BITMAPINFO2

Bit-map information structure.

Each bit plane logically contains (*cx * cy * cBitCount*) bits, although the actual length can be greater because of padding.

Note: Many functions can accept either this structure or the BITMAPINFO structure. Where possible, BITMAPINFO2 should be used.

The *cbFix* field is used to find the color table, if any, that goes with the information in this structure. A color table is an array of color (RGB2) values. If there are N bits per pel (N = *cPlanes* cBitCount*), the array contains 2**N color values. However, if N = 24, the color table is not required (because the standard-format bit map, with 24 bits per pel, is assumed to contain RGB values), unless either *cclrUsed* or *cclrImportant* is non-zero.

```
typedef struct _BITMAPINFO2 {
ULONG     cbFix;
ULONG     cx;
ULONG     cy;
USHORT    cPlanes;
USHORT    cBitCount;
ULONG     ulCompression;
ULONG     cbImage;
ULONG     cxResolution;
ULONG     cyResolution;
ULONG     cclrUsed;
ULONG     cclrImportant;
USHORT    usUnits;
USHORT    usReserved;
USHORT    usRecording;
USHORT    usRendering;
ULONG     cSize1;
ULONG     cSize2;
ULONG     ulColorEncoding;
ULONG     ulIdentifier;
RGB2      argb2Color[1];
 } BITMAPINFO2;
```

cbFix (ULONG)
Length of fixed portion of structure.

The structure can be truncated after *cBitCount* or any subsequent field.

The length does not include the length of the color table. Where the color table is present, it is at an offset of *cbFix* from the start of the BITMAPINFO2 structure.

cx (ULONG)
Bit-map width in pels.

cy (ULONG)
Bit-map height in pels.

cPlanes (USHORT)
Number of bit planes.

cBitCount (USHORT)
Number of bits per pel within a plane.

ulCompression (ULONG)
Compression scheme used to store the bit map:

BCA_UNCOMP	Bit map is uncompressed.
BCA_HUFFMAN1D	The bit map is compressed by a modified Huffman encoding. This is valid for a bi-level (one bit per pel) bit map.
BCA_RLE4	The bit map is a 4-bit per pel run-length encoded bit map. See BITMAPINFOHEADER2 for a description of the format of the compressed data.
BCA_RLE8	The bit map is a 8-bit per pel run-length encoded bit map. See BITMAPINFOHEADER2 for a description of the format of the compressed data.
BCA_RLE24	The bit map is a 24-bit per pel run-length encoded bit map. See BITMAPINFOHEADER2 for a description of the format of the compressed data.

cbImage (ULONG)
Length of bit-map storage data, in bytes.

If the bit map is uncompressed, zero (default) can be specified for this.

cxResolution (ULONG)
Horizontal component of the resolution of target device.

The resolution of the device the bit map is intended for, in the units specified by *usUnits*. This information enables an application to select

from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution (ULONG)
Vertical component of the resolution of target device.

See the description of *cxResolution*.

cclrUsed (ULONG)
Number of color indexes used.

The number of color indexes from the color table that are used by the bit map. If it is zero (the default), all the indexes are used. If it is non-zero, only the first *cclrUsed* entries in the table are accessed by the system, and further entries can be omitted.

For the standard formats with a *cBitCount* of 1, 4, or 8 (and *cPlanes* equal to 1), any indexes beyond *cclrUsed* are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, *cclrUsed* is the number of colors used by the bit map.

cclrImportant (ULONG)
Minimum number of color indexes for satisfactory appearance of the bit map.

More colors may be used in the bit map, but it is not necessary to assign them to the device palette. These additional colors may be mapped to the nearest colors available.

Zero (the default) means that all entries are important.

For a 24-bitcount standard format bit map, the *cclrImportant* colors are also listed in the color table following the BITMAPINFO2 structure.

usUnits (USHORT)
Units of measure.

Units of measure of the horizontal and vertical components of resolution, *cxResolution* and *cyResolution*.

BRU_METRIC Pels per meter. This is the default value.

usReserved (USHORT)
Reserved.

This is a reserved field. If present, it must be zero.

usRecording (USHORT)
Recording algorithm.

The format in which the bit map data is recorded.

BRA_BOTTOMUP Scan lines are recorded bottom-to-top. This is the default value.

usRendering (USHORT)
Halftoning algorithm.

The algorithm used to record bit map data that has been digitally halftoned.

BRH_NOTHALFTONED Bit-map data is not halftoned. This is the default value.

BRH_ERRORDIFFUSION Error Diffusion or Damped Error Diffusion algorithm.

BRH_PANDA Processing Algorithm for Non-coded Document Acquisition.

BRH_SUPERCIRCLE Super Circle algorithm.

cSize1 (ULONG)
Size value 1.

If BRH_ERRORDIFFUSION is specified in *usRendering*, *cSize1* is the error

damping as a percentage in the range 0 through 100. A value of 100% indicates no damping, and a value of 0% indicates that any errors are not diffused.

If BRH_PANDA or BRH_SUPERCIRCLE is specified, *cSize1* is the x dimension of the pattern used, in pels.

cSize2 (ULONG)
Size value 2.

If BRH_ERRORDIFFUSION is specified in *usRendering*, this parameter is ignored.

If BRH_PANDA or BRH_SUPERCIRCLE is specified, *cSize2* is the y dimension of the pattern used, in pels.

ulColorEncoding (ULONG)
Color encoding.

BCE_RGB Each element in the color array is an RGB2 datatype. This is the default value.

ulIdentifier (ULONG)
Reserved for application use.

argb2Color[1] (RGB2)
Array of RGB values.

This is a packed array of 24-bit RGB values. If there are N bits per pel (N = the array contains 2**N RGB values. However, if N = 24 the bit map does not need the *color* array because the standard-format bit map, with 24 bits per pel, is assumed to contain RGB values.

BITMAPINFOHEADER Bit-map information header structure.

Each bit plane logically contains (*cx* cy* cBitCount*) bits, although the actual length can be greater because of padding.

See also BITMAPINFOHEADER2, which is preferred.

```
typedef struct _BITMAPINFOHEADER {
ULONG     cbFix;
USHORT    cx;
USHORT    cy;
USHORT    cPlanes;
USHORT    cBitCount;
 } BITMAPINFOHEADER;
```

cbFix (ULONG)
Length of structure.

cx (USHORT)
Bit-map width in pels.

cy (USHORT)
Bit-map height in pels.

cPlanes (USHORT)
Number of bit planes.

cBitCount (USHORT)
Number of bits per pel within a plane.

BITMAPINFOHEADER2 Bit-map information header structure.

Each bit plane logically contains (*cx * cy * cBitCount*) bits, although the actual length can be greater because of padding.

Note: Many functions can accept either this structure or the BITMAPINFOHEADER structure. Where possible, use BITMAPINFOHEADER2.

```
typedef struct _BITMAPINFOHEADER2 {
ULONG     cbFix;
ULONG     cx;
ULONG     cy;
USHORT    cPlanes;
USHORT    cBitCount;
ULONG     ulCompression;
ULONG     cbImage;
ULONG     cxResolution;
ULONG     cyResolution;
ULONG     cclrUsed;
ULONG     cclrImportant;
USHORT    usUnits;
USHORT    usReserved;
USHORT    usRecording;
USHORT    usRendering;
ULONG     cSize1;
ULONG     cSize2;
ULONG     ulColorEncoding;
ULONG     ulIdentifier;
 } BITMAPINFOHEADER2;
```

cbFix (ULONG)
Length of structure.

The structure can be truncated after *cBitCount* or any subsequent field.

cx (ULONG)
Bit-map width in pels.

cy (ULONG)
Bit-map height in pels.

cPlanes (USHORT)
Number of bit planes.

cBitCount (USHORT)
Number of bits per pel within a plane.

ulCompression (ULONG)
Compression scheme used to store the bit map:

BCA_UNCOMP	Bit map is uncompressed.
BCA_HUFFMAN1D	The bit map is compressed by a modified Huffman encoding. This is valid for a bi-level (one bit per pel) bit map.
BCA_RLE4	The bit map is a 4-bit per pel run-length encoded bit map. See below for a description of the format of the compressed data.
BCA_RLE8	The bit map is a 8-bit per pel run-length encoded bit map. See below for a description of the format of the compressed data.
BCA_RLE24	The bit map is a 24-bit per pel run-length encoded bit map. See below for a description of the format of the compressed data.

cbImage (ULONG)
Length of bit-map storage data, in bytes.

If the bit map is uncompressed, zero (the default) can be specified for this.

cxResolution (ULONG)
Horizontal component of the resolution of target device.

The resolution of the device the bit map is intended for, in the units specified by *usUnits*. This information enables applications to select from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution (ULONG)
Vertical component of the resolution of target device.

See the description of *cxResolution*.

cclrUsed (ULONG)
Number of color indexes used.

The number of color indexes from the color table that are used by the bit map. If this is zero (the default), all the indexes are used. If it is non-zero, only the first *cclrUsed* entries in the table are accessed by the system, and further entries can be omitted.

For the standard formats with a *cBitCount* of 1, 4, or 8 (and *cPlanes* equal to 1), any indexes beyond *cclrUsed* are invalid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, *cclrUsed* is the number of colors used by the bit map.

cclrImportant (ULONG)
Minimum number of color indexes for satisfactory appearance of the bit map.

More colors may be used in the bit map, but it is not necessary to assign them to the device palette. These additional colors may be mapped to the nearest colors available.

Zero (the default) means that all entries are important.

For a 24-bitcount standard format bit map, the *cclrImportant* colors are also listed in the color table relating to this bit map.

usUnits (USHORT)
Units of measure.

Units of measure of the horizontal and vertical resolution, *cxResolution* and *cyResolution*.

BRU_METRIC Pels per meter. This is the default value.

usReserved (USHORT)
Reserved.

This is a reserved field. If present, it must be zero.

usRecording (USHORT)
Recording algorithm.

The format in which the bit-map data is recorded.

BRA_BOTTOMUP Scan lines are recorded bottom-to-top. This is the default value.

usRendering (USHORT)
Halftoning algorithm.

The algorithm used to record bit-map data that has been digitally halftoned.

BRH_NOTHALFTONED Bit-map data is not halftoned. This is the default value.

BRH_ERRORDIFFUSION Error Diffusion or Damped Error Diffusion algorithm.

BRH_PANDA Processing Algorithm for Non-coded Document Acquisition.

BRH_SUPERCIRCLE Super Circle algorithm.

cSize1 (ULONG)
Size value 1.

If BRH_ERRORDIFFUSION is specified in *usRendering*, *cSize1* is the error damping as a percentage in the range 0 through 100. A value of 100%

indicates no damping, and a value of 0% indicates that any errors are not diffused.

If BRH_PANDA or BRH_SUPERCIRCLE is specified, *cSize1* is the x dimension of the pattern used, in pels.

cSize2 (ULONG)
Size value 2.

If BRH_ERRORDIFFUSION is specified in *usRendering*, this parameter is ignored.

If BRH_PANDA or BRH_SUPERCIRCLE is specified, *cSize2* is the y dimension of the pattern used, in pels.

ulColorEncoding (ULONG)
Color encoding.

BCE_RGB Each element in the color array is an RGB2 datatype. This is the default value.

ulIdentifier (ULONG)
Reserved for application use.

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 25-11 and "BKM_QUERYTABTEXT" on page 25-12 for information about those messages.

```
typedef struct _BOOKTEXT {
PSZ       pszString;
USHORT    textLen;
 } BOOKTEXT;
```

pszString (PSZ)
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 (USHORT)
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.

BOOL

Boolean.

Valid values are FALSE, which is 0, and TRUE, which is 1.

```
typedef unsigned long BOOL;
```

BTNCDATA

Button-control-data structure.

```
typedef struct _BTNCDATA {
USHORT     cb;
USHORT     fsCheckState;
USHORT     fsHiliteState;
LHANDLE    hImage;
 } BTNCDATA;
```

cb (USHORT)
Length of the control data in bytes.

8 The length of the control data for a button control.

fsCheckState (USHORT)
Check state of button.

This is the same value as returned by the BM_QUERYCHECK message and passed to the BM_SETCHECK message.

fsHiliteState (USHORT)
Highlighting state of button.

This is the same value as returned by the BM_QUERYHILITE message and passed to the BM_SETHILITE message.

hImage (LHANDLE)
Resource handle for icon or bit map.

BYTE

Byte.

```
typedef unsigned char BYTE;
```

CATCHBUF

Saved execution environment buffer.

```
typedef struct _CATCHBUF {
ULONG    reserved[7];
 } CATCHBUF;
```

reserved[7] (ULONG)
Save area.

CDATE

Structure that contains date information for a data element in the details view of a container control.

```
typedef struct _CDATE {
UCHAR     day;
UCHAR     month;
USHORT    year;
 } CDATE;
```

day (UCHAR)
Day.

month (UCHAR)
Month.

year (USHORT)
Year.

CELL

Class specific cell data follows immediately afterwards.

```
typedef struct _CELL {
ULONG    cbData;
 } CELL;
```

cbData (ULONG)
Size of the data that follows.

Class specific cell data follows immediately afterwards. For example the font palette would store the ASCII name of the font, and the color palette would store the RGB color of the cell.

CHAR

Single-byte character.

```
#define CHAR char
```

CHARBUNDLE

Character-attributes bundle structure.

```
typedef struct _CHARBUNDLE {
LONG      lColor;
LONG      lBackColor;
USHORT    usMixMode;
USHORT    usBackMixMode;
USHORT    usSet;
USHORT    usPrecision;
SIZEF     sizfxCell;
POINTL    ptlAngle;
POINTL    ptlShear;
USHORT    usDirection;
USHORT    usTextAlign;
FIXED     fxExtra;
FIXED     fxBreakExtra;
 } CHARBUNDLE;
```

lColor (LONG)
Character foreground color.

lBackColor (LONG)
Character background color.

usMixMode (USHORT)
Character foreground-mix mode.

usBackMixMode (USHORT)
Character background-mix mode.

usSet (USHORT)
Character set.

usPrecision (USHORT)
Character precision.

sizfxCell (SIZEF)
Character cell size.

ptlAngle (POINTL)
Character angle.

ptlShear (POINTL)
Character shear.

usDirection (USHORT)
Character direction.

usTextAlign (USHORT)
Text alignment.

fxExtra (FIXED)
Character extra.

fxBreakExtra (FIXED)
Character break extra.

CLASSINFO

Class-information structure.

```
typedef struct _CLASSINFO {
ULONG    flClassStyle;
PFNWP    pfnWindowProc;
ULONG    cbWindowData;
 } CLASSINFO;
```

flClassStyle (ULONG)
Class-style flags.

pfnWindowProc (PFNWP)
Window procedure.

cbWindowData (ULONG)
Number of additional window words.

CLASSDETAILS

Class details data structure.

```
typedef struct _CLASSDETAILS {
PSZ      pszAttribute;
PVOID    pSortRecord;
 } CLASSDETAILS;
```

pszAttribute (PSZ)

Translatable string for a class attribute.

pSortRecord (PVOID)

Function pointer for sort function for attribute.

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" on page 24-10
- "CN_DRAGLEAVE" on page 24-11
- "CN_DRAGOVER" on page 24-12
- "CN_DROP" on page 24-13
- "CN_DROPHELP" on page 24-14

```
typedef struct _CNRDRAGINFO {
PDRAGINFO      pDragInfo;
PRECORDCORE    pRecord;
 } CNRDRAGINFO;
```

pDragInfo (PDRAGINFO)
Pointer.

Pointer to a DRAGINFO structure.

pRecord (PRECORDCORE)
Pointer.

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 CMA_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" on page 24-18 for information about that notification code.

```
typedef struct _CNRDRAGINIT {
HWND           hwndCnr;
PRECORDCORE    pRecord;
LONG           x;
LONG           y;
LONG           cx;
LONG           cy;
 } CNRDRAGINIT;
```

hwndCnr (HWND)
Container control handle.

pRecord (PRECORDCORE)
Pointer.

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.

X-coordinate of the pointer of the pointing device in desktop coordinates.

y (LONG)
Y-coordinate.

Y-coordinate of the pointer of the pointing device in desktop coordinates.

cx (LONG)
X-offset.

X-offset from the hot spot of the pointer of the pointing device (in pels) to the record origin.

cy (LONG)
Y-offset.

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 24-6 for information about that message.

```
typedef struct _CNRDRAWITEMINFO {
PRECORDCORE    pRecord;
PFIELDINFO     pFieldInfo;
 } CNRDRAWITEMINFO;
```

pRecord (PRECORDCORE)
Pointer.

Pointer to the RECORDCORE structure for the record that is 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.

Pointer to the FIELDINFO structure for the container column that is 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" on page 24-8
- "CN_ENDEDIT" on page 24-15

- "CN_REALLOCPSZ" on page 24-20
- "CM_OPENEDIT" on page 24-35

```
typedef struct _CNREDITDATA {
ULONG          cb;
HWND           hwndCnr;
PRECORDCORE    pRecord;
PFIELDINFO     pFieldInfo;
PPSZ           ppszText;
ULONG          cbText;
ULONG          id;
 } CNREDITDATA;
```

cb (ULONG)
Structure size.

The size (in bytes) of the CNREDITDATA data structure.

hwndCnr (HWND)
Container window handle.

pRecord (PRECORDCORE)
Pointer or NULL.

Pointer to a RECORDCORE data structure. 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 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 (PPSZ)
Pointer or NULL.

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.

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)
Window ID.

ID of the window to be edited. The ID can be one of the following:

application-defined container identifier	Container window.
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.

CNRINFO

Structure that contains information about the container.

```
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;
```

cb (ULONG)
Structure size.

The size (in bytes) of the CNRINFO data structure.

pSortRecord (PVOID)
Pointer or NULL.

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" on page 24-51 for a further description of the comparison function.

pFieldInfoLast (PFIELDINFO)
Pointer or NULL.

Pointer to 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.

pFieldInfoObject (PFIELDINFO)
Pointer.

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 A-17 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 A-17 and CV_FLOW on page A-17 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 A-17 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) message.

- 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).

COLOR

Color value.

```
typedef long COLOR;
```

CONVCONTEXT

Dynamic-data-exchange conversation context structure.

```
typedef struct _CONVCONTEXT {
ULONG    cb;
ULONG    ulContext;
ULONG    ulCountry;
ULONG    ulCodepage;
ULONG    usLangID;
ULONG    usSubLangID;
 } CONVCONTEXT;
```

cb (ULONG)
Length of structure.

This must be set to the length of the CONVCONTEXT structure.

ulContext (ULONG)
Options.

DDECTXT_CASESENSITIVE All strings in this conversation are case sensitive.

ulCountry (ULONG)
Country code.

ulCodepage (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.

CPTEXT

String structure containing the code-page and language of the string.

```
typedef struct _CPTEXT {
USHORT    idCountry;
USHORT    usCodepage;
USHORT    usLangID;
USHORT    usSubLangID;
BYTE      abText[1];
 } CPTEXT;
```

idCountry (USHORT)
Country code.

usCodepage (USHORT)
Code-page identity.

usLangID (USHORT)
Language.

Zero is valid and means no language information.

usSubLangID (USHORT)
Sub-language.

Zero is valid and means no sub-language information.

abText[1] (BYTE)
Zero-terminated text string.

CREATESTRUCT

Create-window data structure.

```
typedef struct _CREATESTRUCT {
PVOID    pPresParams;
PVOID    pCtlData;
ULONG    id;
HWND     hwndInsertBehind;
HWND     hwndOwner;
LONG     cy;
LONG     cx;
LONG     y;
LONG     x;
ULONG    flStyle;
PSZ      pszText;
PSZ      pszClass;
HWND     hwndParent;
 } CREATESTRUCT;
```

pPresParams (PVOID)
Presentation parameters.

pCtlData (PVOID)
Control data.

id (ULONG)
Window identifier.

hwndInsertBehind (HWND)
Window behind which the window is to be placed.

hwndOwner (HWND)
Window owner.

cy (LONG)
Window height.

cx (LONG)
Window width.

y (LONG)
y-coordinate of origin.

x (LONG)
x coordinate of origin.

flStyle (ULONG)
Window style.

pszText (PSZ)
Window text.

pszClass (PSZ)
Registered window class name.

hwndParent (HWND)
Parent window handle.

CTIME

Structure that contains time information for a data element in the details view of a container control.

```
typedef struct _CTIME {
UCHAR    hours;
UCHAR    minutes;
UCHAR    seconds;
UCHAR    ucReserved;
 } CTIME;
```

hours (UCHAR)
Hour.

minutes (UCHAR)
Minute.

seconds (UCHAR)
Second.

ucReserved (UCHAR)
Reserved.

CURSORINFO

Cursor-information structure.

```
typedef struct _CURSORINFO {
HWND     hwnd;
LONG     x;
LONG     y;
LONG     cx;
LONG     cy;
ULONG    fs;
RECTL    rclClip;
 } CURSORINFO;
```

hwnd (HWND)
Window handle.

x (LONG)
x coordinate.

y (LONG)
y coordinate.

cx (LONG)
Cursor width.

cy (LONG)
Cursor height.

fs (ULONG)
Options.

rclClip (RECTL)
Cursor box.

DDEINIT

Dynamic-data-exchange initiation structure.

```
typedef struct _DDEINIT {
ULONG    cb;
PSZ      pszAppName;
PSZ      pszTopic;
ULONG    offConvContext;
 } DDEINIT;
```

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.

```
typedef struct _DDESTRUCT {
ULONG     ulData;
USHORT    usStatus;
USHORT    usFormat;
USHORT    offszItemName;
USHORT    offabData;
 } DDESTRUCT;
```

ulData (ULONG)
Total length.

This is the length of this structure plus the item name and data, which occur after the *offabData* parameter.

usStatus (USHORT)
Status.

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.

This is the offset to the item name referred to in this message, from the start of this structure.

offabData (USHORT)
Offset to beginning of data.

This is the offset to the data, from the start of this structure.

For compatibility reasons, this data should not contain embedded pointers. Offsets should be used instead.

DELETENOTIFY

Structure that contains information about the application page that is being deleted from a notebook.

```
typedef struct _DELETENOTIFY {
HWND        hwndBook;
HWND        hwndPage;
ULONG       ulAppPageData;
HBITMAP     hbmTab;
 } DELETENOTIFY;
```

hwndBook (HWND)
Notebook window handle.

hwndPage (HWND)
Application page window handle.

ulAppPageData (ULONG)
Application-specified page data.

hbmTab (HBITMAP)
Application-specified tab bit map.

DESKTOP

Desktop background state structure.

```
typedef struct _DESKTOP {
ULONG       cbSize;
HBITMAP     hbm;
LONG        x;
LONG        y;
ULONG       fl;
CHAR        szFile[MAX_FILENAME];
LONG        lTileCount;
 } DESKTOP;
```

cbSize (ULONG)
Length of structure.

hbm (HBITMAP)
Bit-map handle of desktop background.

x (LONG)
x desktop coordinate of the origin of the bit map.

y (LONG)
y desktop coordinate of the origin of the bit map.

fl (ULONG)
Desktop background state indicators or setting options:

SDT_CENTER	The desktop background bit map is, or is to be, centered on the screen. If this option is specified, then the values of the *x* the *y* parameters are inapplicable.
SDT_DESTROY	Any existing desktop background bit map is to be destroyed. The setting of this option is not returned on the WinQueryDesktopBkgnd function.

SDT_NOBKGND There is no desktop background bit map, that is the desktop background i a solid color. For the WinQueryDesktopBkgnd function the existing background is to be left unmodified unless SDT_DESTROY is also specified.

SDT_PATTERN The bit map represents a fill pattern.

SDT_RETAIN The *szFile*[*MAX_FILENAME*] is, or is to be, remembered for use when the system is started.

SDT_SCALE The bit map is, or is to be, scaled to fill the desktop. If this option is specified, then the values of the *x* and *y* parameters are inapplicable.

SDT_TILE The bit map is, or is to be, tiled to fill the desktop.

SDT_LOADFILE For the WinSetDesktopBkgnd function the bit map is to be loaded from the filename specified. If the SDT_NOBKGND flag is also set then the bit map is loaded but the background is not set. Tiling and scaling may be performed at load time or later when setting the bit map.

szFile[MAX_FILENAME] (CHAR)
Zero-terminated name of the file containing the bit map.

lTileCount (LONG)
Number of images of the bit map to be tiled.

The tile count is the number of images to be drawn in the vertical and horizontal direction when tiling the desktop background.

DEVOPENSTRUC Open-device data structure.

```
typedef struct _DEVOPENSTRUC {
PSZ          pszLogAddress;
PSZ          pszDriverName;
PDRIVDATA    pdriv;
PSZ          pszDataType;
PSZ          pszComment;
PSZ          pszQueueProcName;
PSZ          pszQueueProcParams;
PSZ          pszSpoolerParams;
PSZ          pszNetworkParams;
 } DEVOPENSTRUC;
```

pszLogAddress (PSZ)
Logical address.

This is required for an OD_DIRECT device being opened with DevOpenDC; it is the logical device address, such as "LPT1" on OS/2. Some drivers may accept a file name for this parameter, or even a named pipe. A driver can restrict the logical address to certain names because special hardware is involved; for example a printer driver that uses shared memory to access the memory of a laser printer.

Where output is to be queued (for an OD_QUEUED device), this is the name of the queue for the output device, and must always be supplied if it is not available from *pszToken*. The queue name can be a UNC name.

pszDriverName (PSZ)
Driver name.

A string containing the name of the Presentation Manager* (PM) device driver (for example, "IBM4019"). This information must always be supplied if it is not available from *pszToken*.

---

* Trademark of IBM Corporation

pdriv (PDRIVDATA)
Driver data.

Data that is to be passed directly to the PM device driver. Whether any of this is required depends upon the device driver.

pszDataType (PSZ)
Data type.

For a OD_QUEUED or OD_DIRECT device, this parameter defines the type of data that is to be (or was) queued as follows:

PM_Q_STD Standard format
PM_Q_RAW Raw format.

Note that a device driver can define other data types.

With DevOpenDC, for both of the above device types the default is supplied by the device driver if *pszDataType* is not specified. For any other device type, *pszDataType* is ignored.

pszComment (PSZ)
Comment.

This is a natural language description of the file for queued output, For example, this can be displayed by the spooler to the user, and is optional.

pszQueueProcName (PSZ)
Queue-processor name.

This is the name of the queue processor for queued output, and is usually the default.

pszQueueProcParams (PSZ)
Queue-processor parameters.

This is a parameter string for the queue processor, for queued output, and is optional.

pszSpoolerParams (PSZ)
Spooler parameters.

This is a parameter string for the spooler for queued output, and is optional. It has the following options, which must be separated by one or more blanks:

FORM=f Specifies a form name 'f'. This must be a valid form name for the printer. If more than one form is needed for the job, all of the required form names are supplied, separated by commas, as FORM=aaaa,bbbb,cccc; the first one is the one that is to be used first. See HCINFO.

A form name can be enclosed in double quotes to permit form names to contain the characters ',' ';' or '='. For example,

```
FORM="A","A4 with heading","C,D"
```

calls for three forms: 'A', 'A4 with heading' and 'C,D'. If a double quote is part of a form name, it should be supplied twice.

If this option is not specified, the data is printed on the forms in use, when this print job is ready to be printed.

PRTY=n Specifies a priority in the range 1 through 99, with 99 being the highest. If it is not specified, a default priority of 50 is used.

pszNetworkParams (PSZ)
Network parameters.

This is a parameter string for the network program for queued output, and is optional. The format of the parameter string is keyword=value, and the following keyword is defined (additional ones can be defined by the network program):

USER=u Specifies the user identifier 'u'. If it is not specified, a null user identifier is used.

DLGTEMPLATE

Dialog-template structure.

```
typedef struct _DLGTEMPLATE {
USHORT      cbTemplate;
USHORT      type;
USHORT      codepage;
USHORT      offadlgti;
USHORT      fsTemplateStatus;
USHORT      iItemFocus;
USHORT      coffPresParams;
DLGTITEM    adlgti[1];
 } DLGTEMPLATE;
```

cbTemplate (USHORT)
Length of template.

type (USHORT)
Template format type.

codepage (USHORT)
Code page.

offadlgti (USHORT)
Offset to dialog items.

fsTemplateStatus (USHORT)
Template status.

iItemFocus (USHORT)
Index of item to receive focus initially.

coffPresParams (USHORT)
Count of presentation-parameter offsets.

adlgti[1] (DLGTITEM)
Start of dialog items.

DLGTITEM

Dialog-item structure.

```
typedef struct _DLGTITEM {
USHORT    fsItemStatus;
USHORT    cChildren;
USHORT    cchClassName;
USHORT    offClassName;
USHORT    cchText;
USHORT    offText;
ULONG     flStyle;
SHORT     x;
SHORT     y;
SHORT     cx;
SHORT     cy;
USHORT    id;
USHORT    offPresParams;
USHORT    offCtlData;
 } DLGTITEM;
```

fsItemStatus (USHORT)
Status.

cChildren (USHORT)
Count of children to this dialog item.

cchClassName (USHORT)
Length of class name.

If zero, *offClassName* contains the hexadecimal equivalent of a preregistered class name.

offClassName (USHORT)
Offset to class name.

If *cchClassName* is nonzero, this is the offset to a null-terminated ASCII string that contains the classname. If *cchClassName* is zero, this is of the form 0xhhhh, where hhhh is the hexadecimal equivalent of the preregistered class name.

cchText (USHORT)
Length of text.

offText (USHORT)
Offset to text.

flStyle (ULONG)
Dialog item window style.

The high-order 16 bits are the standard WS_* style bits. The low-order 16 bits are available for class-specific use.

x (SHORT)
x-coordinate of origin of dialog-item window.

y (SHORT)
y-coordinate of origin of dialog-item window.

cx (SHORT)
Dialog-item window width.

cy (SHORT)
Dialog-item window height.

id (USHORT)
Identity.

offPresParams (USHORT)
Reserved.

offCtlData (USHORT)
Offset to control data.

DRAGIMAGE

Dragged-object-image structure.

```
typedef struct _DRAGIMAGE {
USHORT     cb;
USHORT     cptl;
LHANDLE    hImage;
SIZEL      sizlStretch;
ULONG      fl;
SHORT      cxOffset;
SHORT      cyOffset;
 } DRAGIMAGE;
```

cb (USHORT)
Structure size.

Size, in bytes, of the DRAGIMAGE structure.

cptl (USHORT)
Number of points.

The number of points in the point array if *fl* is specified as DRG_POLYGON.

hImage (LHANDLE)
Image handle.

Handle representing the image to display. The type is determined by *fl*.

sizlStretch (SIZEL)
Dimensions for stretching.

Specifies the 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.

X-offset from the pointer hot spot to the origin of the image.

cyOffset (SHORT)
Y-offset.

Y-offset from the pointer hot spot to the origin of the image.

DRAGINFO

Drag-information structure.

```
typedef struct _DRAGINFO {
ULONG     ulDraginfo;
USHORT    usDragitem;
SHORT     usOperation;
HWND      hwndSource;
SHORT     xDrop;
SHORT     yDrop;
USHORT    cditem;
USHORT    usReserved;
 } DRAGINFO;
```

ulDraginfo (ULONG)
Structure size.

Size, in bytes, of the structure. The size includes the array of DRAGITEM structures.

usDragitem (USHORT)
DRAGITEM structures sizes.

Size, in bytes, of each DRAGITEM structure.

usOperation (SHORT)
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.

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.

Window handle of the source of the drag operation.

xDrop (SHORT)
X-coordinate.

X-coordinate of drop point expressed in desktop coordinates.

yDrop (SHORT)
Y-coordinate.

Y-coordinate of drop point expressed in desktop coordinates.

cdItem (USHORT)
Count of DRAGITEM structures.

usReserved (USHORT)
Reserved.

DRAGITEM

Drag-object structure.

```
typedef struct _DRAGITEM {
HWND      hwndItem;
ULONG     ulItemID;
HSTR      hstrType;
HSTR      hstrRMF;
HSTR      hstrContainerName;
HSTR      hstrSourceName;
HSTR      hstrTargetName;
SHORT     cxOffset;
SHORT     cyOffset;
USHORT    usControl;
USHORT    usSupportedOps;
 } DRAGITEM;
```

hwndItem (HWND)
Window handle.

Window handle of the source of the drag operation.

ulItemID (ULONG)
Item information.

Information used by the source to identify the object being dragged.

hstrType (HSTR)
String handle.

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 version 2.0 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.

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:

1. <mechanism(1),format(1)>
2. (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.

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.

String handle of the name of the source object. The string handle must be created using the DrgAddStrHandle function.

hstrTargetName (HSTR)
String handle.

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.

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.

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.

usControl (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.

usSupportedOps (USHORT)
Supported operations.

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.

```
typedef struct _DRAGTRANSFER {
ULONG        cb;
HWND         hwndClient;
PDRAGITEM    pditem;
HSTR         hstrSelectedRMF;
HSTR         hstrRenderToName;
ULONG        ulTargetInfo;
USHORT       usOperation;
USHORT       usReply;
 } DRAGTRANSFER;
```

cb (ULONG)
Structure size.

Size, in bytes, of the structure.

hwndClient (HWND)
Window handle.

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.

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.

The 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.

A string handle representing the name where the source will place, and the target will find, 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* 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.

usReply (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.

DRIVDATA

Driver-data structure.

```
typedef struct _DRIVDATA {
LONG    cb;
LONG    lVersion;
CHAR    szDeviceName[32];
CHAR    abGeneralData[1];
 } DRIVDATA;
```

cb (LONG)
Length.

The length of the structure.

lVersion (LONG)
Version.

The version number of the data. Version numbers are defined by particular PM device drivers.

szDeviceName[32] (CHAR)
Device name.

A string in a 32-byte field, identifying the particular device (model

number, and so on). Again, valid values are defined by PM device drivers.

abGeneralData[1] (CHAR)
General data.

Data as defined by the Presentation Manager device driver.

The data type of this field is defined by the Presentation Manager device driver. It does not contain pointers, as these are not necessarily valid when passed to the device driver.

DRIVPROPS Printer driver property structure.

```
typedef struct _DRIVPROPS {
PSZ      pszKeyName;
ULONG    cbBuf;
PVOID    pBuf;
 } DRIVPROPS;
```

pszKeyName (PSZ)
key name

This is the key name for an individual property. For example "FORMS."

cbBuf (ULONG)
The length of the key data.

pBuf (PVOID)
The key data.

This is the data associated with the key name. For example "LETTER, LEGAL, LEDGER."

ENTRYFDATA Entry-field control data structure.

```
typedef struct _ENTRYFDATA {
USHORT    cb;
USHORT    cchEditLimit;
USHORT    ichMinSel;
USHORT    ichMaxSel;
 } ENTRYFDATA;
```

cb (USHORT)
Length of control data in bytes.

8 The length of the control data for an entry field control.

cchEditLimit (USHORT)
Edit limit.

This is the maximum number of characters that can be entered into the entry field control.

If the operator tries to enter more text into an entry field control than is specified by the text limit set by the EM_SETTEXTLIMIT message, the entry field control indicates the error by sounding the alarm and does not accept the characters.

ichMinSel (USHORT)
Minimum selection.

ichMaxSel (USHORT)
Maximum selection.

The *ichMinSel* and *ichMaxSel* parameters identify the current selection within the entry field control. Characters within the text with byte offsets less than the *ichMaxSel* parameter and greater than or equal to the *ichMinSel* parameter are the current selection. The cursor is positioned immediately before the character identified by the *ichMaxSel* parameter.

If the *ichMinSel* parameter is equal to the *ichMaxSel* parameter, the current selection becomes the insertion point.

If the *ichMinSel* parameter is equal to 0 and the *ichMaxSel* is greater than or equal to text limit set by the EM_SETTEXTLIMIT message, the entire text is selected.

ERRINFO

Error-information structure.

```
typedef struct _ERRINFO {
ULONG      cbFixedErrInfo;
ERRORID    idError;
ULONG      cDetailLevel;
ULONG      offaoffszMsg;
ULONG      ulBinaryData;
 } ERRINFO;
```

cbFixedErrInfo (ULONG)
Length of fixed data to this structure.

idError (ERRORID)
Error identity.

This is identical to the value returned by the WinGetLastError function.

cDetailLevel (ULONG)
Number of levels of detail.

This is the number of entries in the array of words pointed to by the following field. One level of detail is provided.

offaoffszMsg (ULONG)
Offset to the array of message offsets.

ulBinaryData (ULONG)
Offset to the binary data.

This can contain additional information relating to the error.

ERRORID

Error identity.

```
typedef ULONG ERRORID;
```

ESCSETMODE

Structure for setting printer mode. See DevEscape (DEVESC_SETMODE).

```
typedef struct _ESCSETMODE {
ULONG     mode;
USHORT    codepage;
 } ESCSETMODE;
```

mode (ULONG)
Mode

Mode to be set.

0 Set mode to specified code page. Any font can be used.

codepage (USHORT)
Code page.

If zero is specified for the code page, the printer is set to the hardware default.

FACENAMEDESC

Face-name description structure. See GpiQueryFaceString.

```
typedef struct _FACENAMEDESC {
USHORT    usSize;
USHORT    usWeightClass;
USHORT    usWidthClass;
USHORT    usReserved;
ULONG     flOptions;
 } FACENAMEDESC;
```

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.

```
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;
```

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), 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 (only), any white space generated by the character extra and character break extra attributes (see GpiSetCharExtra and GpiSetCharBreakExtra), as well as increments provided by the vector of increments on GpiCharStringPos and GpiCharStringPosAt, is 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 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) 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/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 pels.
FATTR_FONTUSE_TRANSFORMABLE	Characters can be transformed (for example, scaled, rotated, or sheared).

FDATE

Date data structure for file-system functions.

```
typedef struct _FDATE {
USHORT    usday;
USHORT    usmonth;
USHORT    usyear;
 } FDATE;
```

usday (USHORT)

Binary day for directory entry.

usmonth (USHORT)

Binary month for directory entry.

usyear (USHORT)

Binary year for directory entry.

FFDESCS

Font-file descriptor.

```
typedef CHAR FFDESCS[2][FACESIZE];
```

FFDESCS2

Font-file descriptor.

```
typedef struct _FFDESCS2 {
ULONG    cbLength;
ULONG    cbFacenameOffset;
BYTE     abFamilyName[1];
 } FFDESCS2;
```

cbLength (ULONG)
Structure length.

cbLength is the overall length of the FFDESCS2 structure. It is always rounded up to a multiple of four.

cbFacenameOffset (ULONG)
Offset of Facename in the structure.

The facename is a null terminated string. It starts at *cbFacenameOffset* bytes offset into FFDESCS2.

abFamilyName[**1**] (BYTE)
Family name.

abFamilyName[*1*] is a null terminated string.

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.

```
typedef struct _FIELDINFO {
ULONG         cb;
ULONG         flData;
ULONG         flTitle;
PVOID         pTitleData;
ULONG         offStruct;
PVOID         pUserData;
PFIELDINFO    pNextFieldInfo;
ULONG         cxWidth;
 } FIELDINFO;
```

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_STRING
  Character or text data is displayed in this column.

  CFA_ULONG
  Unsigned number data is displayed in this column. National Language Support (NLS) is enabled for number format.

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 on page A-10.

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 on page A-22.

- Specify any or all of the following column attributes:

  CFA_HORZSEPARATOR
  A horizontal separator is provided beneath column headings.

  CFA_SEPARATOR
  A vertical separator is drawn after this column.

  CFA_OWNER
  Ownerdraw is enabled for this container column.

  CFA_INVISIBLE
  Invisible container column. The default is visible.

  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.

- Specify one of the following for each column to vertically position data in that column:

  CFA_TOP
  Top-justifies field data.

  CFA_BOTTOM
  Bottom-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)
Attributes of column headings.

- 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.

- 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.

Pointer to user data.

pNextFieldInfo (PFIELDINFO)
Pointer.

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 24-30 for information about that message.

```
typedef struct _FIELDINFOINSERT {
ULONG         cb;
PFIELDINFO    pFieldInfoOrder;
ULONG         cFieldInfoInsert;
ULONG         fInvalidateFieldInfo;
 } FIELDINFOINSERT;
```

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.

cFieldInfoInsert (ULONG)
Number of columns.

The number of FIELDINFO structures to be inserted. The *cFieldInfoInsert* field value must be greater than 0.

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.

FILEDLG

File-dialog structure.

```
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;
```

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 *hwndOwner*.
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.

pszlType (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 *papszlTypeList*.

papszlTypeList (PAPSZ)
Pointer.

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.

pszlDrive (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 *papszlDriveList*.

papszlDriveList (PAPSZ)
Pointer.

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 *pszlType*. 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.

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 *hwndParent*, 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 *hwndParent*, 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.

FILEFINDBUF4

32-bit level 2 information (used with EAs).

```
typedef struct _FILEFINDBUF4 {
ULONG    uloNextEntryOffset;
FDATE    fdateCreation;
FTIME    ftimeCreation;
FDATE    fdateLastAccess;
FTIME    ftimeLastAccess;
FDATE    fdateLastWrite;
FTIME    ftimeLastWrite;
ULONG    ulcbFile;
ULONG    ulcbFileAlloc;
ULONG    ulattrFile;
ULONG    ulcbList;
UCHAR    uccchName;
CHAR     chachName[CCHMAXPATHCOMP];
 } FILEFINDBUF4;
```

uloNextEntryOffset (ULONG)

fdateCreation (FDATE)

ftimeCreation (FTIME)

fdateLastAccess (FDATE)

ftimeLastAccess (FTIME)

fdateLastWrite (FDATE)

ftimeLastWrite (FTIME)

ulcbFile (ULONG)

ulcbFileAlloc (ULONG)

ulattrFile (ULONG)

ulcbList (ULONG)

uccchName (UCHAR)

chachName[**CCHMAXPATHCOMP**] (CHAR)

FIXED

Signed-integer fraction (16:16). This can be treated as a LONG where the value has been multiplied by 65 536.

```
typedef LONG FIXED;
```

FOLDERDATA

FOLDERDATA data structure.

```
typedef struct _FOLDERDATA {
WPFolder *Folder;
USEITEM       pUseItem;
VIEWITEM      pViewItem;
ULONG         ulView;
HWND          hwndCnr;
HWND          hwndCtxtMenu;
PSZ           pszEditName;
PVOID         precEditName;
PRECORDCORE   pRecordContextMenu;
 } FOLDERDATA;
```

Folder (WPFolder *)

Pointer to folder object.

pUseItem (USEITEM)

Folder object's INUSE list item.

pViewItem (VIEWITEM)

Folder object's view information.

ulView (ULONG)

Folder type.

hwndCnr (HWND)

Container control window handle.

hwndCtxtMenu (HWND)

Pop-up menu handle.

pszEditName (PSZ)

A pointer to direct name edit string. Used only during direct name edit.

precEditName (PVOID)

A pointer to direct name edit record. Used only during direct name edit.

pRecordContextMenu (PRECORDCORE)

A pointer to object record of last pop-up menu.

FONTDLG

Font-dialog structure.

```
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;
SHORT      sNominalPointSize;
USHORT     usWeight;
USHORT     usWidth;
SHORT      x;
SHORT      y;
USHORT     usDlgId;
USHORT     usFamilyBufLen;
FATTRS     fAttrs;
 } FONTDLG;
```

cbSize (ULONG)

Structure size.

Size of structure. 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: Take care 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 *hwndOwner* 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.

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 *hwndParent*, 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 *hwndParent*, 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)
Buffer size.

Size of the buffer passed in the *pszFamilyname* field.

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.

FONTMETRICS Font-metrics structure.

This structure is returned to applications on the GpiQueryFonts and GpiQueryFontMetrics calls and conveys information from the font creator to the application.

```
typedef struct _FONTMETRICS {
CHAR      szFamilyname[FACESIZE];
CHAR      szFacename[FACESIZE];
USHORT    usRegistry;
USHORT    usCodePage;
LONG      lEmHeight;
LONG      lXHeight;
LONG      lMaxAscender;
LONG      lMaxDescender;
LONG      lLowerCaseAscent;
LONG      lLowerCaseDescent;
LONG      lInternalLeading;
LONG      lExternalLeading;
LONG      lAveCharWidth;
LONG      lMaxCharInc;
LONG      lEmInc;
LONG      lMaxBaselineExt;
SHORT     sCharSlope;
SHORT     sInlineDir;
SHORT     sCharRot;
USHORT    usWeightClass;
USHORT    usWidthClass;
SHORT     sXDeviceRes;
SHORT     sYDeviceRes;
SHORT     sFirstChar;
SHORT     sLastChar;
SHORT     sDefaultChar;
SHORT     sBreakChar;
SHORT     sNominalPointSize;
SHORT     sMinimumPointSize;
SHORT     sMaximumPointSize;
USHORT    usType;
USHORT    usDefn;
USHORT    usSelection;
USHORT    usCapabilities;
LONG      lSubscriptXSize;
LONG      lSubscriptYSize;
```

```
LONG      lSubscriptXOffset;
LONG      lSubscriptYOffset;
LONG      lSuperscriptXSize;
LONG      lSuperscriptYSize;
LONG      lSuperscriptXOffset;
LONG      lSuperscriptYOffset;
LONG      lUnderscoreSize;
LONG      lUnderscorePosition;
LONG      lStrikeoutSize;
LONG      lStrikeoutPosition;
SHORT     sKerningPairs;
SHORT     sFamilyClass;
LONG      lMatch;
ATOM      FamilyNameAtom;
ATOM      FaceNameAtom;
PANOSE    panPanose;
 } FONTMETRICS;
```

szFamilyname[FACESIZE] (CHAR)
Family name.

The family name of the font that describes the basic appearance of the font, for example, Times New Roman**. This string is null terminated, and is thus limited to 31 characters in length. Longer names may be retrieved by using the *FamilyNameAtom* field to retrieve the full name from the System Atom table.

szFacename[FACESIZE] (CHAR)
Face name.

The typeface name that defines the particular font, for example, Times New Roman Bold Italic. This string is null terminated, and is thus limited to 31 characters in length. Longer names may be retrieved by using the *FaceNameAtom* field to retrieve the full name from the System Atom table.

usRegistry (USHORT)
Registry identifier.

The IBM registered number (or zero).

usCodePage (USHORT)
Code page.

Defines the registered code page supported by the font. For example, the original IBM PC code page is 437. A value of 0 implies that the font may be used with any of the OS/2* supported code pages.

Where a font contains special symbols for which there is no registered code page, then code page 65400 is used.

lEmHeight (LONG)
Em height.

The height of the Em square in world coordinate units. This corresponds to the point size for the font.

lXHeight (LONG)
x height.

The nominal height above the baseline for lowercase characters (ignoring ascenders) in world coordinate units.

** Trademark of Monotype

* Trademark of IBM Corporation

lMaxAscender (LONG)
Maximum ascender.

The maximum height above the baseline reached by any part of any symbol in the font in world coordinate units. This field may exceed *lEmHeight*.

lMaxDescender (LONG)
Maximum descender.

The maximum depth below the baseline reached by any part of any symbol in the font in world coordinate units. This field may exceed *lEmHeight*.

lLowerCaseAscent (LONG)
Lowercase ascent.

The maximum height above the baseline reached by any part of any lowercase (Latin unaccented "a" through "z") symbol in the font in world coordinate units.

lLowerCaseDescent (LONG)
Lowercase descent.

The maximum depth below the baseline reached by any part of any lowercase (Latin unaccented "a" through "z") symbol in the font in world coordinate units.

lInternalLeading (LONG)
Internal leading.

The amount of space which, when subtracted from *lMaxAscender*, gives a font-design dependent, but glyph-set independent, measure of the distance above the baseline that characters extend. This calculation thus approximates the visual top to a row of characters without actually looking at the characters in the row.

The recommended use of this field by applications is to position the first line of a block of text by subtracting it from *lMaxAscender* and positioning the baseline that distance below whatever is above the text.

Note: This does not guarantee that characters will not overwrite things above them, but does give a font designer's view of where to place the text. Collision should be tested for, and additional space allocated if necessary.

lExternalLeading (LONG)
External leading.

The amount of guaranteed white space advised by the font designer to appear between adjacent rows of text. This value may be zero.

Note: The fonts built in to Presentation Manager have zero in this field.

lAveCharWidth (LONG)
Average character width.

This is determined by multiplying the width of each lowercase character by a constant, adding the products, and then dividing by 1000. The letters involved in this, plus their constants, are as follows:

Letter	Constant
a	64
b	14
c	27
d	35
e	100
f	20
g	14
h	42
i	63
j	3

k	6
l	35
m	20
n	56
o	56
p	17
q	4
r	49
s	56
t	71
u	31
v	10
w	18
x	3
y	18
z	2
space	166

Note: For fixed pitch fonts this value will be the same as the (A width + B width + C width) escapement of each character.

lMaxCharInc (LONG)
Maximum character increment.

The maximum character increment for the font in world coordinate units.

lEmInc (LONG)
Em increment.

The width of the Em square in world coordinate units. This corresponds to the point size of the font. When the horizontal device resolution equals the vertical device resolution this is equal to the em height.

lMaxBaselineExt (LONG)
Maximum baseline extent.

The maximum vertical space occupied by the font, in world coordinate units. This is the sum of *lMaxAscender* and *lMaxDescender* if both are positive. It is also the sum of *lInternalLeading* and *lEmHeight*.

One possible line spacing can be computed by adding *lMaxBaselineExt* to *lExternalLeading*. Such a line spacing, however, would be dependent on the glyph set included in the font. If a new version of the font should be made available, with new glyphs, then it is possible that this value will change because one of the new glyphs has gone above the previous *lMaxAscender* or below the previous *lMaxDescender*. More sophisticated applications will base line spacing on the point size (*lEmHeight*) of the font, which is an invariant of the font, multiplied by some factor (say 120%) plus any external leading.

This field may exceed *lEmHeight*.

sCharSlope (SHORT)
Character slope.

Defines the nominal slope for the characters of a font. The slope is defined in degrees increasing clockwise from the vertical. An *Italic* font is an example of a font with a nonzero slope.

Note: The units for this metric are degrees and minutes, encoded as shown in the following example:

```
180 degrees 59 minutes would be represented as :

|  < byte 1 >   |  < byte 2 >   |

| | <Minutes> |  <  Degrees  >  |

|0|1 1 1 0 1 1|0 1 0 1 1 0 1 0 0|

  |  59 min   |   180 degrees   |
```

sInlineDir (SHORT)
Inline direction.

The direction in which the characters in the font are designed for viewing, in degrees increasing clockwise from the horizontal (left-to-right). Characters are added to a line of text in the inline direction.

Note: The units for this metric are degrees and minutes, encoded as shown in sCharSlope on page A-55.

sCharRot (SHORT)
Character rotation.

The rotation of the character glyphs with respect to the baseline, the angle increasing counter clockwise. This is the angle assigned by the font designer.

Note: The units for this metric are degrees and minutes, encoded as shown in sCharSlope on page A-55.

usWeightClass (USHORT)
Weight class.

Indicates the visual weight (thickness of strokes) of the characters in the font:

Value	Description
1000	Ultra-light
2000	Extra-light
3000	Light
4000	Semi-light
5000	Medium (normal)
6000	Semi-bold
7000	Bold
8000	Extra-bold
9000	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:

Value	Description	% of normal width
1000	Ultra-condensed	50
2000	Extra-condensed	62.5
3000	Condensed	75
4000	Semi-condensed	87.5
5000	Medium (normal)	100
6000	Semi-expanded	112.5
7000	Expanded	125
8000	Extra-expanded	150
9000	Ultra-expanded	200

sXDeviceRes (SHORT)
x-device resolution.

For bit-map fonts this is the resolution in the X direction of the intended target device, measured in pels per inch.

For outline fonts this is the number of notional units in the X direction of the Em square, measured in notional units per Em. (Notional units are the units in which the outline is defined.

sYDeviceRes (SHORT)
y-device resolution.

For bit-map fonts this is the resolution in the Y direction of the intended target device, measured in pels per inch.

For outline fonts this is the number of notional units in the Y direction of the Em square, measured in notional units per Em. (Notional units are the units in which the outline is defined.

sFirstChar (SHORT)
First character.

The code point of the first character in the font.

sLastChar (SHORT)
Last character.

The code point of the last character in the font, expressed as an offset from *sFirstChar*.

All code points between the first and last character specified must be supported by the font.

sDefaultChar (SHORT)
Default character.

The code point that is used if a code point outside the range supported by the font is used, expressed as an offset from *sFirstChar*.

sBreakChar (SHORT)
Break character.

The code point that represents the "space" or "break" character for this font, expressed as an offset from *sFirstChar*. For example, if the first character is the space in code page 850, *sFirstChar* = 32, and *sBreakChar* = 0.

sNominalPointSize (SHORT)
Nominal point size.

For a bit-map font this field contains the height of the font.

For an outline font, this field contains the height the font designer had in mind for this font. For example some fonts are designed for text use in which case a value of 120 (12 point) would probably be placed in this field, whereas other fonts are designed for "display" use ("display" is typographer's terminology for larger sizes). This is not the only size the font can be used at.

Measured in decipoints (a decipoint is 1/720th of an inch).

sMinimumPointSize (SHORT)
Minimum point size.

For a bit-map font, this field is meaningless.

For an outline font, this field contains the minimum height the font designer had in mind for this font. Note that this is not a restriction on the size the font can be used at.

Measured in decipoints (a decipoint is 1/720th of an inch).

sMaximumPointSize (SHORT)
Maximum point size.

For a bit-map font, this field is meaningless.

For an outline font, this field contains the maximum height the font designer had in mind for this font. Note that this is not a restriction on the size the font can be used at.

Measured in decipoints (a decipoint is 1/720th of an inch).

usType (USHORT)
Type indicators.

Contains this information:

FM_TYPE_FIXED	Characters in the font have the same fixed width.
FM_TYPE_LICENSED	Licensed (protected) font.
FM_TYPE_KERNING	Font contains kerning information.
FM_TYPE_64K	Font is larger than 64KB (KB equals 1024 bytes) in size. If the following two bits are false, the font is for single-byte code pages. One of the bits may be set.
FM_TYPE_DBCS	Font is for double-byte code pages.
FM_TYPE_MBCS	Font is for mixed single/double-byte code pages.
FM_TYPE_FACETRUNC	Font *szFacename*[*FACESIZE*] has been truncated.
FM_TYPE_FAMTRUNC	Font *szFamilyname*[*FACESIZE*] has been truncated.
FM_TYPE_ATOMS	The System Atom table atom values in *FamilyNameAtom* and in *FaceNameAtom* are valid.

usDefn (USHORT)
Definition indicators.

Contains the following font definition data:

FM_DEFN_OUTLINE	Font is a vector (outline) font, otherwise it is a bit-map font.
FM_DEFN_GENERIC	Font is in a format that can be used by the GPI, otherwise it is a device font.

usSelection (USHORT)
Selection indicators.

Contains information about the font patterns in the physical font.

Note: The flags do not reflect simulations applied to the physical font.

FM_SEL_ITALIC	True indicates that this font is designed as an italic font.
FM_SEL_UNDERSCORE	TRUE indicates that this font is designed with underscores included in each character.
FM_SEL_NEGATIVE	TRUE indicates that this font is designed with the background and foreground reversed.
FM_SEL_OUTLINE	TRUE indicates that this font is designed with outline (hollow) characters.
FM_SEL_STRIKEOUT	TRUE indicates that this font is designed with an overstrike through each character.
FM_SEL_BOLD	TRUE indicates that this font is designed with bold characters.

usCapabilities (USHORT)
Capabilities.

This attribute applies only to device fonts.

FM_CAP_NOMIX	Characters may not be mixed with graphics.

QUALITY The most significant byte may contain the following numeric value:

0	Undefined
1	DP quality
2	DP draft
3	Near Letter Quality
4	Letter Quality

lSubscriptXSize (LONG)
Subscript x-size.

The recommended horizontal size for subscripts for this font in world coordinate units.

lSubscriptYSize (LONG)
Subscript y-size.

The recommended vertical size for subscripts for this font in world coordinate units.

lSubscriptXOffset (LONG)
Subscript x-offset.

The recommended baseline x-offset for subscripts for this font in world coordinate units.

lSubscriptYOffset (LONG)
Subscript y-offset.

The recommended baseline y-offset for subscripts for this font in world coordinate units.

Note: Positive numbers mean below the baseline.

lSuperscriptXSize (LONG)
Superscript x-size.

The recommended horizontal size for superscripts for this font in world coordinate units.

lSuperscriptYSize (LONG)
Superscript y-size.

The recommended vertical point size for superscripts for this font in world coordinate units.

lSuperscriptXOffset (LONG)
Superscript x-offset.

The recommended baseline x-offset for superscripts for this font in world coordinate units.

lSuperscriptYOffset (LONG)
Superscript y-offset.

The recommended baseline y-offset for superscripts for this font in world coordinate units.

lUnderscoreSize (LONG)
Underscore size.

The width (thickness) of the underscore stroke in world coordinate units This describes the actual underscore in the font if FM_SEL_UNDERSCORE is also set. Otherwise it describes what the engine will simulate if underscore is requested in GpiCreateLogFont.

lUnderscorePosition (LONG)
Underscore position.

The position of the underscore stroke from the baseline in world coordinate units. This describes the actual underscore in the font if FM_SEL_UNDERSCORE is also set. Otherwise it describes what the engine will simulate if underscore is requested in GpiCreateLogFont.

Note: Positive values mean below the baseline.

lStrikeoutSize (LONG)
Strikeout size.

The width of the strikeout stroke in world coordinate units. This describes the actual underscore in the font if FM_SEL_STRIKEOUT is also set. Otherwise it describes what the engine will simulate if overstrike is requested in GpiCreateLogFont.

lStrikeoutPosition (LONG)
Strikeout position.

The position of the strikeout stroke relative to the baseline in world coordinate units. This describes the actual underscore in the font if FM_SEL_STRIKEOUT is also set. Otherwise it describes what the engine will simulate if overstrike is requested in GpiCreateLogFont.

sKerningPairs (SHORT)
Kerning pairs.

The number of kerning pairs in the kerning pair table.

sFamilyClass (SHORT)
Font family design classification.

This value contains a font class and its subclass.

lMatch (LONG)
Matched font identity.

This uniquely identifies the font for a given device/device driver combination. A positive match number signifies that the font is a generic (engine) font while a negative number indicates a device font (a native or downloadable font). This value should not be used to identify a font across system boundaries.

FamilyNameAtom (ATOM)
Font family name atom.

This value contains the atom identifier for the font family name in the System Atom Table.

FaceNameAtom (ATOM)
Font facename atom.

This value contains the atom identifier for the font face name in the System Atom Table.

panPanose (PANOSE)
Panose font descriptor.

This is the Panose descriptor identifying the visual characteristics of the font.

FRAMECDATA

Frame-control data structure.

```
typedef struct _FRAMECDATA {
USHORT     cb;
ULONG      flCreateFlags;
HMODULE    hmodResources;
USHORT     idResources;
 } FRAMECDATA;
```

cb (USHORT)
Length.

flCreateFlags (ULONG)
Frame-creation flags.

hmodResources (HMODULE)
Identifier of required resource.

This is supplied in an environment-dependent manner.

idResources (USHORT)
Resource identifier.

FTIME

Time data structure for file-system functions.

```
typedef struct _FTIME {
USHORT    ustwosecs;
USHORT    usminutes;
USHORT    ushours;
 } FTIME;
```

ustwosecs (USHORT)

A binary number of two-second increments.

usminutes (USHORT)

A binary number of minutes.

ushours (USHORT)

A binary number of hours.

GRADIENTL

Direction-vector structure.

```
typedef struct _GRADIENTL {
LONG    x;
LONG    y;
 } GRADIENTL;
```

x (LONG)
x-component of direction.

y (LONG)
y-component of direction.

HAB

Anchor-block handle.

```
typedef LHANDLE HAB;
```

HACCEL

Accelerator-table handle.

```
typedef LHANDLE HACCEL;
```

HAPP

Handle of an application.

```
typedef LHANDLE HAPP;
```

HATOMTBL

Atom-table handle.

```
typedef LHANDLE HATOMTBL;
```

HBITMAP

Bit-map handle.

```
typedef LHANDLE HBITMAP;
```

HCINFO

Hardcopy-capabilities structure.

```
typedef struct _HCINFO {
CHAR    szFormname[32];
LONG    cx;
LONG    cy;
LONG    xLeftClip;
LONG    yBottomClip;
LONG    xRightClip;
LONG    yTopClip;
LONG    xPels;
LONG    yPels;
LONG    flAttributes;
 } HCINFO;
```

szFormname[32] (CHAR)
Form name.

cx (LONG)
Width (left-to-right) in millimeters.

cy (LONG)
Height (top-to-bottom) in millimeters.

xLeftClip (LONG)
Left clip limit in millimeters.

yBottomClip (LONG)
Bottom clip limit in millimeters.

xRightClip (LONG)
Right clip limit in millimeters.

yTopClip (LONG)
Top clip limit in millimeters.

xPels (LONG)
Number of pels between left and right clip limits.

yPels (LONG)
Number of pels between bottom and top clip limits.

flAttributes (LONG)
Attributes of the form identifier.

HCAPS_CURRENT	Currently installed form.
HCAPS_SELECTABLE	Form is available from an alternate form source, without operator intervention.

The value returned is the sum of the applicable values. The bits in the field that are affected by each piece of information are separate.

HDC

Device-context handle.

```
typedef LHANDLE HDC;
```

HDDF

Dynamic data formatting handle.

```
typedef LHANDLE HDDF;
```

HELPINIT

Help manager initialization structure.

```
typedef struct _HELPINIT {
ULONG          cb;
ULONG          ulReturnCode;
PSZ            pszTutorialName;
PHELPTABLE     phtHelpTable;
HMODULE        hmodHelpTableModule;
HMODULE        hmodAccelActionBarModule;
ULONG          idAccelTable;
ULONG          idActionBar;
PSZ            pszHelpWindowTitle;
ULONG          fShowPanelId;
PSZ            pszHelpLibraryName;
 } HELPINIT;
```

cb (ULONG)
Count of bytes of the initialization structure.

ulReturnCode (ULONG)
Value returned by the help manager from initialization.

0 Initialization was successful.

pszTutorialName (PSZ)
Indicates to the help manager that the application has a tutorial program.

NULL	The application either does not have a tutorial program, or the tutorial name is specified in each help panel definition.
Other	Default tutorial name.

phtHelpTable (PHELPTABLE)
Help table.

The help table or the identity of the help table. If this is the identity of the

help table in a resource file, the low-order word contains the identity of the table and the high-order word must be X'FFFF'.

The help table associates each application window with its help subtable and the identity of its extended help panel.

hmodHelpTableModule (HMODULE)
Resource file identity.

If the *phtHelpTable* contains the identity of the help table, this field identifies the module handle returned by the DosLoadModule call by which the application loaded the resource file.

NULL The resource file containing the help table was appended to the application's .EXE file.
Other Resource file identity.

hmodAccelActionBarModule (HMODULE)
Handle of the containing DLL.

The handle of the DLL which contains the accelerator table and action bar template to be used by the help manager.

NULL Use the default action bar and accelerator table defined by the help manager.
Other Handle of the DLL.

idAccelTable (ULONG)
Identity of the accelerator table.

The accelerator table resides in the DLL provided in the *hmodAccelActionBarModule* field.

NULL Use the default accelerator table.
Other Identity of the accelerator table.

idActionBar (ULONG)
Identity of the action bar template used by the e help manager.

The action bar template resides in the DLL provided in the *hmodAccelActionBarModule* field.

NULL Use the default action bar.
Other Identity of the action bar.

pszHelpWindowTitle (PSZ)
Window title for the main help window of this help instance.

fShowPanelId (ULONG)
Show panel identity indicator.

The constants corresponding to the panel identity flags are in the PMHELP.H include file.

CMIC_SHOW_PANEL_ID Show the panel identity on a help panel.
CMIC_HIDE_PANEL_ID Do not show the panel identity on a help panel.

pszHelpLibraryName (PSZ)
Help panel library names.

The names of the help panel libraries that the help manager searches on each help request. The names must be separated by a blank.

The help manager looks for the libraries in the path set by the HELP environment variable. If the library is not found, the help manager will look for the libraries in the current directory.

HELPTABLE Help table.

This is a collection of help table entries, each of which has the structure defined below, the last entry of the collection being a NULL structure.

```
typedef struct _HELPTABLE {
USHORT    idAppWindow;
PSHORT    phstHelpSubTable;
USHORT    idExtPanel;
 } HELPTABLE;
```

idAppWindow (USHORT)
Application window identity.

phstHelpSubTable (PSHORT)
Help subtable for this application window.

idExtPanel (USHORT)
Identity of the extended help panel for the application window.

HENUM

Window-enumeration handle.

```
typedef LHANDLE HENUM;
```

HEV

32-bit value used as an event handle.

```
typedef ULONG *HEV;
```

HFILE

Resource handle.

```
typedef LHANDLE HFILE;
```

HFIND

Handle associated to a wpclsFindObjectFirst request.

```
typedef LHANDLE HFIND;
```

HINI

Initialization-file handle.

```
typedef LHANDLE HINI;
```

HLIB

Library handle.

```
typedef LHANDLE HLIB;
```

HMF

Metafile handle.

```
typedef LHANDLE HMF;
```

HMODULE

Module handle.

```
typedef LHANDLE HMODULE;
```

HMQ

Message-queue handle.

```
typedef LHANDLE HMQ;
```

HMTX

32-bit value used as a mutex-semaphore handle.

```
typedef ULONG *HMTX;
```

HMUX

32-bit value used as a muxwait semaphore handle.

```
typedef ULONG *HMUX;
```

HOBJECT

Workplace object handle.

```
typedef LHANDLE HOBJECT;
```

HPAL

Palette handle.

```
typedef LHANDLE HPAL;
```

HPOINTER

Pointer handle.

```
typedef LHANDLE HPOINTER;
```

HPROC

Processor handle.

```
typedef LHANDLE HPROC;
```

HPROGARRAY

Array of program handles.

```
typedef struct _HPROGARRAY {
HPROGRAM    ahprog[1];
 } HPROGARRAY;
```

ahprog[1] (HPROGRAM)
Program handle array.

HPROGRAM Program handle.

```
typedef LHANDLE HPROGRAM;
```

HPS Presentation-space handle.

```
typedef LHANDLE HPS;
```

HRGN Region handle.

```
typedef LHANDLE HRGN;
```

HSEM Semaphore handle.

```
typedef VOID *HSEM;
```

HSPL Spooler handle.

```
typedef LHANDLE HSPL;
```

HSTR String handle.

```
typedef LHANDLE HSTR;
```

HSVWP Frame window-repositioning process handle.

```
typedef LHANDLE HSVWP;
```

HSWITCH Switch-list entry handle.

```
typedef LHANDLE HSWITCH;
```

HWND Window handle.

```
typedef LHANDLE HWND;
```

ICONINFO ICONINFO data structure.

```
typedef struct _ICONINFO {
ULONG      ulcb;
ULONG      fFormat;
PSZ        pszFileName;
HMODULE    hmod;
ULONG      ulresid;
ULONG      cbIconData;
PVOID      pIconData;
 } ICONINFO;
```

ulcb (ULONG)

Length of ICONINFO structure.

fFormat (ULONG)

Indicates from where the icon resides.

ICON_FILE	Icon file supplied.
ICON_RESOURCE	Icon resource supplied.
ICON_DATA	Icon data supplied.
ICON_CLEAR	Go back to default icon.

pszFileName (PSZ)

Name of file containing icon data. This value is ignored if fFormat is not equal to to ICON_FILE.

hmod (HMODULE)

Module containing the icon resource. This value is ignored if fFormat is not equal to to ICON_RESOURCE.

ulresid (ULONG)

Identity of icon resource. This value is ignored if fFormat is not equal to to ICON_RESOURCE.

cbIconData (ULONG)

Length of icon data in bytes. This value is ignored if fFormat is not equal to to ICON_DATA.

pIconData (PVOID)

Pointer to buffer containing icon data. This value is ignored if fFormat is not equal to to ICON_DATA.

IconPos

Icon position structure.

```
typedef ICONPOS *IconPos;
```

IMAGEBUNDLE

Image-attributes bundle structure.

```
typedef struct _IMAGEBUNDLE {
LONG      lColor;
LONG      lBackColor;
USHORT    usMixMode;
USHORT    usBackMixMode;
 } IMAGEBUNDLE;
```

lColor (LONG)
Image foreground color.

lBackColor (LONG)
Image background color.

usMixMode (USHORT)
Image foreground-mix mode.

usBackMixMode (USHORT)
Image background-mix mode.

IPT

Insertion point for multi-line entry field.

```
typedef LONG IPT;
```

KERNINGPAIRS

Kerning-pair records structure.

```
typedef struct _KERNINGPAIRS {
SHORT    sFirstChar;
SHORT    sSecondChar;
LONG     lKerningAmount;
 } KERNINGPAIRS;
```

sFirstChar (SHORT)
First character of pair.

sSecondChar (SHORT)
Second character of pair.

lKerningAmount (LONG)
Amount of kerning for this pair.

LHANDLE

The handle of a resource.

```
typedef ULONG *LHANDLE;
```

LINEBUNDLE

Line-attributes bundle structure.

```
typedef struct _LINEBUNDLE {
LONG      lColor;
LONG      lReserved;
ULONG     ulMixMode;
USHORT    usReserved;
FIXED     fxWidth;
LONG      lGeomWidth;
ULONG     ulType;
ULONG     ulEnd;
ULONG     ulJoin;
 } LINEBUNDLE;
```

lColor (LONG)
Line foreground color.

lReserved (LONG)
Reserved.

ulMixMode (ULONG)
Line foreground-mix mode.

usReserved (USHORT)
Reserved.

fxWidth (FIXED)
Line width.

lGeomWidth (LONG)
Geometric line width.

ulType (ULONG)
Line type.

ulEnd (ULONG)
Line end.

ulJoin (ULONG)
Line join.

LONG

Signed integer in the range –2 147 483 648 through 2 147 483 647.

Note: Where this data type represents a graphic coordinate in world or model space, its value is restricted to –134 217 728 through 134 217 727.

A graphic coordinate in device or screen coordinates is restricted to –32 768 through 32 767.

The value of a graphic coordinate may be further restricted by any transforms currently in force, including the positioning of the origin of the window on the screen. In particular, coordinates in world or model space must not generate coordinate values after transformation (that is, in device or screen space) outside the range –32 768 through 32 767.

```
#define LONG long
```

*M_WPFileSystem **

Pointer to a WPFileSystem class object.

```
typedef M_WPFileSystem *M_WPFileSystem *;
```

*M_WPFolder **

Pointer to a WPFolder class object.

```
typedef M_WPFolder *M_WPFolder *;
```

*M_WPObject **

Pointer to a WPObject class object.

```
typedef M_WPObject *M_WPObject *;
```

*M_WPPalette **

Pointer to a WPPalette class object.

```
typedef M_WPPalette *M_WPPalette *;
```

MARKERBUNDLE

Marker-attributes bundle structure.

```
typedef struct _MARKERBUNDLE {
LONG      lColor;
LONG      lBackColor;
USHORT    usMixMode;
USHORT    usBackMixMode;
USHORT    usSet;
USHORT    usSymbol;
SIZEF     sizfxCell;
 } MARKERBUNDLE;
```

lColor (LONG)
Marker foreground color.

lBackColor (LONG)
Marker background color.

usMixMode (USHORT)
Marker foreground-mix mode.

usBackMixMode (USHORT)
Marker background-mix mode.

usSet (USHORT)
Marker set.

usSymbol (USHORT)
Marker symbol.

sizfxCell (SIZEF)
Marker cell.

MATRIXLF

Matrix-elements structure.

```
typedef struct _MATRIXLF {
FIXED    fxM11;
FIXED    fxM12;
LONG     lM13;
FIXED    fxM21;
FIXED    fxM22;
LONG     lM23;
LONG     lM31;
LONG     lM32;
LONG     lM33;
 } MATRIXLF;
```

fxM11 (FIXED)
First element of first row.

fxM12 (FIXED)
Second element of first row.

lM13 (LONG)
Third element of first row.

fxM21 (FIXED)
First element of second row.

fxM22 (FIXED)
Second element of second row.

lM23 (LONG)
Third element of second row.

lM31 (LONG)
First element of third row.

lM32 (LONG)
Second element of third row.

lM33 (LONG)
Third element of third row.

MEMORYITEM

USAGE_MEMORY structure.

```
typedef MEMORYITEM FAR *MEMORYITEM;
```

MENUITEM

Menu item.

```
typedef struct _MENUITEM {
LONG     iPosition;
ULONG    afStyle;
ULONG    afAttribute;
ULONG    id;
HWND     hwndSubMenu;
ULONG    hItem;
 } MENUITEM;
```

iPosition (LONG)
Position.

afStyle (ULONG)
Style.

afAttribute (ULONG)
Attribute.

id (ULONG)
Identity.

hwndSubMenu (HWND)
Submenu.

hItem (ULONG)
Item.

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.

```
typedef struct _MINIRECORDCORE {
ULONG              cb;
ULONG              flRecordAttr;
POINTL             ptlIcon;
PMINIRECORDCORE    pNextRecord;
PSZ                pszIcon;
HPOINTER           hptrIcon;
 } MINIRECORDCORE;
```

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.

pNextRecord (PMINIRECORDCORE)
Pointer.

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.

MLECTLDATA

Multiline entry-field (MLE) control data structure.

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

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 pels.

cyFormat (LONG)
Formatting-rectangle height in pels.

The *cxFormat* and *cyFormat* parameters identify the dimensions in pels 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.

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

afMargins (USHORT)

This gives the 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)

The message identity of the original mouse event.

iptNear (IPT)

The insertion point nearest to the margin event.

MLEOVERFLOW

Overflow error structure for multiline entry field.

```
typedef struct _MLEOVERFLOW {
ULONG    ulErrInd;
LONG     lBytesOver;
PIX      pixHorzOver;
PIX      pixVertOver;
 } MLEOVERFLOW;
```

ulErrInd (ULONG)
One or more EFR_* flags.

lBytesOver (LONG)
Number of bytes over the limit.

pixHorzOver (PIX)
Number of pels over the horizontal limit.

pixVertOver (PIX)
Number of pels over the vertical limit.

MLE_SEARCHDATA

Search structure for multiline entry field.

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

cb (USHORT)
Size of MLE_SEARCHDATA 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 (SHORT)
Length of string found at *iptStart*.

MPARAM

4-byte message-dependent parameter structure.

Certain elements of information, placed into the parameters of a message, have data types that do not use all 4 bytes of this data type. The rules governing these cases are:

BOOL	The value is contained in the low word and the high word is 0.
SHORT	The value is contained in the low word and its sign is extended into the high word.
USHORT	The value is contained in the low word and the high word is 0.
NULL	The entire 4 bytes are 0.

```
typedef VOID FAR *MPARAM;
```

MQINFO

Message-queue information structure.

```
typedef struct _MQINFO {
ULONG    ulb;
PID      pid;
TID      tid;
ULONG    ulmsgs;
PVOID    pReserved;
 } MQINFO;
```

ulb (ULONG)
Length of structure.

pid (PID)
Process identity.

tid (TID)
Thread identity.

ulmsgs (ULONG)
Message count.

pReserved (PVOID)
Reserved.

MRESULT

4-byte message-dependent reply parameter structure.

Certain elements of information, placed into the parameters of a message, have data types that do not use all 4 bytes of this data type. The rules governing these cases are:

BOOL	The value is contained in the low word and the high word is 0.
SHORT	The value is contained in the low word and its sign is extended into the high word.
USHORT	The value is contained in the low word and the high word is 0.
NULL	The entire 4 bytes are 0.

```
typedef VOID FAR *MRESULT;
```

MTI

Menu template item.

```
typedef struct _MTI {
USHORT    afStyle;
USHORT    afAttrs;
USHORT    idItem;
CHAR      c[2];
 } MTI;
```

afStyle (USHORT)
Style.

afAttrs (USHORT)
Attributes.

idItem (USHORT)
Item identity.

c[2] (CHAR)
Item data.

The format and length of this depend upon the value of *afStyle*.

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" on page 24-19 for information about that notification code.

```
typedef struct _NOTIFYDELTA {
HWND     hwndCnr;
ULONG    fDelta;
 } NOTIFYDELTA;
```

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" on page 24-15 for information about that notification code.

```
typedef struct _NOTIFYRECORDEMPHASIS {
HWND           hwndCnr;
PRECORDCORE    pRecord;
ULONG          fEmphasisMask;
 } NOTIFYRECORDEMPHASIS;
```

hwndCnr (HWND)
Container control handle.

pRecord (PRECORDCORE)
Pointer.

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" on page 24-16 for information about that notification code.

```
typedef struct _NOTIFYRECORDENTER {
HWND           hwndCnr;
PRECORDCORE    pRecord;
ULONG          fKey;
 } NOTIFYRECORDENTER;
```

hwndCnr (HWND)
Container control handle.

pRecord (PRECORDCORE)
Pointer.

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.

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.

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" on page 24-21 for information about that notification code.

```
typedef struct _NOTIFYSCROLL {
HWND     hwndCnr;
LONG     lScrollInc;
ULONG    fScroll;
 } NOTIFYSCROLL;
```

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.

OBJCLASS Object class structure.

```
typedef struct _OBJCLASS {
STRUCT    _OBJCLASS;
PSZ       pszClassName;
PSZ       pszModName;
 } OBJCLASS;
```

_OBJCLASS (STRUCT)

Next object class structure.

pszClassName (PSZ)

Class name.

pszModName (PSZ)

Module name.

OBJDATA Object data structure. Class specific information is contained in this structure.

```
typedef struct _OBJDATA {
WPSRCLASSBLOCK*    CurrentClass;
WPSRCLASSBLOCK*    First;
UCHAR              ucNextData;
USHORT             usLength;
 } OBJDATA;
```

CurrentClass (WPSRCLASSBLOCK*)

Pointer to current save or restore class block.

First (WPSRCLASSBLOCK*)

Pointer to first save or restore class block.

ucNextData (UCHAR)

Pointer to next block of data.

usLength (USHORT)

Length.

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 24-35 for information about that message.

```
typedef struct _OWNERBACKGROUND {
HWND     hwnd;
HPS      hps;
RECTL    rclBackground;
LONG     idWindow;
 } OWNERBACKGROUND;
```

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.

```
typedef struct _OWNERITEM {
HWND      hwnd;
HPS       hps;
ULONG     ulState;
ULONG     ulAttribute;
ULONG     ulStateOld;
USHORT    fsAttributeOld;
RECTL     rclItem;
LONG      idItem;
ULONG     hItem;
 } OWNERITEM;
```

hwnd (HWND)
Window handle.

hps (HPS)
Presentation-space handle.

ulState (ULONG)
State.

ulAttribute (ULONG)
Attribute.

ulStateOld (ULONG)
Old state.

fsAttributeOld (USHORT)
Old attribute.

rclItem (RECTL)
Item rectangle.

idItem (LONG)
Item identity.

hItem (ULONG)
Item.

PACCEL Pointer to ACCEL.

```
typedef ACCEL *PACCEL;
```

PACCELTABLE Pointer to ACCELTABLE.

```
typedef ACCELTABLE *PACCELTABLE;
```

PAGEINFO Settings page information structure.

```
typedef struct _PAGEINFO {
ULONG      ulcb;
HWND       hwndPage;
PFNWP      ppfnwp;
ULONG      ulresid;
PVOID      pCreateParams;
USHORT     usdlgid;
USHORT     usPageStyleFlags;
USHORT     usPageInsertFlags;
USHORT     usReserved;
PSZ        pszName;
USHORT     idDefaultHelpPanel;
USHORT     usReserved2;
PSZ        pszHelpLibraryName;
PUSHORT    pHelpSubtable;
HMODULE    hmodHelpSubtable;
ULONG      ulPageInsertId;
 } PAGEINFO;
```

ulcb (ULONG)

Length of PAGEINFO structure.

hwndPage (HWND)

Handle of page.

ppfnwp (PFNWP)

Window procedure.

ulresid (ULONG)

Resource identity.

pCreateParams (PVOID)

Pointer to creation parameters.

usdlgid (USHORT)

Dialog identity.

usPageStyleFlags (USHORT)

Notebook control page style flags.

usPageInsertFlags (USHORT)

Notebook control page insertion flags.

usReserved (USHORT)

Reserved value must be zero.

pszName (PSZ)

Pointer to a string containing page name.

idDefaultHelpPanel (USHORT)

Identity of default help panel.

usReserved2 (USHORT)

Reserved value must be zero.

pszHelpLibraryName (PSZ)

Pointer to name of help file.

pHelpSubtable (PUSHORT)

Pointer to help subtable.

hmodHelpSubtable (HMODULE)

Module handle for help subtable.

ulPageInsertId (ULONG)

Notebook control page identity.

PAGESELECTNOTIFY Structure that contains information about the application page being selected.

```
typedef struct _PAGESELECTNOTIFY {
HWND     hwndBook;
ULONG    ulPageIdCur;
ULONG    ulPageIdNew;
 } PAGESELECTNOTIFY;
```

hwndBook (HWND)
Notebook window handle.

ulPageIdCur (ULONG)
Current top page identifier.

ulPageIdNew (ULONG)
New top page identifier.

PALINFO Class specific palette information data.

```
typedef struct _PALINFO {
ULONG    ulxCellCount;
ULONG    ulyCellCount;
ULONG    ulxCursor;
ULONG    ulyCursor;
ULONG    ulxCellWidth;
ULONG    ulyCellHeight;
ULONG    ulxGap;
ULONG    ulyGap;
 } PALINFO;
```

ulxCellCount (ULONG)
Number of columns of palinfos.

ulyCellCount (ULONG)
Number of rows of palinfos.

ulxCursor (ULONG)
Cursor location (readonly).

ulyCursor (ULONG)
Cursor location (readonly).

ulxCellWidth (ULONG)
Width of each palinfo.

ulyCellHeight (ULONG)
Height of each palinfo.

ulxGap (ULONG)
X separation of palinfos.

ulyGap (ULONG)
Y separation of palinfos.

PANOSE The Panose field in the font metrics will allow for quantitative descriptions of the visual properties of font faces. The PANOSE definition contains ten digits, each of which currently describes up to sixteen variations.

```
typedef struct _PANOSE {
BYTE    bbFamilyType;
BYTE    bbSerifStyle;
BYTE    bbWeight;
BYTE    bbProportion;
BYTE    bbContrast;
BYTE    bbStrokeVariation;
BYTE    bbArmStyle;
BYTE    bbLetterform;
BYTE    bbMidline;
BYTE    bbXHeight;
BYTE    ababReserved[FACESIZE];
 } PANOSE;
```

bbFamilyType (BYTE)
Family kind.

0 Any

1 No Fit

2 Text and Display

3 Script

4 Decorative

5 Pictorial

bbSerifStyle (BYTE)
Serif style.

0 Any

1 No Fit

2 Cove

3 Obtuse Cove

4 Square Cove

5 Obtuse Square Cove

6 Square

7 Thin

8 Bone

9 Exaggerated

10 Triangle

11 Normal Sans

12 Obtuse Sans

13 Perp Sans

14 Flared

15 Rounded

bbWeight (BYTE)
Weight.

0 Any

1 No Fit

2 Very Light

3 Light

4 Thin

5 Book

6 Medium

7 Demi

8 Bold

9 Heavy

10 Black

11 Nord

bbProportion (BYTE)
Proportion.

0 Any

1 No Fit

2 Old Style

3 Modern

4 Even Width

5 Expanded

6 Condensed

7 Very Expanded

8 Very Condensed

9 Monospaced

bbContrast (BYTE)
Contrast.

0 Any

1 No Fit

2 None

3 Very Low

4 Low

5 Medium Low

6 Medium

7 Medium High

8 High

9 Very High

bbStrokeVariation (BYTE)
Stroke Variation.

0 Any

1 No Fit

2 Gradual/Diagonal

3 Gradual/Transitional

4 Gradual/Vertical

5 Gradual/Horizontal

6 Rapid/Vertical

7 Rapid/Horizontal

8 Instant/Vertical

bbArmStyle (BYTE)
Arm Style.

0 Any

1 No Fit

2 Straight Arms/Horizontal

3 Straight Arms/Wedge

4 Straight Arms/Vertical

5 Straight Arms/Single Serif

6 Straight Arms/Double Serif

7 Non-Straight Arms/Horizontal

8 Non-Straight Arms/Wedge

9 Non-Straight Arms/Vertical

10 Non-Straight Arms/Single Serif

11 Non-Straight Arms/Double Serif

bbLetterform (BYTE)
Letterform.

0 Any

1 No Fit

2 Normal/Contact

3 ONormal/Weighted

4 ONormal/Boxed

5 ONormal/Flattened

6 ONormal/Rounded

7 ONormal/Off Center

8 ONormal/Square

9 Oblique/Contact

10 Oblique/Weighted

11 Oblique/Boxed

12 Oblique/Flattened

13 Oblique/Rounded

14 Oblique/Off Center

15 Oblique/Square

bbMidline (BYTE)
Midline.

0 Any

1 No Fit

2 Standard/Trimmed

3 Standard/Pointed

4 Standard/Serifed

5 High/Trimmed

6 High/Pointed

7 High/Serifed

8 Constant/Trimmed

9 Constant/Pointed

10 Constant/Serifed

11 Low/Trimmed

12 Low/Pointed

13 Low/Serifed

bbXHeight (BYTE)
X-Height.

0 Any

1 No Fit

2 Constant/Small

3 Constant/Standard

4 Constant/Large

5 Ducking/Small

6 Ducking/Standard

7 Ducking/Large

ababReserved[FACESIZE] (BYTE)
Reserved.

PAPSZ

Pointer to an array of pointers to null-terminated strings.

```
typedef char *PAPSZ;
```

PARAM

Presentation parameter attribute definition.

```
typedef struct _PARAM {
ULONG    id;
ULONG    cb;
BYTE     abab[1];
 } PARAM;
```

id (ULONG)
Attribute type identity.

These identities are in the range of X'00000000' to X'FFFFFFFF'. The window manager uses values of this parameter in the range X'00000000' to PP_USER, therefore an application should not define private presentation parameter attribute identities in this range. An application should use the WinAddAtom call to guarantee obtaining a unique 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.

cb (ULONG)
Byte count of the *abab[1]* parameter.

abab[1] (BYTE)
Attribute value.

The format of a value depends on the attribute type identity as follows:

PP_FOREGROUNDCOLOR	Foreground color value of data type RGB.
PP_BACKGROUNDCOLOR	Background color value of data type RGB.
PP_FOREGROUNDCOLORINDEX	Foreground color index value of data type LONG.
PP_BACKGROUNDCOLORINDEX	Background color index value of data type LONG.

PP_HILITEFOREGROUNDCOLOR	Highlighted foreground color value of data type RGB.
PP_HILITEBACKGROUNDCOLOR	Highlighted background color value of data type RGB.
PP_HILITEFOREGROUNDCOLORINDEX	Highlighted foreground color index value of data type LONG.
PP_HILITEBACKGROUNDCOLORINDEX	Highlighted background color index value of data type LONG.
PP_DISABLEDFOREGROUNDCOLOR	Disabled foreground color value of data type RGB.
PP_DISABLEDBACKGROUNDCOLOR	Disabled background color value of data type RGB.
PP_DISABLEDFOREGROUNDCOLORINDEX	Disabled foreground color index value of data type LONG.
PP_DISABLEDBACKGROUNDCOLORINDEX	Disabled background color index value of data type LONG.
PP_BORDERCOLOR	Border color value of data type RGB.
PP_BORDERCOLORINDEX	Border color index value of data type LONG.
PP_FONTNAMESIZE	Font name and size value of data type PSZ. The string is in two parts, separated by a period. The first part is the font point size and the second part is the font facename, for example, "12.Helv".

PARCPARAMS Pointer to ARCPARAMS.

```
typedef ARCPARAMS *PARCPARAMS;
```

PAREABUNDLE Pointer to AREABUNDLE.

```
typedef AREABUNDLE *PAREABUNDLE;
```

PBANDRECT Pointer to BANDRECT.

```
typedef BANDRECT *PBANDRECT;
```

PBITMAPINFO Pointer to BITMAPINFO.

```
typedef BITMAPINFO *PBITMAPINFO;
```

PBITMAPINFOHEADER Pointer to BITMAPINFOHEADER.

```
typedef BITMAPINFOHEADER *PBITMAPINFOHEADER;
```

PBITMAPINFOHEADER2 Pointer to BITMAPINFOHEADER2.

```
typedef BITMAPINFOHEADER2 *PBITMAPINFOHEADER2;
```

PBITMAPINFO2 Pointer to BITMAPINFO2.

```
typedef BITMAPINFO2 *PBITMAPINFO2;
```

PBOOKTEXT Pointer to a BOOKTEXT data structure.

```
typedef BOOKTEXT *PBOOKTEXT;
```

PBOOL Pointer to BOOL.

```
typedef BOOL *PBOOL;
```

PBUFFER Pointer to PBYTE.

```
typedef BUFFER *PBUFFER;
```

PBUNDLE Points to a bundle data area.

```
typedef PVOID PBUNDLE;
```

PBYTE Pointer to a data area.

```
typedef BYTE *PBYTE;
```

PCATCHBUF Pointer to CATCHBUF.

```
typedef CATCHBUF *PCATCHBUF;
```

PCDATE Pointer to CDATE.

```
typedef CDATE *PCDATE;
```

PCELL Pointer to CELL.

```
typedef CELL *PCELL;
```

PCH Pointer to a character string.

```
typedef char *PCH;
```

PCHAR Pointer to CHAR.

```
typedef CHAR *PCHAR;
```

PCHARBUNDLE Pointer to CHARBUNDLE.

```
typedef CHARBUNDLE *PCHARBUNDLE;
```

PCLASSDETAILS Pointer to an CLASSDETAILS data structure.

```
typedef CLASSDETAILS *PCLASSDETAILS;
```

PCLASSFIELDINFO Pointer to an ClassFieldInfo data structure.

```
typedef CLASSFIELDINFO *PCLASSFIELDINFO;
```

PCLASSINFO Pointer to CLASSINFO.

```
typedef CLASSINFO *PCLASSINFO;
```

PCNRDRAGINFO Pointer to a CNRDRAGINFO data structure.

```
typedef CNRDRAGINFO *PCNRDRAGINFO;
```

PCNRDRAGINIT Pointer to a CNRDRAGINIT data structure.

```
typedef CNRDRAGINIT *PCNRDRAGINIT;
```

PCNRDRAWITEMINFO Pointer to a CNRDRAWITEMINFO data structure.

```
typedef CNRDRAWITEMINFO *PCNRDRAWITEMINFO;
```

PCNREDITDATA Pointer to a CNREDITDATA data structure.

```
typedef CNREDITDATA *PCNREDITDATA;
```

PCNRINFO Pointer to a CNRINFO data structure.

```
typedef CNRINFO *PCNRINFO;
```

PCOLOR Pointer to COLOR.

```
typedef COLOR *PCOLOR;
```

PCONVCONTEXT Pointer to a CONVCONTEXT data structure.

```
typedef CONVCONTEXT *PCONVCONTEXT;
```

PCPTEXT Pointer to CPTEXT.

```
typedef CPTEXT *PCPTEXT;
```

PCREATEPARAMS Pointer to PVOID.

```
typedef CREATEPARAMS FAR *PCREATEPARAMS;
```

PCREATESTRUCT Pointer to a CREATESTRUCT data structure.

```
typedef CREATESTRUCT *PCREATESTRUCT;
```

PCTIME Pointer to CTIME.

```
typedef CTIME *PCTIME;
```

PCURSORINFO Pointer to CURSORINFO.

```
typedef CURSORINFO *PCURSORINFO;
```

PDDEINIT Pointer to a DDEINIT data structure.

```
typedef DDEINIT *PDDEINIT;
```

PDDESTRUCT Pointer to DDESTRUCT.

```
typedef DDESTRUCT *PDDESTRUCT;
```

PDELETENOTIFY Pointer to a DELETENOTIFY data structure.

```
typedef DELETENOTIFY *PDELETENOTIFY;
```

PDESKTOP Pointer to a DESKTOP image data structure.

```
typedef DESKTOP *PDESKTOP;
```

PDEVOPENDATA Open device-data array.

This data type points to data whose format is described by the DEVOPENSTRUC data type.

```
typedef PSZ *PDEVOPENDATA;
```

PDEVOPENSTRUC Pointer to DEVOPENSTRUC.

```
typedef DEVOPENSTRUC *PDEVOPENSTRUC;
```

PDLGTEMPLATE Pointer to DLGTEMPLATE.

```
typedef DLGTEMPLATE *PDLGTEMPLATE;
```

PDLGTITEM Pointer to DLGTITEM.

```
typedef DLGTITEM *PDLGTITEM;
```

PDRAGIMAGE Pointer to a DRAGIMAGE data structure.

```
typedef DRAGIMAGE *PDRAGIMAGE;
```

PDRAGINFO Pointer to a DRAGINFO data structure.

```
typedef DRAGINFO *PDRAGINFO;
```

PDRAGITEM Pointer to a DRAGITEM data structure.

```
typedef DRAGITEM *PDRAGITEM;
```

PDRAGTRANSFER Pointer to a DRAGTRANSFER data structure.

```
typedef DRAGTRANSFER *PDRAGTRANSFER;
```

PDRIVDATA Driver-data structure.

This data type points to data whose format is described by the DRIVDATA data type.

```
typedef DRIVDATA *PDRIVDATA;
```

PDRIVPROPS Driver property structure.

This data type points to data whose format is described by the DRIVPROPS data type.

```
typedef DRIVPROPS *PDRIVPROPS;
```

PENTRYFDATA Pointer to ENTRYFDATA.

```
typedef ENTRYFDATA *PENTRYFDATA;
```

PERRINFO Pointer to ERRINFO.

```
typedef ERRINFO *PERRINFO;
```

PERRORID Pointer to ERRORID.

```
typedef ERRORID *PERRORID;
```

PESCMODE Pointer to ESCSETMODE.

```
typedef ESCMODE *PESCMODE;
```

PFACENAMEDESC Pointer to FACENAMEDESC.

```
typedef FACENAMEDESC *PFACENAMEDESC;
```

PFATTRS Pointer to FATTRS.

```
typedef FATTRS *PFATTRS;
```

PFFDESCS Pointer to a font file descriptor.

```
typedef FFDESCS *PFFDESCS;
```

PFIELDINFO Pointer to a FIELDINFO data structure.

```
typedef FIELDINFO *PFIELDINFO;
```

PFIELDINFOINSERT Pointer to a FIELDINFOINSERT data structure.

```
typedef FIELDINFOINSERT *PFIELDINFOINSERT;
```

PFILEDLG Pointer to a FILEDLG data structure.

```
typedef FILEDLG *PFILEDLG;
```

PFILEFINDBUF4 Pointer to FILEFINDBUF4.

```
typedef FILEFINDBUF4 *PFILEFINDBUF4;
```

PFIXED Pointer to FIXED.

```
typedef FIXED *PFIXED;
```

PFN Pointer to procedure.

```
typedef int *PFN();
```

PFNWP Pointer to a window procedure.

```
typedef MRESULT (EXPENTRY *PFNWP)(HWND, USHORT, MPARAM, MPARAM);
```

PFONTDLG Pointer to a FONTDLG data structure.

```
typedef FONTDLG *PFONTDLG;
```

PFONTMETRICS Pointer to FONTMETRICS.

```
typedef FONTMETRICS *PFONTMETRICS;
```

PGRADIENTL Pointer to GRADIENTL.

```
typedef GRADIENTL *PGRADIENTL;
```

PHAB Pointer to HAB.

```
typedef HAB *PHAB;
```

PHBITMAP Pointer to HBITMAP.

```
typedef HBITMAP *PHBITMAP;
```

PHCINFO Pointer to HCINFO.

```
typedef HCINFO *PHCINFO;
```

PHDC Pointer to HDC.

```
typedef HDC *PHDC;
```

PHELPINIT Pointer to HELPINIT.

```
typedef HELPINIT *PHELPINIT;
```

PHELPSUBTABLE Pointer to SHORT.

```
typedef HELPSUBTABLE *PHELPSUBTABLE;
```

PHELPTABLE Pointer to a HELPTABLE data structure.

```
typedef HELPTABLE *PHELPTABLE;
```

PHFIND Pointer to HFIND.

```
typedef HFIND *PHFIND;
```

PHMF Pointer to HMF.

```
typedef HMF *PHMF;
```

PHMODULE Pointer to HMODULE.

```
typedef HMODULE *PHMODULE;
```

PHPAL Pointer to HPAL.

```
typedef HPAL *PHPAL;
```

PHPROGARRAY Pointer to HPROGARRAY.

```
typedef HPROGARRAY *PHPROGARRAY;
```

PHPROGRAM Pointer to HPROGRAM.

```
typedef HPROGRAM *PHPROGRAM;
```

PHPS Pointer to HPS.

```
typedef HPS *PHPS;
```

PHRGN Pointer to HRGN.

```
typedef HRGN *PHRGN;
```

PHSEM Pointer to HSEM.

```
typedef HSEM *PHSEM;
```

PHSWITCH Pointer to HSWITCH.

```
typedef HSWITCH *PHSWITCH;
```

PHWND Pointer to HWND.

```
typedef HWND *PHWND;
```

PIBSTRUCT Program-information-block structure.

```
typedef struct _PIBSTRUCT {
PROGTYPE     progt;
CHAR         szTitle[MAXNAMEL+1];
CHAR         szIconFileName[MAXPATHL+1];
CHAR         szExecutable[MAXPATHL+1];
CHAR         szStartupDir[MAXPATHL+1];
XYWINSIZE    xywinInitial;
USHORT       res1;
LHANDLE      res2;
USHORT       cchEnvironmentVars;
PCH          pchEnvironmentVars;
USHORT       cchProgramParameter;
PCH          pchProgramParameter;
 } PIBSTRUCT;
```

progt (PROGTYPE)
Program type and visibility.

szTitle[MAXNAMEL + 1] (CHAR)
Program title (null-terminated).

szIconFileName[MAXPATHL + 1] (CHAR)
Program icon filename (null-terminated).

szExecutable[MAXPATHL + 1] (CHAR)
Executable file name (null-terminated).

szStartupDir[MAXPATHL + 1] (CHAR)
Start-up directory (null-terminated).

xywinInitial (XYWINSIZE)
Initial window position and size.

res1 (USHORT)
Reserved; must be 0.

res2 (LHANDLE)
Reserved; must be NULLHANDLE.

cchEnvironmentVars (USHORT)
Environment string length.

pchEnvironmentVars (PCH)
Environment string.

cchProgramParameter (USHORT)
Parameter string length.

pchProgramParameter (PCH)
Parameter string.

PICONINFO Pointer to ICONINFO structure.

```
typedef ICONINFO *PICONINFO;
```

PICONPOS Pointer to IconPos data structure.

```
typedef ICONPOS *PICONPOS;
```

PID Process identity.

```
typedef LHANDLE PID;
```

PIMAGEBUNDLE Pointer to IMAGEBUNDLE.

```
typedef IMAGEBUNDLE *PIMAGEBUNDLE;
```

PIPT Pointer to IPT.

```
typedef IPT *PIPT;
```

PIX Pel count for multi-line entry field.

```
typedef LONG PIX;
```

PKERNINGPAIRS Pointer to KERNINGPAIRS.

```
typedef KERNINGPAIRS *PKERNINGPAIRS;
```

PLINEBUNDLE Pointer to LINEBUNDLE.

```
typedef LINEBUNDLE *PLINEBUNDLE;
```

PLONG Pointer to LONG.

```
typedef LONG *PLONG;
```

PMARGSTRUCT Pointer to a MLEMARGSTRUCT data structure.

```
typedef MLEMARGSTRUCT *PMARGSTRUCT;
```

PMARKERBUNDLE Pointer to MARKERBUNDLE.

```
typedef MARKERBUNDLE *PMARKERBUNDLE;
```

PMATRIXLF Pointer to MATRIXLF.

```
typedef MATRIXLF *PMATRIXLF;
```

PMENUITEM Pointer to a MENUITEM data structure.

```
typedef MENUITEM *PMENUITEM;
```

PMINIRECORDCORE Pointer to a MINIRECORDCORE data structure.

```
typedef MINIRECORDCORE *PMINIRECORDCORE;
```

POBJECTS Pointer to WPObject *.

```
typedef OBJECTS *POBJECTS;
```

PPALINFO Pointer to PALINFO.

```
typedef PALINFO *PPALINFO;
```

PPID Pointer to PID.

```
typedef PID *PPID;
```

PMLE_SEARCHDATA Pointer to a MLE_SEARCHDATA data structure.

```
typedef MLE_SEARCHDATA *PMLE_SEARCHDATA;
```

PMPARAM Pointer to a 4-byte message-dependent parameter structure.

```
typedef MPARAM *PMPARAM;
```

PMQINFO Pointer to MQINFO.

```
typedef MQINFO *PMQINFO;
```

PMRESULT Pointer to a 4-byte message-dependent reply parameter structure.

```
typedef MRESULT *PMRESULT;
```

PNOTIFYDELTA Pointer to a NOTIFYDELTA data structure.

```
typedef NOTIFYDELTA *PNOTIFYDELTA;
```

PNOTIFYRECORDEMPHASIS Pointer to a NOTIFYRECORDEMPHASIS data structure.

```
typedef NOTIFYRECORDEMPHASIS *PNOTIFYRECORDEMPHASIS;
```

PNOTIFYRECORDENTER Pointer to a NOTIFYRECORDENTER data structure.

```
typedef NOTIFYRECORDENTER *PNOTIFYRECORDENTER;
```

PNOTIFYSCROLL Pointer to a NOTIFYSCROLL data structure.

```
typedef NOTIFYSCROLL *PNOTIFYSCROLL;
```

POBJCLASS Pointer to an OBJCLASS data structure.

```
typedef OBJECTCLASS *POBJCLASS;
```

POBJDATA Pointer to OBJDATA structure.

```
typedef OBJDATA *POBJDATA;
```

POINTERINFO Pointer-information structure.

```
typedef struct _POINTERINFO {
ULONG      ulPointer;
LONG       xHotspot;
LONG       yHotspot;
HBITMAP    hbmPointer;
HBITMAP    hbmColor;
 } POINTERINFO;
```

ulPointer (ULONG)
Bit-map size indicator.

TRUE Pointer-sized bit map
FALSE Icon-sized bit map.

xHotspot (LONG)
x-coordinate of action point.

yHotspot (LONG)
y-coordinate of action point.

hbmPointer (HBITMAP)
Bit-map handle of pointer.

hbmColor (HBITMAP)
Bit-map handle of color bit map.

POINTL Point structure (long integer).

```
typedef struct _POINTL {
LONG    x;
LONG    y;
 } POINTL;
```

x (LONG)
x-coordinate.

y (LONG)
y-coordinate.

POINTS Point structure (short integer).

```
typedef struct _POINTS {
SHORT    x;
SHORT    y;
 } POINTS;
```

x (SHORT)
x-coordinate.

y (SHORT)
y-coordinate.

POLYGON Polygon structure.

```
typedef struct _POLYGON {
PPOINTL    pPointl;
LONG       lnumPoints;
 } POLYGON;
```

pPointl (PPOINTL)
Array of points.

lnumPoints (LONG)
number of points in array.

POVERFLOW Pointer to a MLEOVERFLOW data structure.

```
typedef MLEOVERFLOW *POVERFLOW;
```

POWNERBACKGROUND Pointer to an OWNERBACKGROUND data structure.

```
typedef OWNERBACKGROUND *POWNERBACKGROUND;
```

POWNERITEM Pointer to a OWNERITEM data structure.

```
typedef OWNERITEM *POWNERITEM;
```

PPAGEINFO Pointer to PAGEINFO structure.

```
typedef PAGEINFO *PPAGEINFO;
```

PPAGESELECTNOTIFY Pointer to a PAGESELECTNOTIFY data structure.

```
typedef PAGESELECTNOTIFY *PPAGESELECTNOTIFY;
```

PPIBSTRUCT Pointer to PIBSTRUCT.

```
typedef PIBSTRUCT *PPIBSTRUCT;
```

PPOINTL Pointer to a POINTL data structure.

```
typedef POINTL *PPOINTL;
```

PPOINTS Pointer to POINTS.

```
typedef POINTS *PPOINTS;
```

PPOLYGON Pointer to POLYGON.

```
typedef POLYGON *PPOLYGON;
```

PPRDINFO3 Pointer to PRDINFO3.

```
typedef PRDINFO3 *PPRDINFO3;
```

PPRDRIVINFO Pointer to PRDRIVINFO.

```
typedef PRDRIVINFO *PPRDRIVINFO;
```

PPRESPARAMS Pointer to PRESPARAMS.

```
typedef PRESPARAMS *PPRESPARAMS;
```

PPRINTDEST Pointer to PRINTDEST structure.

```
typedef PRINTDEST *PPRINTDEST;
```

PPRINTERINFO Pointer to PRINTERINFO.

```
typedef PRINTERINFO *PPRINTERINFO;
```

PPRJINFO2 Pointer to PRJINFO2.

```
typedef PRJINFO2 *PPRJINFO2;
```

PPRJINFO3 Pointer to PRJINFO3.

```
typedef PRJINFO3 *PPRJINFO3;
```

PPROGCATEGORY Pointer to PROGCATEGORY.

```
typedef PROGCATEGORY *PPROGCATEGORY;
```

PPROGDETAILS Pointer to PROGDETAILS.

```
typedef PROGDETAILS *PPROGDETAILS;
```

PPROGRAMENTRY Pointer to PROGRAMENTRY.

```
typedef PROGRAMENTRY *PPROGRAMENTRY;
```

PPROGTITLE Pointer to PROGTITLE.

```
typedef PROGTITLE *PPROGTITLE;
```

PPROGTYPE Pointer to PROGTYPE.

```
typedef PROGTYPE *PPROGTYPE;
```

PPRPORTINFO Pointer to PRPORTINFO.

```
typedef PRPORTINFO *PPRPORTINFO;
```

PPRPORTINFO1 Pointer to PRPORTINFO1.

```
typedef PRPORTINFO1 *PPRPORTINFO1;
```

PPRQINFO3 Pointer to PRQINFO3.

```
typedef PRQINFO3 *PPRQINFO3;
```

PPRQINFO6 Pointer to PRQINFO6.

```
typedef PRQINFO6 *PPRQINFO6;
```

PPRQPROCINFO Pointer to PRQPROCINFO.

```
typedef PRQPROCINFO *PPRQPROCINFO;
```

PPSZ Pointer to a PSZ pointer.

```
typedef char *PPSZ;
```

PPVOID Pointer to PVOID.

```
typedef PVOID *PPVOID;
```

PQMOPENDATA Open queue-manager data array.

This data type points to data whose format is described by the DEVOPENSTRUC data type.

```
typedef PSZ *PQMOPENDATA;
```

PQMSG Pointer to a QMSG data structure.

```
typedef QMSG *PQMSG;
```

PQUERYRECFROMRECT Pointer to a QUERYRECFROMRECT data structure.

```
typedef QUERYRECFROMRECT *PQUERYRECFROMRECT;
```

PQUERYRECORDRECT Pointer to a QUERYRECORDRECT data structure.

```
typedef QUERYRECORDRECT *PQUERYRECORDRECT;
```

PRDINFO3 Print device information structure (level 3).

```
typedef struct _PRDINFO3 {
PSZ       pszPrinterName;
PSZ       pszUserName;
PSZ       pszLogAddr;
USHORT    uJobId;
USHORT    fsStatus;
PSZ       pszStatus;
PSZ       pszComment;
PSZ       pszDrivers;
USHORT    time;
USHORT    usTimeOut;
 } PRDINFO3;
```

pszPrinterName (PSZ)
Print device name.

pszUserName (PSZ)
User who submitted job.

This parameter is valid only while the job is printing. It is NULL for a job submitted locally.

pszLogAddr (PSZ)
Logical address (for example LPT1).

If NULL or an empty string, the printer is not connected to a logical address.

uJobId (USHORT)
Identity of current job.

If 0, no job is printing.

fsStatus (USHORT)
Print destination status.

Use the mask PRD_STATUS_MASK to determine the print job status:

PRD_ACTIVE	Processing
PRD_PAUSED	Not processing, or paused.

Use the mask PRJ_DEVSTATUS for further information about print job status:

PRJ_COMPLETE	Job complete
PRJ_INTERV	Intervention required
PRJ_ERROR	Error occurred (in this case, *pszStatus* may contain a comment about the error)
PRJ_DESTOFFLINE	Print device offline
PRJ_DESTPAUSED	Print device paused
PRJ_NOTIFY	Raise alert
PRJ_DESTNOPAPER	Print device out of paper.

pszStatus (PSZ)
Print device comment while printing.

A comment posted by the print processor of the print device. This parameter is valid only during printing.

pszComment (PSZ)
Print device description.

pszDrivers (PSZ)
Drivers supported by print device.

List items are separated by commas. Each printer driver name may have a device name separated by a dot (for example, PLOTTER.HP7475A). The default printer is listed first.

time (USHORT)
Time job has been printing (minutes).

This parameter applies only during printing.

usTimeOut (USHORT)
Device timeout (seconds).

The time that elapses before the device driver notifies the spooler that the print device has not responded.

PRDRIVINFO Printer driver information structure (level 0).

```
typedef struct _PRDRIVINFO {
CHAR    szDriverName[DRIV_NAME_SIZE+DRIV_DEVICENAME_SIZE+2];
 } PRDRIVINFO;
```

szDriverName[DRIV_NAME_SIZE+DRIV_DEVICENAME_SIZE+2] (CHAR)
Name of printer driver.

This is the name of the printer driver and device is the format of DRIVER.DEVICE. For example "IBM4019.IBM Laserprinter E."

PRECORDCORE Pointer to a RECORDCORE data structure.

```
typedef RECORDCORE *PRECORDCORE;
```

PRECORDINSERT Pointer to a RECORDINSERT data structure.

```
typedef RECORDINSERT *PRECORDINSERT;
```

PRECTL

Pointer to a RECTL data structure.

```
typedef RECTL *PRECTL;
```

PRENDERFILE

Pointer to RENDERFILE.

```
typedef RENDERFILE *PRENDERFILE;
```

PRESPARAMS

Presentation parameter data.

```
typedef struct _PRESPARAMS {
ULONG    cb;
PARAM    aparam[1];
 } PRESPARAMS;
```

cb (ULONG)
Byte count of the *aparam*[*1*] parameter.

aparam[**1**] (PARAM)
Array of attribute parameters.

PRFPROFILE

Profile structure.

```
typedef struct _PRFPROFILE {
ULONG    cchUserName;
PSZ      pszUserName;
ULONG    cchSysName;
PSZ      pszSysName;
 } PRFPROFILE;
```

cchUserName (ULONG)
Length of user profile name.

pszUserName (PSZ)
User profile name.

cchSysName (ULONG)
Length of system profile name.

pszSysName (PSZ)
System profile name.

PRGB2

Pointer to RGB2.

```
typedef RGB2 *PRGB2;
```

PRGNRECT

Pointer to RGNRECT.

```
typedef RGNRECT *PRGNRECT;
```

PRINTDEST

PRINTDEST data structure.

Contains all the parameters required to issue a DevPostDeviceModes and DevOpenDC function calls.

```
typedef struct _PRINTDEST {
ULONG          cb;
LONG           lType;
PSZ            pszToken;
LONG           lCount;
PDEVOPENDATA   pdopData;
ULONG          fl;
PSZ            pszPrinter;
 } PRINTDEST;
```

cb (ULONG)
Length of data structure, in bytes.

The value is always 28.

lType (LONG)
Type of device context.

OD_QUEUED	The device context is queued.
OD_DIRECT	The device context is direct.

pszToken (PSZ)
Device-information token.

This is always "*."

lCount (LONG)
Number of items.

This is the number of items present in the *pdopData* field.

pdopData (PDEVOPENDATA)
Open device context data area.

See DEVOPENSTRUC for information on the format of *pdopData*.

fl (ULONG)
Flags.

PD_JOB_PROPERTY	This flag indicates that DevPostDeviceModes should be called with DPDM_POSTJOBPROP before calling DevOpenDC.

pszPrinter (PSZ)
Name of Printer.

A name that specifies the device, for example "PRINTER1." The name is used for calling DevPostDeviceModes.

PRINTERINFO

Print destination information structure.

This structure is used at information level 0.

```
typedef struct _PRINTERINFO {
ULONG    flType;
PSZ      pszComputerName;
PSZ      pszPrintDestinationName;
PSZ      pszDescription;
PSZ      pszLocalName;
 } PRINTERINFO;
```

flType (ULONG)
Type of printer.

This is a flag used to describe the type of print destination:

SPL_PR_QUEUE	Print destination is a queue
SPL_PR_DIRECT_DEVICE	Print destination is a direct print device
SPL_PR_QUEUED_DEVICE	Print destination is a queued print device

pszComputerName (PSZ)
Computer name.

A NULL string specifies the local workstation.

pszPrintDestinationName (PSZ)
Name of Print Destination.

It is either a queue name or a print device name depending upon the value of *flType*. The maximum length of the name in the network case is 256 (including one byte for the null terminator).

pszDescription (PSZ)
Description of print destination.

The maximum length is 48 characters (including one byte for the null terminator).

pszLocalName (PSZ)
Local name of remote print destination.

This is a local port name (for instance "LPT4") that is connected to the remote print destination. A NULL string specifies that no connection exists.

PRJINFO2 Print-job information structure.

This structure provides a subset of the information supplied by PRJINFO3. It minimizes the storage required for job-information retrieval, and is sufficient for most uses.

```
typedef struct _PRJINFO2 {
USHORT    uJobId;
USHORT    uPriority;
PSZ       pszUserName;
USHORT    uPosition;
USHORT    fsStatus;
ULONG     ulSubmitted;
ULONG     ulSize;
PSZ       pszComment;
PSZ       pszDocument;
 } PRJINFO2;
```

uJobId (USHORT)
Job identification number.

uPriority (USHORT)
Job priority.

The job-priority range is 1 through 99, with 99 the highest job priority. (For queue priorities, 1 is the highest priority.)

The job priority determines the order of jobs in the queue. If multiple queues print to the same printer, the job at the front of each queue is examined. The job with the highest priority is printed first; if there is more than one job with the highest priority, the oldest job with this priority is printed first.

PRJ_MAX_PRIORITY	Highest priority
PRJ_MIN_PRIORITY	Lowest priority
PRJ_NO_PRIORITY	No priority.

pszUserName (PSZ)
User who submitted the job.

This parameter applies only to jobs created by a user and enqueued on a remote server. A NULL string signifies a local job.

uPosition (USHORT)
Job position in queue.

If 1, the job is scheduled to be the next job printed from this queue.

fsStatus (USHORT)
Job status.

To find the job status, use the PRJ_QSTATUS mask:

PRJ_QS_QUEUED	Queued
PRJ_QS_PAUSED	Paused by a SplHoldJob function
PRJ_QS_SPOOLING	Job being created
PRJ_QS_PRINTING	Printing (bits 2 through 11 are valid).

For further information, use the PRJ_DEVSTATUS mask:

PRJ_COMPLETE	Job complete
PRJ_INTERV	Intervention required
PRJ_ERROR	Error occurred.
PRJ_DESTOFFLINE	Print destination offline
PRJ_DESTPAUSED	Print destination paused
PRJ_NOTIFY	Alert should be raised
PRJ_DESTNOPAPER	Print destination out of paper
PRJ_DESTFORMCHG	Printer waiting for form change
PRJ_DESTCRTCHG	Printer waiting for cartridge change
PRJ_DESTPENCHG	Printer waiting for pen change.

This bit indicates that the job is deleted:

PRJ_DELETED	Job deleted.

ulSubmitted (ULONG)
Time job submitted.

Time format is the same as that stored in the global information segment.

ulSize (ULONG)
Print-job size (bytes).

pszComment (PSZ)
Comment string.

Information about the print job. The maximum length of the string is 48 characters(including one byte for the null terminator).

pszDocument (PSZ)
Document name.

The document name of the print job (set by the application that submitted the print job). The maximum length of the string is 260 characters.

PRJINFO3

Print-job information structure.

This structure is used when complete job details are required. A subset of this information is supplied by PRJINFO2.

```
typedef struct _PRJINF03 {
USHORT       uJobId;
USHORT       uPriority;
PSZ          pszUserName;
USHORT       uPosition;
USHORT       fsStatus;
ULONG        ulSubmitted;
ULONG        ulSize;
PSZ          pszComment;
PSZ          pszDocument;
PSZ          pszNotifyName;
PSZ          pszDataType;
PSZ          pszParms;
PSZ          pszStatus;
PSZ          pszQueue;
PSZ          pszQProcName;
PSZ          pszQProcParms;
PSZ          pszDriverName;
PDRIVDATA    pDriverData;
PSZ          pszPrinterName;
 } PRJINF03;
```

uJobId (USHORT)
Job identification number.

uPriority (USHORT)
Job priority.

The job-priority range is 1 through 99, with 99 the highest job priority. (For queue priorities, 1 is the highest priority.)

The job priority determines the order of jobs in the queue. If multiple queues print to the same printer, the job on the front of each queue is examined. The job with the highest priority is printed first; if there is more than one job with the highest priority, the oldest job with this priority is printed first.

PRJ_MAX_PRIORITY	Highest priority
PRJ_MIN_PRIORITY	Lowest priority
PRJ_NO_PRIORITY	No priority.

pszUserName (PSZ)
User who submitted the job.

This parameter applies only to jobs created by a user on a remote workstation and queued on a server. A NULL string signifies a local job.

uPosition (USHORT)
Job position in queue.

If 1, the job is scheduled to be the next job printed from this queue.

fsStatus (USHORT)
Job status.

To find the job status, use the PRJ_QSTATUS mask:

PRJ_QS_QUEUED	Queued
PRJ_QS_PAUSED	Paused by a SplHoldJob function
PRJ_QS_SPOOLING	Job being created
PRJ_QS_PRINTING	Printing (bits 2 through 11 are valid).

For further information, use the PRJ_DEVSTATUS mask:

PRJ_COMPLETE	Job complete
PRJ_INTERV	Intervention required
PRJ_ERROR	Error occurred. (In this case, *pszStatus* may contain a comment about the error)
PRJ_DESTOFFLINE	Print destination offline
PRJ_DESTPAUSED	Print destination paused
PRJ_NOTIFY	Alert should be raised
PRJ_DESTNOPAPER	Print destination out of paper
PRJ_DESTFORMCHG	Printer waiting for form change
PRJ_DESTCRTCHG	Printer waiting for cartridge change
PRJ_DESTPENCHG	Printer waiting for pen change.

This bit indicates that the job is deleted:

PRJ_DELETED	Job deleted.

ulSubmitted (ULONG)
Time job submitted.

Time format is the same as that stored in the global information segment.

ulSize (ULONG)
Print-job size (bytes).

pszComment (PSZ)
Comment string.

Information about the print job.

The maximum length of the string is 48 characters (including one byte for the null terminator).

pszDocument (PSZ)
Document name.

The document name of the print job (set by the application that submitted the print job). The maximum length of the string is 260 characters.

pszNotifyName (PSZ)
Messaging alias for print alert.

This parameter is a computer name and applies only to jobs on a remote server queue. A NULL string is returned for jobs on a local queue.

pszDataType (PSZ)
Data type of submitted file.

This is specified by the *pszDataType* parameter in the DEVOPENSTRUC structure passed to the DevOpenDC call when the job is created. The name is truncated to fit the field if necessary, and contains a trailing NULL.

pszParms (PSZ)
Parameters.

The form of this string is:

```
parm1=val1 parm2=val2 ...
```

pszStatus (PSZ)
Status comment.

A text string, posted by the queue processor, that provides additional job-status information. The default string type is NULL.

pszQueue (PSZ)
Queue name.

The name of the queue the job is on.

pszQProcName (PSZ)
Queue processor.

The name of the queue processor.

pszQProcParms (PSZ)
Queue processor parameters.

Spaces are used to separate parameters.

pszDriverName (PSZ)
Driver name.

The name of the device driver (for example, "LASERJET"). The device name is part of *pDriverData*.

pDriverData (PDRIVDATA)
Job Properties (driver data).

The contents are specific to the device driver.

pszPrinterName (PSZ)
Printer name.

If the job is printing, the printer name, otherwise NULL.

PROGCATEGORY

Program category.

```
typedef CHAR PROGCATEGORY;
```

PROGDETAILS

Program-details structure.

```
typedef struct _PROGDETAILS {
ULONG       Length;
PROGTYPE    progt;
USHORT      pad1[3];
PSZ         pszTitle;
PSZ         pszExecutable;
PSZ         pszParameters;
PSZ         pszStartupDir;
PSZ         pszIcon;
PSZ         pszEnvironment;
SWP         swpInitial;
USHORT      pad2[5];
 } PROGDETAILS;
```

Length (ULONG)
Length of structure.

progt (PROGTYPE)
Program type.

pad1[3] (USHORT)
Reserved.

pszTitle (PSZ)
Title.

pszExecutable (PSZ)
Executable file name.

pszParameters (PSZ)
Parameter string.

pszStartupDir (PSZ)
Start-up directory.

pszIcon (PSZ)
Icon-file name.

pszEnvironment (PSZ)
Environment string.

A list of null-terminated strings, ending with an extra null.

swpInitial (SWP)
Initial window position and size.

pad2[**5**] (USHORT)
Reserved.

PROGRAMENTRY

Program-entry structure.

```
typedef struct _PROGRAMENTRY {
HPROGRAM     hprog;
PROGTYPE     progt;
CHAR         szTitle[MAXNAMEL+1];
 } PROGRAMENTRY;
```

hprog (HPROGRAM)
Program handle.

progt (PROGTYPE)
Program type.

szTitle[**MAXNAMEL + 1**] (CHAR)
Program title (null-terminated).

PROGTITLE

Program-title structure.

```
typedef struct _PROGTITLE {
HPROGRAM     hprog;
PROGTYPE     progt;
USHORT       pad1[3];
PSZ          pszTitle;
 } PROGTITLE;
```

hprog (HPROGRAM)
Program handle.

progt (PROGTYPE)
Program type.

pad1[**3**] (USHORT)
Reserved.

pszTitle (PSZ)
Program title.

PROGTYPE

Program-type structure.

```
typedef struct _PROGTYPE {
PROGCATEGORY     progc;
UCHAR            fbVisible;
 } PROGTYPE;
```

progc (PROGCATEGORY)
Program category:

PROG_DEFAULT	Default application
PROG_PM	Presentation Manager application
PROG_WINDOWABLEVIO	Text-windowed application
PROG_FULLSCREEN	Full-screen application
PROG_WINDOWEDVDM	PC DOS executable process (windowed)
PROG_VDM	PC DOS executable process (full screen)
PROG_REAL	PC DOS executable process (full screen). Same as PROG_VDM.

PROG_WINDOW_REAL	Windows program which requires Windows Real mode to execute
PROG_WINDOW_PROT	Windows program which will execute in Windows protect mode

fbVisible (UCHAR)
Visibility attribute.

When testing this field, allow for the possibility that other bits may be defined in the future. SHE_INVISIBLE and SHE_PROTECTED can be used to mask the visibility and protected flags, respectively.

SHE_VISIBLE	Visible
SHE_INVISIBLE	Invisible
SHE_UNPROTECTED	Unprotected
SHE_PROTECTED	Protected.

PRPORTINFO

Port information structure (level 0).

```
typedef struct _PRPORTINFO {
CHAR    szPortName[PDLEN+1];
 } PRPORTINFO;
```

szPortName[PDLEN+1] (CHAR)
Name of the port.

This is the name of the port. For example "LPT1."

PRPORTINFO1

Port information structure (level 1).

```
typedef struct _PRPORTINFO1 {
PSZ    pszPortName;
PSZ    pszPortDriverName;
PSZ    pszPortDriverPathName;
 } PRPORTINFO1;
```

pszPortName (PSZ)
Name of the port.

This is the name of the port. For example "LPT1."

pszPortDriverName (PSZ)
Name of the port driver.

This is the name of the port driver. For example "PARALLEL."

pszPortDriverPathName (PSZ)
Full path name of the port driver.

This is the full path name of the port driver. For example "C:\OS2\DLL\PARALLEL.PDR."

PRQINFO3

Print-queue information structure.

This structure is used at information levels 3 and 4.

```
typedef struct _PRQINFO3 {
PSZ          pszName;
USHORT       uPriority;
USHORT       uStartTime;
USHORT       uUntilTime;
USHORT       fsType;
PSZ          pszSepFile;
PSZ          pszPrProc;
PSZ          pszParms;
PSZ          pszComment;
USHORT       fsStatus;
USHORT       cJobs;
PSZ          pszPrinters;
PSZ          pszDriverName;
PDRIVDATA    pDriverData;
 } PRQINFO3;
```

pszName (PSZ)
Queue name.

The maximum length of the name in the network case is 256 (including one byte for zero termination).

uPriority (USHORT)
Queue priority.

The range is 1 through 9, with 1 being the highest queue priority.

The default job priority (DefJobPrio) is determined from: DefJobPrio=100–(10* *uPriority*).

If a job is added with *PRJ_NO_PRIORITY* specified, DefJobPrio is used. If a default priority higher than the default job priority is specified, the default job priority is used. If a default priority lower than the default is specified, the specified job priority is used.

PRQ_DEF_PRIORITY	Default priority
PRQ_MAX_PRIORITY	Highest priority
PRQ_MIN_PRIORITY	Minimum priority
PRQ_NO_PRIORITY	No priority.

uStartTime (USHORT)
Minutes after midnight when queue becomes active.

For example, the value 75 represents 1:15 a.m.

If *uStartTime* and *uUntilTime* are both 0, the print queue is always available.

uUntilTime (USHORT)
Minutes after midnight when queue ceases to be active.

For example, the value 1200 represents 8 p.m.

If *uUntilTime* and *uStartTime* are both 0, the print queue is always available.

fsType (USHORT)
Queue type.

PRQ3_TYPE_RAW	Data is always enqueued in the device specific format.
PRQ3_TYPE_QP_BYPASS	Allows the spooler to bypass the queue processor and send data directly to the Printer Driver. Setting this bit allows the spooler to print jobs of type PM_Q_RAW while they are still being spooled.

pszSepFile (PSZ)
Separator-page file.

The path and file name of a separator-page file on the target computer.

This file contains formatting information for the page or pages to be used between print jobs. A relative path name is taken as relative to the current spool directory. A NULL string indicates no separator page.

See *IBM Operating System/2 Local Area Network Server Version 1.2: Network Administrator's Guide* for information about the format of separator files.

pszPrProc (PSZ)
Default queue-processor.

pszParms (PSZ)
Queue parameters.

This can be any text string or a NULL string.

pszComment (PSZ)
Queue description.

A NULL string results in no comment. The maximum length is 48 characters (including one byte for the null terminator).

fsStatus (USHORT)
Queue status.

PRQ3_PAUSED Queue is paused (held).
PRQ3_PENDING Queue is pending deletion.

cJobs (USHORT)
Number of jobs in queue.

pszPrinters (PSZ)
Print devices connected to queue.

This cannot be NULL.

pszDriverName (PSZ)
Default device driver.

pDriverData (PDRIVDATA)
Default queue job properties.

Note: An application can use *pszDriverName*, *pDriverData*, *pszPrProc*, and *pszParms* to construct a valid DevOpenDC call based only on the queue name.

PRQINFO6

Print-queue information structure.

This structure is used at information level 6.

```
typedef struct _PRQINFO6 {
PSZ          pszName;
USHORT       uPriority;
USHORT       uStartTime;
USHORT       uUntilTime;
USHORT       fsType;
PSZ          pszSepFile;
PSZ          pszPrProc;
PSZ          pszParms;
PSZ          pszComment;
USHORT       fsStatus;
USHORT       cJobs;
PSZ          pszPrinters;
PSZ          pszDriverName;
PDRIVDATA    pDriverData;
PSZ          pszRemoteComputerName;
PSZ          pszRemoteQueueName;
 } PRQINFO6;
```

pszName (PSZ)
Queue name.

The maximum length of the name in the network case is 256 (including one byte for zero termination).

uPriority (USHORT)
Queue priority.

The range is 1 through 9, with 1 being the highest queue priority.

The default job priority (DefJobPrio) is determined from: DefJobPrio=100–(10* *uPriority*).

If a job is added with *PRJ_NO_PRIORITY* specified, DefJobPrio is used. If a default priority higher than the default job priority is specified, the default job priority is used. If a default priority lower than the default is specified, the specified job priority is used.

PRQ_DEF_PRIORITY Default priority
PRQ_MAX_PRIORITY Highest priority

PRQ_MIN_PRIORITY Minimum priority
PRQ_NO_PRIORITY No priority.

uStartTime (USHORT)
Minutes after midnight when queue becomes active.

For example, the value 75 represents 1:15 a.m.

If *uStartTime* and *uUntilTime* are both 0, the print queue is always available.

uUntilTime (USHORT)
Minutes after midnight when queue ceases to be active.

For example, the value 1200 represents 8 p.m.

If *uUntilTime* and *uStartTime* are both 0, the print queue is always available.

fsType (USHORT)
Queue type.

PRQ3_TYPE_RAW Data is always enqueued in the device specific format.
PRQ3_TYPE_QP_BYPASS Allows the spooler to bypass the queue processor and send data directly to the Printer Driver. Setting this bit allows the spooler to print jobs of type PM_Q_RAW while they are still being spooled.

pszSepFile (PSZ)
Separator-page file.

The path and file name of a separator-page file on the target computer.

This file contains formatting information for the page or pages to be used between print jobs. A relative path name is taken as relative to the current spool directory. A NULL string indicates no separator page.

See *IBM Operating System/2 Local Area Network Server Version 1.2: Network Administrator's Guide* for information about the format of separator files.

pszPrProc (PSZ)
Default queue-processor.

pszParms (PSZ)
Queue parameters.

This can be any text string or a NULL string.

pszComment (PSZ)
Queue description.

A NULL string results in no comment. The maximum length is 48 characters (including one byte for the null terminator).

fsStatus (USHORT)
Queue status.

PRQ3_PAUSED Queue is paused (held).
PRQ3_PENDING Queue is pending deletion.

cJobs (USHORT)
Number of jobs in queue.

pszPrinters (PSZ)
Print devices connected to queue.

This cannot be NULL.

pszDriverName (PSZ)
Default device driver.

pDriverData (PDRIVDATA)
Default queue job properties.

Note: An application can use *pszDriverName*, *pDriverData*, *pszPrProc*, and *pszParms* to construct a valid DevOpenDC call based only on the queue name.

pszRemoteComputerName (PSZ)
Remote computer name.

The computer name part of a remote queue for which this queue is a local alias.

pszRemoteQueueName (PSZ)
Remote queue name.

The queue name part of a remote queue for which this queue is a local alias.

PRQPROCINFO Queue processor information structure (level 0).

```
typedef struct _PRQPROCINFO {
CHAR    szQProcName[DRIV_NAME_SIZE+1];
 } PRQPROCINFO;
```

szQProcName[DRIV_NAME_SIZE+1] (CHAR)
Name of queue processor.

This is the name of the queue processor (driver). For example "PMPRINT."

PSBCDATA Pointer to SBCDATA.

```
typedef SBCDATA *PSBCDATA;
```

PSEARCHSTRING Pointer to a SEARCHSTRING data structure.

```
typedef SEARCHSTRING *PSEARCHSTRING;
```

PSFACTORS Pointer to SFACTORS.

```
typedef SFACTORS *PSFACTORS;
```

PSHORT Pointer to SHORT.

```
typedef SHORT *PSHORT;
```

PSIZEF Pointer to SIZEF.

```
typedef SIZEF *PSIZEF;
```

PSIZEL Pointer to SIZEL.

```
typedef SIZEL *PSIZEL;
```

PSLDCDATA Pointer to a SLDCDATA data structure.

```
typedef SLDCDATA *PSLDCDATA;
```

PSTRL Pointer to PSZ.

```
typedef STRL *PSTRL;
```

PSTR8 Pointer to STR8.

```
typedef STRL *PSTR8;
```

PSTR16 Pointer to STR16.

```
typedef STR16 *PSTR16;
```

PSTR32 Pointer to STR32.

```
typedef STR32 *PSTR32;
```

PSTR64 Pointer to STR64.

```
typedef STR64 *PSTR64;
```

PSTYLECHANGE Pointer to a STYLECHANGE data structure.

```
typedef STYLECHANGE *PSTYLECHANGE;
```

PSWBLOCK Pointer to a switch-list block structure.

```
typedef SWBLOCK *PSWBLOCK;
```

PSWCNTRL Pointer to a switch-list control block structure.

```
typedef SWCNTRL *PSWCNTRL;
```

PSWENTRY Pointer to SWENTRY.

```
typedef SWENTRY *PSWENTRY;
```

PSWP Pointer to a SWP data structure.

```
typedef SWP *PSWP;
```

PSZ Pointer to a null-terminated string.

```
typedef char *PSZ;
```

PTID Pointer to TID.

```
typedef TID *PTID;
```

PTRACKINFO Pointer to a TRACKINFO data structure.

```
typedef TRACKINFO *PTRACKINFO;
```

PTREEITEMDESC Pointer to a TREEITEMDESC data structure.

```
typedef TREEITEMDESC *PTREEITEMDESC;
```

PUCHAR Pointer to UCHAR.

```
typedef UCHAR *PUCHAR;
```

PULONG Pointer to ULONG.

```
typedef ULONG *PULONG;
```

PUSERBUTTON Pointer to USERBUTTON.

```
typedef USERBUTTON *PUSERBUTTON;
```

PUSEITEM Pointer to USEITEM data structure.

```
typedef USEITEM *PUSEITEM;
```

PUSHORT Pointer to USHORT.

```
typedef USHORT *PUSHORT;
```

PVIOFONTCELLSIZE Pointer to VIOFONTCELLSIZE.

```
typedef VIOFONTCELLSIZE *PVIOFONTCELLSIZE;
```

PVIOSIZECOUNT Pointer to VIOSIZECOUNT.

```
typedef VIOSIZECOUNT *PVIOSIZECOUNT;
```

PVOID Pointer to a data type of undefined format.

```
typedef VOID *PVOID;
```

PVSCDATA Pointer to VSCDATA.

```
typedef VSCDATA *PVSCDATA;
```

PVSDRAGINFO Pointer to VSDRAGINFO.

```
typedef VSDRAGINFO *PVSDRAGINFO;
```

PVSDRAGINIT Pointer to VSDRAGINIT.

```
typedef VSDRAGINIT *PVSDRAGINIT;
```

PVSTEXT Pointer to a VSTEXT data structure.

```
typedef VSTEXT *PVSTEXT;
```

PWNDPARAMS Pointer to a WNDPARAMS data structure.

```
typedef WNDPARAMS *PWNDPARAMS;
```

PWPOINT Pointer to a WPOINT data structure.

```
typedef WPOINT *PWPOINT;
```

QMOPENSTRUC Open queue manager data structure.

```
typedef struct _QMOPENSTRUC {
PSZ          pszQueueName;
PSZ          pszDriverName;
PDRIVDATA    pdrivDriverData;
PSZ          pszDataType;
PSZ          pszComment;
PSZ          pszQueueProcName;
PSZ          pszQueueProcParams;
PSZ          pszSpoolerParams;
PSZ          pszNetworkParams;
 } QMOPENSTRUC;
```

pszQueueName (PSZ)
Queue name.

The name of the queue for the output device. The queue can be a UNC name.

pszDriverName (PSZ)
Driver name.

A string containing the name of the Presentation Manager Device Driver (for example, "IBM4019").

pdrivDriverData (PDRIVDATA)
Driver data.

Data which is to be passed directly to the Presentation Manager Device Driver. Whether or not any of this is required depends upon the Device Driver.

pszDataType (PSZ)
Data type.

This defines the type of data which is to be queued, as follows:

- "PM_Q_STD" - standard format
- "PM_Q_RAW" - raw format

Note that a Presentation Manager device driver may define other datatypes and may not support all of these queued data types.

pszComment (PSZ)
Comment.

A natural language description of the file. This may, for example, be displayed by the spooler to the end user. It is optional.

pszQueueProcName (PSZ)
Queue processor name.

The name of the queue processor. This is normally the default.

pszQueueProcParams (PSZ)
Queue processor parameters.

A parameter string for the queue processor. It is optional.

pszSpoolerParams (PSZ)
Spooler parameters.

A parameter string for the spooler, which is optional. This has the following options, which must be separated by one or more blanks:

- FORM=f

  Specifies a forms code 'f'. This must be a valid forms code for the printer.

  If not specified, then the data is printed on the forms in use, when this print job is ready to be printed.

- PRTY=n

Specifies a priority in the range 0-99, with 99 being the highest. If not specified, then a priority of 50 is used.

pszNetworkParams (PSZ)
Network parameters.

The format of the parameter string is keyword=value, and the following keywords are defined (additional ones can be defined by the network program):

- USER=u

  specifies the userid 'u'. If not specified, a null userid is used.

QMSG

Message structure.

```
typedef struct _QMSG {
HWND      hwnd;
ULONG     msg;
MPARAM    mp1;
MPARAM    mp2;
ULONG     time;
POINTL    ptl;
 } QMSG;
```

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.

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 24-41 for information about that message.

```
typedef struct _QUERYRECFROMRECT {
ULONG    cb;
RECTL    rect;
ULONG    fsSearch;
 } QUERYRECFROMRECT;
```

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 that bounds a specified container record. This structure is used in the CM_QUERYRECORDRECT container message only. See "CM_QUERYRECORDRECT" on page 24-43 for information about that message.

```
typedef struct _QUERYRECORDRECT {
ULONG          cb;
PRECORDCORE    pRecord;
ULONG          fsExtent;
ULONG          fRightSplitWindow;
 } QUERYRECORDRECT;
```

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.

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.

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.

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.

```
typedef struct _RECORDCORE {
ULONG           cb;
ULONG           flRecordAttr;
POINTL          ptlIcon;
PRECORDCORE     pNextRecord;
PSZ             pszIcon;
HPOINTER        hptrIcon;
HPOINTER        hptrMiniIcon;
HBITMAP         hbmBitmap;
HBITMAP         hbmMiniBitmap;
PTREEITEMDESC   pTreeItemDesc;
PSZ             pszText;
PSZ             pszName;
PSZ             pszTree;
 } RECORDCORE;
```

cb (ULONG)
Structure size.

The size (in bytes) of the RECORDCORE structure.

flRecordAttr (ULONG)
Record attributes.

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.

pNextRecord (PRECORDCORE)
Pointer.

Pointer to the next linked record.

pszIcon (PSZ)
Text.

Text for the icon view (CV_ICON).

hptrIcon (HPOINTER)
Icon.

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)
Mini-icon.

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.

Bit map that is 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)
Mini-bit map.

Bit map that is 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.

Pointer to a TREEITEMDESC structure, which 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 view text.

Text for the text view (CV_TEXT).

pszName (PSZ)
Name view text.

Text for the name view (CV_NAME).

pszTree (PSZ)
Tree view text.

Text for the tree view (CV_TREE).

RECORDITEM

USAGE_RECORD structure.

```
typedef RECORDITEM FAR *RECORDITEM;
```

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 24-31 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.

```
typedef struct _RECORDINSERT {
ULONG          cb;
PRECORDCORE    pRecordOrder;
PRECORDCORE    pRecordParent;
ULONG          zOrder;
ULONG          cRecordsInsert;
ULONG          fInvalidateRecord;
 } RECORDINSERT;
```

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.

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.

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.

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.

RECTL

Rectangle structure.

```
typedef struct _RECTL {
LONG    xLeft;
LONG    yBottom;
LONG    xRight;
LONG    yTop;
 } RECTL;
```

xLeft (LONG)
x-coordinate of left-hand edge of rectangle.

yBottom (LONG)
y-coordinate of bottom edge of rectangle.

xRight (LONG)
x-coordinate of right-hand edge of rectangle.

yTop (LONG)
y-coordinate of top edge of rectangle.

RENDERFILE

File-rendering structure.

```
typedef struct _RENDERFILE {
HWND      hwndDragFiles;
HSTR      hstrSource;
HSTR      hstrTarget;
BOOL      fMove;
USHORT    usReserved;
 } RENDERFILE;
```

hwndDragFiles (HWND)
Conversation handle.

Created by DrgDragFiles.

hstrSource (HSTR)
Handle to source file name.

hstrTarget (HSTR)
Handle to target file name.

fMove (BOOL)
Operation.

TRUE Move the file.
FALSE Copy the file.

usReserved (USHORT)
Reserved.

RGB

RGB color value.

```
typedef struct _RGB {
BYTE    bBlue;
BYTE    bGreen;
BYTE    bRed;
 } RGB;
```

bBlue (BYTE)
Blue component of the color definition.

bGreen (BYTE)
Green component of the color definition.

bRed (BYTE)
Red component of the color definition.

RGB2

RGB color value.

```
typedef struct _RGB2 {
BYTE    bBlue;
BYTE    bGreen;
BYTE    bRed;
BYTE    fcOptions;
 } RGB2;
```

bBlue (BYTE)
Blue component of the color definition.

bGreen (BYTE)
Green component of the color definition.

bRed (BYTE)
Red component of the color definition.

fcOptions (BYTE)
Entry options.

These can be ORed together if required:

PC_RESERVED The color entry is reserved for animating color with the palette manager.

PC_EXPLICIT The low-order word of the color table entry designates a physical palette slot. This allows an application to show the actual contents of the device palette as realized for other logical palettes. This does not

prevent the color in the slot from being changed for any reason.

RGNRECT

Region-rectangle structure.

```
typedef struct _RGNRECT {
ULONG    ircStart;
ULONG    crc;
ULONG    crcReturned;
ULONG    ulDirection;
 } RGNRECT;
```

ircStart (ULONG)
Rectangle number from which to start enumerating.

Numbering starts from 1.

crc (ULONG)
Number of rectangles that can be returned.

This must be 1 or greater.

crcReturned (ULONG)
Number of rectangles returned.

A value of less than *crc* indicates that there are no more rectangles to enumerate.

ulDirection (ULONG)
Direction in which the returned rectangles are to be ordered.

This ordering uses the leading edge of a rectangle.

RECTDIR_LFRT_TOPBOT	Left-to-right, top-to-bottom
RECTDIR_RTLF_TOPBOT	Right-to-left, top-to-bottom
RECTDIR_LFRT_BOTTOP	Left-to-right, bottom-to-top
RECTDIR_RTLF_BOTTOP	Right-to-left, bottom-to-top.

SBCDATA

Scroll-bar control data structure.

```
typedef struct _SBCDATA {
USHORT    cb;
USHORT    sHilite;
SHORT     posFirst;
SHORT     posLast;
SHORT     posThumb;
SHORT     cVisible;
SHORT     cTotal;
 } SBCDATA;
```

cb (USHORT)
Length of control data in bytes.

10 The length of the control data for a scroll-bar control.

sHilite (USHORT)
Highlighting code.

This indicates which part of the scroll bar is to be highlighted, if any.

ZERO	No highlighting
SB_LINEUP	Line up arrow
SB_LINELEFT	Line left arrow
SB_LINEDOWN	Line down arrow
SB_LINERIGHT	Line right arrow
SB_PAGEUP	Page up arrow
SB_PAGELEFT	Page left arrow
SB_PAGEDOWN	Page down arrow
SB_PAGERIGHT	Page right arrow
SB_SLIDERTRACK	Slider.

posFirst (SHORT)
First bound of the scroll-bar range.

posLast (SHORT)
Last bound of the scroll-bar range.

posThumb (SHORT)
Slider position.

cVisible (SHORT)
Number of data items visible.

cTotal (SHORT)
Number of data items available.

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 24-48 for information about that message.

```
typedef struct _SEARCHSTRING {
ULONG    cb;
PSZ      pszSearch;
ULONG    ulView;
ULONG    fsPrefix;
ULONG    fsCaseSensitive;
 } SEARCHSTRING;
```

cb (ULONG)
Structure size.

The size (in bytes) of the SEARCHSTRING structure.

pszSearch (PSZ)
Pointer.

Pointer to the search string.

ulView (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.

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.

Determines case sensitivity of the search.

TRUE The search is case sensitive.
FALSE The search is not case sensitive.

SFACTORS

Scaling factors, see DevEscape.

```
typedef struct _SFACTORS {
LONG    lx;
LONG    ly;
 } SFACTORS;
```

lx (LONG)
x-scaling factor, as an exponent of 2.

ly (LONG)
y-scaling factor, as an exponent of 2.

SHANDLE

The handle of a resource.

```
typedef USHORT SHANDLE;
```

SHORT

Signed integer in the range -32 768 through 32 767.

```
#define SHORT short
```

SIZEF

Size structure.

```
typedef struct _SIZEF {
FIXED    cx;
FIXED    cy;
 } SIZEF;
```

cx (FIXED)
Width.

cy (FIXED)
Height.

SIZEL

Size structure.

```
typedef struct _SIZEL {
LONG    cx;
LONG    cy;
 } SIZEL;
```

cx (LONG)
Width.

cy (LONG)
Height.

SLDCDATA

Slider control data structure.

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

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.

SMHSTRUCT

Send-message-hook structure.

```
typedef struct _SMHSTRUCT {
MPARAM    mp2;
MPARAM    mp1;
ULONG     msg;
HWND      hwnd;
ULONG     model;
 } SMHSTRUCT;
```

mp2 (MPARAM)
Parameter 2.

mp1 (MPARAM)
Parameter 1.

msg (ULONG)
Message identity.

hwnd (HWND)
Window handle.

model (ULONG)
Message identity.

SPLERR

Error value in the range 0 to 65 535.

```
typedef ULONG SPLERR;
```

STRUCT

Dummy data structure to be able to nest structures.

```
typedef struct _STRUCT {
```

STR8

String of 8 characters.

```
typedef CHAR STR8[8];
```

STR16

String of characters, with an implicit length, in a 16-byte field.

```
typedef CHAR STR16[16];
```

STR32

String of characters, with an implicit length, in a 32-byte field.

```
typedef CHAR STR32[32];
```

STR64

String of characters, with an implicit length, in a 64-byte field.

```
typedef CHAR STR64[64];
```

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.

```
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;
```

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.

SWBLOCK Switch-list block structure.

```
typedef struct _SWBLOCK {
ULONG      cswentry;
SWENTRY    aswentry[1];
 } SWBLOCK;
```

cswentry (ULONG)
Count of switch list entries.

aswentry[1] (SWENTRY)
Switch list entries.

SWCNTRL Switch-list control block structure.

```
typedef struct _SWCNTRL {
HWND        hwnd;
HWND        hwndIcon;
HPROGRAM    hprog;
PID         idProcess;
ULONG       idSession;
UCHAR       uchVisibility;
UCHAR       fbJump;
CHAR        szSwtitle[MAXNAMEL+1];
BYTE        bProgType;
 } SWCNTRL;
```

hwnd (HWND)
Window handle.

hwndIcon (HWND)
Window-handle icon.

hprog (HPROGRAM)
Program handle.

idProcess (PID)
Process identity.

idSession (ULONG)
Session identity.

uchVisibility (UCHAR)
Visibility:

SWL_VISIBLE	Visible in startup list
SWL_INVISIBLE	Invisible in startup list
SWL_GRAYED	Item cannot be switched to (note that it is not actually grayed in the list).

fbJump (UCHAR)
Jump indicator:

SWL_JUMPABLE	Participates in jump sequence
SWL_NOTJUMPABLE	Does not participate in jump sequence.

szSwtitle[MAXNAMEL+1] (CHAR)
Switch-list control block title (null-terminated).

bProgType (BYTE)
Program type.

SWENTRY

Switch-list entry structure.

```
typedef struct _SWENTRY {
HSWITCH    hswitch;
SWCNTRL    swctl;
 } SWENTRY;
```

hswitch (HSWITCH)
Switch-list entry handle.

swctl (SWCNTRL)
Switch-list control block structure.

SWP

Set-window-position structure.

```
typedef struct _SWP {
ULONG    fl;
LONG     cy;
LONG     cx;
LONG     y;
LONG     x;
HWND     hwndInsertBehind;
HWND     hwnd;
ULONG    ulReserved1;
ULONG    ulReserved2;
 } SWP;
```

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. This must be 0.

ulReserved2 (ULONG)
Reserved. This must be 0.

TID

Thread identity.

```
typedef ULONG TID;
```

TRACKINFO

Tracking-information structure.

```
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;
```

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.

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.

```
typedef struct _TREEITEMDESC {
HBITMAP     hbmExpanded;
HBITMAP     hbmCollapsed;
HPOINTER    hptrExpanded;
HPOINTER    hptrCollapsed;
 } TREEITEMDESC;
```

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.

UCHAR

Unsigned integer in the range 0 through 255.

```
typedef unsigned char UCHAR;
```

ULONG

Unsigned integer in the range 0 through 4 294 967 295.

```
typedef unsigned long ULONG;
```

USEITEM

The use item structure is always followed by a type-specific structure.

Type	Structure
USAGE_MEMORY	MEMORYITEM
USAGE_RECORD	RECORDITEM
USAGE_OPENVIEW	VIEWITEM

```
typedef USEITEM *USEITEM;
```

USERBUTTON

User-button data structure.

```
typedef struct _USERBUTTON {
HWND     hwnd;
HPS      hps;
ULONG    ulState;
ULONG    ulStateOld;
 } USERBUTTON;
```

hwnd (HWND)
Window handle.

hps (HPS)
Presentation-space handle.

ulState (ULONG)
New state of user button.

ulStateOld (ULONG)
Old state of user button.

USHORT

Unsigned integer in the range 0 through 65 535.

```
typedef unsigned short USHORT;
```

VIEWITEM

OPENVIEW_RECORD structure.

```
typedef VIEWITEM *VIEWITEM;
```

VIOFONTCELLSIZE

VIO cell size, see DevEscape.

```
typedef struct _VIOFONTCELLSIZE {
LONG    cx;
LONG    cy;
 } VIOFONTCELLSIZE;
```

cx (LONG)
Cell width.

cy (LONG)
Cell height.

VIOSIZECOUNT

Count of VIO cell sizes, see DevEscape.

```
typedef struct _VIOSIZECOUNT {
LONG    maxcount;
LONG    count;
 } VIOSIZECOUNT;
```

maxcount (LONG)
Maximum number of VIO cell sizes supported.

count (LONG)
Number of VIO cell sizes returned.

VOID

A data area of undefined format.

```
#define VOID void
```

VSCDATA

Structure that contains information about the value set control.

```
typedef struct _VSCDATA {
ULONG     cbSize;
USHORT    usRowCount;
USHORT    usColumnCount;
 } VSCDATA;
```

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.

```
typedef struct _VSDRAGINFO {
PDRAGINFO    pDragInfo;
USHORT       usRow;
USHORT       usColumn;
 } VSDRAGINFO;
```

pDraginfo (PDRAGINFO)
Pointer.

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.

```
typedef struct _VSDRAGINIT {
HWND      hwndVS;
LONG      x;
LONG      y;
LONG      cx;
LONG      cy;
USHORT    usRow;
USHORT    usColumn;
 } VSDRAGINIT;
```

hwndVS (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 27-8 for information about that message.

```
typedef struct _VSTEXT {
PSZ       pszItemText;
USHORT    usBufLen;
 } VSTEXT;
```

pszItemText (PSZ)
Pointer.

Pointer to a buffer to copy the string into.

usBufLen (USHORT)
Buffer size.

Size of the buffer pointed to by the *pszItemText* field.

WNDPARAMS Window parameters.

```
typedef struct _WNDPARAMS {
ULONG    ulStatus;
ULONG    ulText;
PSZ      pszText;
ULONG    ulPresParams;
PVOID    pPresParams;
ULONG    ulCtlData;
PVOID    pCtlData;
 } WNDPARAMS;
```

ulStatus (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.

ulText (ULONG)
Length of window text.

pszText (PSZ)
Window text.

ulPresParams (ULONG)
Length of presentation parameters.

pPresParams (PVOID)
Presentation parameters.

ulCtlData (ULONG)
Length of window class specific data.

pCtlData (PVOID)
Window class specific data.

*WPClock ** Pointer to an object of class WPClock.

```
typedef WPClock *WPClock *;
```

*WPCountry ** Pointer to an object of class WPCountry.

```
typedef WPCountry *WPCountry *;
```

*WPDataFile ** Pointer to an object of class WPDataFile.

```
typedef WPDataFile *WPDataFile *;
```

*WPDesktop ** Pointer to an object of class WPDesktop.

```
typedef WPDesktop *WPDesktop *;
```

*WPDisk ** Pointer to an object of class WPDisk.

```
typedef WPDisk *WPDisk *;
```

*WPFileSystem ** Pointer to an object of class WPFileSystem.

```
typedef WPFileSystem *WPFileSystem *;
```

*WPFolder ** Pointer to an object of class WPFolder.

```
typedef WPFolder *WPFolder *;
```

*WPJob ** Pointer to an object of class WPJob.

```
typedef WPJob *WPJob *;
```

*WPKeyboard ** Pointer to an object of class WPKeyboard.

```
typedef WPKeyboard *WPKeyboard *;
```

*WPMouse ** Pointer to an object of class WPMouse.

```
typedef WPMouse *WPMouse *;
```

*WPObject ** Pointer to an object of class WPObject.

```
typedef WPObject *WPObject *;
```

WPOINT Window-point structure (integer).

```
typedef struct _WPOINT {
SHORT    x;
SHORT    dummy1;
SHORT    y;
SHORT    dummy2;
 } WPOINT;
```

x (SHORT)
x-coordinate.

dummy1 (SHORT)
Reserved.

y (SHORT)
y-coordinate.

dummy2 (SHORT)
Reserved.

*WPPalette ** Pointer to an object of class WPPalette.

```
typedef WPPalette *WPPalette *;
```

*WPPrinter ** Pointer to an object of class WPPrinter.

```
typedef WPPrinter *WPPrinter *;
```

*WPProgram ** Pointer to an object of class WPProgram.

```
typedef WPProgram *WPProgram *;
```

*WPProgramGroup ** Pointer to an object of class WPProgramGroup.

```
typedef WPProgramGroup *WPProgramGroup *;
```

*WPProgramFile ** Pointer to an object of class WPProgramFile.

```
typedef WPProgramFile *WPProgramFile *;
```

*WPRootFolder ** Pointer to an object of class WPRootFolder.

```
typedef WPRootFolder *WPRootFolder *;
```

*WPShadow ** Pointer to an object of class WPShadow.

```
typedef WPShadow *WPShadow *;
```

*WPSound ** Pointer to an object of class WPSound.

```
typedef WPSound *WPSound *;
```

*WPSpooler ** Pointer to an object of class WPSpooler.

```
typedef WPSpooler *WPSpooler *;
```

*WPSRCLASSBLOCK** Save or restore class block structure.

```
typedef struct _WPSRCLASSBLOCK* {
SHORT     sClassNameLength;
USHORT    usIVarLength;
 } WPSRCLASSBLOCK*;
```

sClassNameLength (SHORT)

Length of class name, including the null terminator. This must be a short and must be at the beginning of the structure. The class name immediately follows the control block. The first instance variable control block immediately follows this.

uslVarLength (USHORT)

Length of instance variable information, including the two-byte null terminator.

*WPSystem ** — Pointer to an object of class WPSystem.

```
typedef WPSystem *WPSystem *;
```

WRECT — Window-rectangle structure (integer).

```
typedef struct _WRECT {
SHORT    xLeft;
SHORT    dummy1;
SHORT    yBottom;
SHORT    dummy2;
SHORT    xRight;
SHORT    dummy3;
SHORT    yTop;
SHORT    dummy4;
 } WRECT;
```

xLeft (SHORT)
x-coordinate of left-hand edge of rectangle.

dummy1 (SHORT)
Reserved.

yBottom (SHORT)
y-coordinate of bottom edge of rectangle.

dummy2 (SHORT)
Reserved.

xRight (SHORT)
x-coordinate of right-hand edge of rectangle.

dummy3 (SHORT)
Reserved.

yTop (SHORT)
y-coordinate of top edge of rectangle.

dummy4 (SHORT)
Reserved.

XYWINSIZE — Window position and size structure.

```
typedef struct _XYWINSIZE {
SHORT     x;
SHORT     y;
SHORT     cx;
SHORT     cy;
USHORT    fsWindow;
 } XYWINSIZE;
```

x (SHORT)
x-coordinate of window origin.

y (SHORT)
y-coordinate of window origin.

cx (SHORT)
Window width.

cy (SHORT)
Window height.

fsWindow (USHORT)
Window options.

The values may be ORed together. For example, an invisible iconic window can be created. Note that if both XYF_MINIMIZED and XYF_MAXIMIZED are specified, the window is created in a maximized state.

XYF_INVISIBLE	Create the window initially invisible.
XYF_MAXIMIZED	Show the window initially maximized.
XYF_MINIMIZED	Show the window initially iconic.
XYF_NOAUTOCLOSE	Do not close the window automatically when the VIO application terminates. This parameter is ignored unless the program is a VIO-windowed application.
XYF_NORMAL	Create the window visible, with a size and position as specified. This is the default.

Appendix B. Error Codes

This appendix lists PM errors returned by WinGetLastError in order of their error numbers. For explanations of these errors, see Appendix C, "Error Explanations" on page C-1.

Error Number	Error Constant
0x0000	PMERR_OK
0x1001	HMERR_NO_FRAME_WND_IN_CHAIN
0x1001	PMERR_INVALID_HWND
0x1002	HMERR_INVALID_ASSOC_APP_WND
0x1002	PMERR_INVALID_HMQ
0x1003	HMERR_INVALID_ASSOC_HELP_INST
0x1003	PMERR_PARAMETER_OUT_OF_RANGE
0x1004	HMERR_INVALID_DESTROY_HELP_INST
0x1004	PMERR_WINDOW_LOCK_UNDERFLOW
0x1005	HMERR_NO_HELP_INST_IN_CHAIN
0x1005	PMERR_WINDOW_LOCK_OVERFLOW
0x1006	HMERR_INVALID_HELP_INSTANCE_HDL
0x1006	PMERR_BAD_WINDOW_LOCK_COUNT
0x1007	HMERR_INVALID_QUERY_APP_WND
0x1007	PMERR_WINDOW_NOT_LOCKED
0x1008	HMERR_HELP_INST_CALLED_INVALID
0x1008	PMERR_INVALID_SELECTOR
0x1009	HMERR_HELPTABLE_UNDEFINE
0x1009	PMERR_CALL_FROM_WRONG_THREAD
0x100A	HMERR_HELP_INSTANCE_UNDEFINE
0x100A	PMERR_RESOURCE_NOT_FOUND
0x100B	HMERR_HELPITEM_NOT_FOUND
0x100B	PMERR_INVALID_STRING_PARM
0x100C	HMERR_INVALID_HELPSUBITEM_SIZE
0x100C	PMERR_INVALID_HHEAP
0x100D	HMERR_HELPSUBITEM_NOT_FOUND
0x100D	PMERR_INVALID_HEAP_POINTER
0x100E	PMERR_INVALID_HEAP_SIZE_PARM
0x100F	PMERR_INVALID_HEAP_SIZE
0x1010	PMERR_INVALID_HEAP_SIZE_WORD
0x1011	PMERR_HEAP_OUT_OF_MEMORY
0x1012	PMERR_HEAP_MAX_SIZE_REACHED
0x1013	PMERR_INVALID_HATOMTBL
0x1014	PMERR_INVALID_ATOM
0x1015	PMERR_INVALID_ATOM_NAME
0x1016	PMERR_INVALID_INTEGER_ATOM
0x1017	PMERR_ATOM_NAME_NOT_FOUND
0x1018	PMERR_QUEUE_TOO_LARGE
0x1019	PMERR_INVALID_FLAG
0x101A	PMERR_INVALID_HACCEL
0x101B	PMERR_INVALID_HPTR
0x101C	PMERR_INVALID_HENUM
0x101D	PMERR_INVALID_SRC_CODEPAGE
0x101E	PMERR_INVALID_DST_CODEPAGE
0x101F	PMERR_UNKNOWN_COMPONENT_ID
0x1020	PMERR_UNKNOWN_ERROR_CODE
0x1021	PMERR_SEVERITY_LEVELS
0x1034	PMERR_INVALID_RESOURCE_FORMAT
0x1036	PMERR_NO_MSG_QUEUE
0x1037	PMERR_WIN_DEBUGMSG
0x1038	PMERR_QUEUE_FULL
0x1039	PMERR_LIBRARY_LOAD_FAILED
0x103A	PMERR_PROCEDURE_LOAD_FAILED
0x103B	PMERR_LIBRARY_DELETE_FAILED
0x103C	PMERR_PROCEDURE_DELETE_FAILED

0x103D	PMERR_ARRAY_TOO_LARGE
0x103E	PMERR_ARRAY_TOO_SMALL
0x103F	PMERR_DATATYPE_ENTRY_BAD_INDEX
0x1040	PMERR_DATATYPE_ENTRY_CTL_BAD
0x1041	PMERR_DATATYPE_ENTRY_CTL_MISS
0x1042	PMERR_DATATYPE_ENTRY_INVALID
0x1043	PMERR_DATATYPE_ENTRY_NOT_NUM
0x1044	PMERR_DATATYPE_ENTRY_NOT_OFF
0x1045	PMERR_DATATYPE_INVALID
0x1046	PMERR_DATATYPE_NOT_UNIQUE
0x1047	PMERR_DATATYPE_TOO_LONG
0x1048	PMERR_DATATYPE_TOO_SMALL
0x1049	PMERR_DIRECTION_INVALID
0x104A	PMERR_INVALID_HAB
0x104D	PMERR_INVALID_HSTRUCT
0x104E	PMERR_LENGTH_TOO_SMALL
0x104F	PMERR_MSGID_TOO_SMALL
0x1050	PMERR_NO_HANDLE_ALLOC
0x1051	PMERR_NOT_IN_A_PM_SESSION
0x1052	PMERR_MSG_QUEUE_ALREADY_EXISTS
0x1101	PMERR_INVALID_PIB
0x1102	PMERR_INSUFF_SPACE_TO_ADD
0x1103	PMERR_INVALID_GROUP_HANDLE
0x1104	PMERR_DUPLICATE_TITLE
0x1105	PMERR_INVALID_TITLE
0x1107	PMERR_HANDLE_NOT_IN_GROUP
0x1106	PMERR_INVALID_TARGET_HANDLE
0x1108	PMERR_INVALID_PATH_STATEMENT
0x1109	PMERR_NO_PROGRAM_FOUND
0x110A	PMERR_INVALID_BUFFER_SIZE
0x110B	PMERR_BUFFER_TOO_SMALL
0x110C	PMERR_PL_INITIALISATION_FAIL
0x110D	PMERR_CANT_DESTROY_SYS_GROUP
0x110E	PMERR_INVALID_TYPE_CHANGE
0x110F	PMERR_INVALID_PROGRAM_HANDLE
0x1110	PMERR_NOT_CURRENT_PL_VERSION
0x1111	PMERR_INVALID_CIRCULAR_REF
0x1112	PMERR_MEMORY_ALLOCATION_ERR
0x1113	PMERR_MEMORY_DEALLOCATION_ERR
0x1114	PMERR_TASK_HEADER_TOO_BIG
0x1115	PMERR_INVALID_INI_FILE_HANDLE
0x1116	PMERR_MEMORY_SHARE
0x1117	PMERR_OPEN_QUEUE
0x1118	PMERR_CREATE_QUEUE
0x1119	PMERR_WRITE_QUEUE
0x111A	PMERR_READ_QUEUE
0x111B	PMERR_CALL_NOT_EXECUTED
0x111C	PMERR_UNKNOWN_APIPKT
0x111D	PMERR_INITHREAD_EXISTS
0x111E	PMERR_CREATE_THREAD
0x111F	PMERR_NO_HK_PROFILE_INSTALLED
0x1120	PMERR_INVALID_DIRECTORY
0x1121	PMERR_WILDCARD_IN_FILENAME
0x1122	PMERR_FILENAME_BUFFER_FULL
0x1123	PMERR_FILENAME_TOO_LONG
0x1124	PMERR_INI_FILE_IS_SYS_OR_USER
0x1125	PMERR_BROADCAST_PLMSG
0x1126	PMERR_190_INIT_DONE
0x1127	PMERR_HMOD_FOR_PMSHAPI
0x1128	PMERR_SET_HK_PROFILE
0x1129	PMERR_API_NOT_ALLOWED
0x112A	PMERR_INI_STILL_OPEN
0x112B	PMERR_PROGDETAILS_NOT_IN_INI

0x112C	PMERR_PIBSTRUCT_NOT_IN_INI
0x112D	PMERR_INVALID_DISKPROGDETAILS
0x112E	PMERR_PROGDETAILS_READ_FAILURE
0x112F	PMERR_PROGDETAILS_WRITE_FAILURE
0x1130	PMERR_PROGDETAILS_QSIZE_FAILURE
0x1131	PMERR_INVALID_PROGDETAILS
0x1132	PMERR_SHEPROFILEHOOK_NOT_FOUND
0x1133	PMERR_190PLCONVERTED
0x1134	PMERR_FAILED_TO_CONVERT_INI_PL
0x1135	PMERR_PMSHAPI_NOT_INITIALISED
0x1136	PMERR_INVALID_SHELL_API_HOOK_ID
0x1200	PMERR_DOS_ERROR
0x1201	PMERR_NO_SPACE
0x1202	PMERR_INVALID_SWITCH_HANDLE
0x1203	PMERR_NO_HANDLE
0x1204	PMERR_INVALID_PROCESS_ID
0x1205	PMERR_NOT_SHELL
0x1206	PMERR_INVALID_WINDOW
0x1207	PMERR_INVALID_POST_MSG
0x1208	PMERR_INVALID_PARAMETERS
0x1209	PMERR_INVALID_PROGRAM_TYPE
0x120A	PMERR_NOT_EXTENDED_FOCUS
0x120B	PMERR_INVALID_SESSION_ID
0x120C	PMERR_SMG_INVALID_ICON_FILE
0x120D	PMERR_SMG_ICON_NOT_CREATED
0x120E	PMERR_SHL_DEBUG
0x1301	PMERR_OPENING_INI_FILE
0x1302	PMERR_INI_FILE_CORRUPT
0x1303	PMERR_INVALID_PARM
0x1304	PMERR_NOT_IN_IDX
0x1305	PMERR_NO_ENTRIES_IN_GROUP
0x1306	PMERR_INI_WRITE_FAIL
0x1307	PMERR_IDX_FULL
0x1308	PMERR_INI_PROTECTED
0x1309	PMERR_MEMORY_ALLOC
0x130A	PMERR_INI_INIT_ALREADY_DONE
0x130B	PMERR_INVALID_INTEGER
0x130C	PMERR_INVALID_ASCIIZ
0x130D	PMERR_CAN_NOT_CALL_SPOOLER
0x130D	PMERR_VALIDATION_REJECTED
0x1401	PMERR_WARNING_WINDOW_NOT_KILLED
0x1402	PMERR_ERROR_INVALID_WINDOW
0x1403	PMERR_ALREADY_INITIALIZED
0x1405	PMERR_MSG_PROG_NO_MOU
0x1406	PMERR_MSG_PROG_NON_RECOV
0x1407	PMERR_WINCONV_INVALID_PATH
0x1408	PMERR_PI_NOT_INITIALISED
0x1409	PMERR_PL_NOT_INITIALISED
0x140A	PMERR_NO_TASK_MANAGER
0x140B	PMERR_SAVE_NOT_IN_PROGRESS
0x140C	PMERR_NO_STACK_SPACE
0x140d	PMERR_INVALID_COLR_FIELD
0x140e	PMERR_INVALID_COLR_VALUE
0x140f	PMERR_COLR_WRITE
0x1501	PMERR_TARGET_FILE_EXISTS
0x1502	PMERR_SOURCE_SAME_AS_TARGET
0x1503	PMERR_SOURCE_FILE_NOT_FOUND
0x1504	PMERR_INVALID_NEW_PATH
0x1505	PMERR_TARGET_FILE_NOT_FOUND
0x1506	PMERR_INVALID_DRIVE_NUMBER
0x1507	PMERR_NAME_TOO_LONG
0x1508	PMERR_NOT_ENOUGH_ROOM_ON_DISK
0x1509	PMERR_NOT_ENOUGH_MEM

0x150B	PMERR_LOG_DRV_DOES_NOT_EXIST
0x150C	PMERR_INVALID_DRIVE
0x150D	PMERR_ACCESS_DENIED
0x150E	PMERR_NO_FIRST_SLASH
0x150F	PMERR_READ_ONLY_FILE
0x151F	PMERR_GROUP_PROTECTED
0x152F	PMERR_INVALID_PROGRAM_CATEGORY
0x1530	PMERR_INVALID_APPL
0x1531	PMERR_CANNOT_START
0x1532	PMERR_STARTED_IN_BACKGROUND
0x1533	PMERR_INVALID_HAPP
0x1534	PMERR_CANNOT_STOP
0x1601	PMERR_INTERNAL_ERROR_1
0x1602	PMERR_INTERNAL_ERROR_2
0x1603	PMERR_INTERNAL_ERROR_3
0x1604	PMERR_INTERNAL_ERROR_4
0x1605	PMERR_INTERNAL_ERROR_5
0x1606	PMERR_INTERNAL_ERROR_6
0x1607	PMERR_INTERNAL_ERROR_7
0x1608	PMERR_INTERNAL_ERROR_8
0x1609	PMERR_INTERNAL_ERROR_9
0x160A	PMERR_INTERNAL_ERROR_10
0x160B	PMERR_INTERNAL_ERROR_11
0x160C	PMERR_INTERNAL_ERROR_12
0x160D	PMERR_INTERNAL_ERROR_13
0x160E	PMERR_INTERNAL_ERROR_14
0x160F	PMERR_INTERNAL_ERROR_15
0x1610	PMERR_INTERNAL_ERROR_16
0x1611	PMERR_INTERNAL_ERROR_17
0x1612	PMERR_INTERNAL_ERROR_18
0x1613	PMERR_INTERNAL_ERROR_19
0x1614	PMERR_INTERNAL_ERROR_20
0x1615	PMERR_INTERNAL_ERROR_21
0x1616	PMERR_INTERNAL_ERROR_22
0x1617	PMERR_INTERNAL_ERROR_23
0x1618	PMERR_INTERNAL_ERROR_24
0x1619	PMERR_INTERNAL_ERROR_25
0x161A	PMERR_INTERNAL_ERROR_26
0x161B	PMERR_INTERNAL_ERROR_27
0x161C	PMERR_INTERNAL_ERROR_28
0x161D	PMERR_INTERNAL_ERROR_29
0x1630	PMERR_INVALID_FREE_MESSAGE_ID
0x1641	PMERR_FUNCTION_NOT_SUPPORTED
0x1642	PMERR_INVALID_ARRAY_COUNT
0x1643	PMERR_INVALID_LENGTH
0x1644	PMERR_INVALID_BUNDLE_TYPE
0x1645	PMERR_INVALID_PARAMETER
0x1646	PMERR_INVALID_NUMBER_OF_PARMS
0x1647	PMERR_GREATER_THAN_64K
0x1648	PMERR_INVALID_PARAMETER_TYPE
0x1649	PMERR_NEGATIVE_STRCOND_DIM
0x164A	PMERR_INVALID_NUMBER_OF_TYPES
0x164B	PMERR_INCORRECT_HSTRUCT
0x164C	PMERR_INVALID_ARRAY_SIZE
0x164D	PMERR_INVALID_CONTROL_DATATYPE
0x164E	PMERR_INCOMPLETE_CONTROL_SEQU
0x164F	PMERR_INVALID_DATATYPE
0x1650	PMERR_INCORRECT_DATATYPE
0x1651	PMERR_NOT_SELF_DESCRIBING_DTYP
0x1652	PMERR_INVALID_CTRL_SEQ_INDEX
0x1653	PMERR_INVALID_TYPE_FOR_LENGTH
0x1654	PMERR_INVALID_TYPE_FOR_OFFSET
0x1655	PMERR_INVALID_TYPE_FOR_MPARAM

0x1656	PMERR_INVALID_MESSAGE_ID
0x1657	PMERR_C_LENGTH_TOO_SMALL
0x1658	PMERR_APPL_STRUCTURE_TOO_SMALL
0x1659	PMERR_INVALID_ERRORINFO_HANDLE
0x165A	PMERR_INVALID_CHARACTER_INDEX
0x2001	HMERR_INDEX_NOT_FOUND
0x2001	PMERR_ALREADY_IN_AREA
0x2002	HMERR_CONTENT_NOT_FOUND
0x2002	PMERR_ALREADY_IN_ELEMENT
0x2003	HMERR_OPEN_LIB_FILE
0x2003	PMERR_ALREADY_IN_PATH
0x2004	HMERR_READ_LIB_FILE
0x2004	PMERR_ALREADY_IN_SEG
0x2005	HMERR_CLOSE_LIB_FILE
0x2005	PMERR_AREA_INCOMPLETE
0x2006	HMERR_INVALID_LIB_FILE
0x2006	PMERR_BASE_ERROR
0x2007	HMERR_NO_MEMORY
0x2007	PMERR_BITBLT_LENGTH_EXCEEDED
0x2008	HMERR_ALLOCATE_SEGMENT
0x2008	PMERR_BITMAP_IN_USE
0x2009	HMERR_FREE_MEMORY
0x2009	PMERR_BITMAP_IS_SELECTED
0x200A	PMERR_BITMAP_NOT_FOUND
0x200B	PMERR_BITMAP_NOT_SELECTED
0x200C	PMERR_BOUNDS_OVERFLOW
0x200D	PMERR_CALLED_SEG_IS_CHAINED
0x200E	PMERR_CALLED_SEG_IS_CURRENT
0x200F	PMERR_CALLED_SEG_NOT_FOUND
0x2010	HMERR_PANEL_NOT_FOUND
0x2010	PMERR_CANNOT_DELETE_ALL_DATA
0x2011	HMERR_DATABASE_NOT_OPEN
0x2011	PMERR_CANNOT_REPLACE_ELEMENT_0
0x2012	PMERR_COL_TABLE_NOT_REALIZABLE
0x2013	HMERR_LOAD_DLL
0x2013	PMERR_COL_TABLE_NOT_REALIZED
0x2014	PMERR_COORDINATE_OVERFLOW
0x2015	PMERR_CORR_FORMAT_MISMATCH
0x2016	PMERR_DATA_TOO_LONG
0x2017	PMERR_DC_IS_ASSOCIATED
0x2018	PMERR_DESC_STRING_TRUNCATED
0x2019	PMERR_DEVICE_DRIVER_ERROR_1
0x201A	PMERR_DEVICE_DRIVER_ERROR_2
0x201B	PMERR_DEVICE_DRIVER_ERROR_3
0x201C	PMERR_DEVICE_DRIVER_ERROR_4
0x201D	PMERR_DEVICE_DRIVER_ERROR_5
0x201E	PMERR_DEVICE_DRIVER_ERROR_6
0x201F	PMERR_DEVICE_DRIVER_ERROR_7
0x2020	PMERR_DEVICE_DRIVER_ERROR_8
0x2021	PMERR_DEVICE_DRIVER_ERROR_9
0x2022	PMERR_DEVICE_DRIVER_ERROR_10
0x2023	PMERR_DEV_FUNC_NOT_INSTALLED
0x2024	PMERR_DOSOPEN_FAILURE
0x2025	PMERR_DOSREAD_FAILURE
0x2026	PMERR_DRIVER_NOT_FOUND
0x2027	PMERR_DUP_SEG
0x2028	PMERR_DYNAMIC_SEG_SEQ_ERROR
0x2029	PMERR_DYNAMIC_SEG_ZERO_INV
0x202A	PMERR_ELEMENT_INCOMPLETE
0x202B	PMERR_ESC_CODE_NOT_SUPPORTED
0x202C	PMERR_EXCEEDS_MAX_SEG_LENGTH
0x202D	PMERR_FONT_AND_MODE_MISMATCH
0x202E	PMERR_FONT_FILE_NOT_LOADED

0x202F	PMERR_FONT_NOT_LOADED
0x2030	PMERR_FONT_TOO_BIG
0x2031	PMERR_HARDWARE_INIT_FAILURE
0x2032	PMERR_HBITMAP_BUSY
0x2033	PMERR_HDC_BUSY
0x2034	PMERR_HRGN_BUSY
0x2035	PMERR_HUGE_FONTS_NOT_SUPPORTED
0x2036	PMERR_ID_HAS_NO_BITMAP
0x2037	PMERR_IMAGE_INCOMPLETE
0x2038	PMERR_INCOMPAT_COLOR_FORMAT
0x2039	PMERR_INCOMPAT_COLOR_OPTIONS
0x203A	PMERR_INCOMPATIBLE_BITMAP
0x203B	PMERR_INCOMPATIBLE_METAFILE
0x203C	PMERR_INCORRECT_DC_TYPE
0x203D	PMERR_INSUFFICIENT_DISK_SPACE
0x203E	PMERR_INSUFFICIENT_MEMORY
0x203F	PMERR_INV_ANGLE_PARM
0x2040	PMERR_INV_ARC_CONTROL
0x2041	PMERR_INV_AREA_CONTROL
0x2042	PMERR_INV_ARC_POINTS
0x2043	PMERR_INV_ATTR_MODE
0x2044	PMERR_INV_BACKGROUND_COL_ATTR
0x2045	PMERR_INV_BACKGROUND_MIX_ATTR
0x2046	PMERR_INV_BITBLT_MIX
0x2047	PMERR_INV_BITBLT_STYLE
0x2048	PMERR_INV_BITMAP_DIMENSION
0x2049	PMERR_INV_BOX_CONTROL
0x204A	PMERR_INV_BOX_ROUNDING_PARM
0x204B	PMERR_INV_CHAR_ANGLE_ATTR
0x204C	PMERR_INV_CHAR_DIRECTION_ATTR
0x204D	PMERR_INV_CHAR_MODE_ATTR
0x204E	PMERR_INV_CHAR_POS_OPTIONS
0x204F	PMERR_INV_CHAR_SET_ATTR
0x2050	PMERR_INV_CHAR_SHEAR_ATTR
0x2051	PMERR_INV_CLIP_PATH_OPTIONS
0x2052	PMERR_INV_CODEPAGE
0x2053	PMERR_INV_COLOR_ATTR
0x2054	PMERR_INV_COLOR_DATA
0x2055	PMERR_INV_COLOR_FORMAT
0x2056	PMERR_INV_COLOR_INDEX
0x2057	PMERR_INV_COLOR_OPTIONS
0x2058	PMERR_INV_COLOR_START_INDEX
0x2059	PMERR_INV_COORD_OFFSET
0x205A	PMERR_INV_COORD_SPACE
0x205B	PMERR_INV_COORDINATE
0x205C	PMERR_INV_CORRELATE_DEPTH
0x205D	PMERR_INV_CORRELATE_TYPE
0x205E	PMERR_INV_CURSOR_BITMAP
0x205F	PMERR_INV_DC_DATA
0x2060	PMERR_INV_DC_TYPE
0x2061	PMERR_INV_DEVICE_NAME
0x2062	PMERR_INV_DEV_MODES_OPTIONS
0x2063	PMERR_INV_DRAW_CONTROL
0x2064	PMERR_INV_DRAW_VALUE
0x2065	PMERR_INV_DRAWING_MODE
0x2066	PMERR_INV_DRIVER_DATA
0x2067	PMERR_INV_DRIVER_NAME
0x2068	PMERR_INV_DRAW_BORDER_OPTION
0x2069	PMERR_INV_EDIT_MODE
0x206A	PMERR_INV_ELEMENT_OFFSET
0x206B	PMERR_INV_ELEMENT_POINTER
0x206C	PMERR_INV_END_PATH_OPTIONS
0x206D	PMERR_INV_ESC_CODE

0x206E	PMERR_INV_ESCAPE_DATA
0x206F	PMERR_INV_EXTENDED_LCID
0x2070	PMERR_INV_FILL_PATH_OPTIONS
0x2071	PMERR_INV_FIRST_CHAR
0x2072	PMERR_INV_FONT_ATTRS
0x2073	PMERR_INV_FONT_FILE_DATA
0x2074	PMERR_INV_FOR_THIS_DC_TYPE
0x2075	PMERR_INV_FORMAT_CONTROL
0x2076	PMERR_INV_FORMS_CODE
0x2077	PMERR_INV_FONTDEF
0x2078	PMERR_INV_GEOM_LINE_WIDTH_ATTR
0x2079	PMERR_INV_GETDATA_CONTROL
0x207A	PMERR_INV_GRAPHICS_FIELD
0x207B	PMERR_INV_HBITMAP
0x207C	PMERR_INV_HDC
0x207D	PMERR_INV_HJOURNAL
0x207E	PMERR_INV_HMF
0x207F	PMERR_INV_HPS
0x2080	PMERR_INV_HRGN
0x2081	PMERR_INV_ID
0x2082	PMERR_INV_IMAGE_DATA_LENGTH
0x2083	PMERR_INV_IMAGE_DIMENSION
0x2084	PMERR_INV_IMAGE_FORMAT
0x2085	PMERR_INV_IN_AREA
0x2086	PMERR_INV_IN_CALLED_SEG
0x2087	PMERR_INV_IN_CURRENT_EDIT_MODE
0x2088	PMERR_INV_IN_DRAW_MODE
0x2089	PMERR_INV_IN_ELEMENT
0x208A	PMERR_INV_IN_IMAGE
0x208B	PMERR_INV_IN_PATH
0x208C	PMERR_INV_IN_RETAIN_MODE
0x208D	PMERR_INV_IN_SEG
0x208E	PMERR_INV_IN_VECTOR_SYMBOL
0x208F	PMERR_INV_INFO_TABLE
0x2090	PMERR_INV_JOURNAL_OPTION
0x2091	PMERR_INV_KERNING_FLAGS
0x2092	PMERR_INV_LENGTH_OR_COUNT
0x2093	PMERR_INV_LINE_END_ATTR
0x2094	PMERR_INV_LINE_JOIN_ATTR
0x2095	PMERR_INV_LINE_TYPE_ATTR
0x2096	PMERR_INV_LINE_WIDTH_ATTR
0x2097	PMERR_INV_LOGICAL_ADDRESS
0x2098	PMERR_INV_MARKER_BOX_ATTR
0x2099	PMERR_INV_MARKER_SET_ATTR
0x209A	PMERR_INV_MARKER_SYMBOL_ATTR
0x209B	PMERR_INV_MATRIX_ELEMENT
0x209C	PMERR_INV_MAX_HITS
0x209D	PMERR_INV_METAFILE
0x209E	PMERR_INV_METAFILE_LENGTH
0x209F	PMERR_INV_METAFILE_OFFSET
0x20A0	PMERR_INV_MICROPS_DRAW_CONTROL
0x20A1	PMERR_INV_MICROPS_FUNCTION
0x20A2	PMERR_INV_MICROPS_ORDER
0x20A3	PMERR_INV_MIX_ATTR
0x20A4	PMERR_INV_MODE_FOR_OPEN_DYN
0x20A5	PMERR_INV_MODE_FOR_REOPEN_SEG
0x20A6	PMERR_INV_MODIFY_PATH_MODE
0x20A7	PMERR_INV_MULTIPLIER
0x20A8	PMERR_INV_NESTED_FIGURES
0x20A9	PMERR_INV_OR_INCOMPAT_OPTIONS
0x20AA	PMERR_INV_ORDER_LENGTH
0x20AB	PMERR_INV_ORDERING_PARM
0x20AC	PMERR_INV_OUTSIDE_DRAW_MODE

0x20AD	PMERR_INV_PAGE_VIEWPORT
0x20AE	PMERR_INV_PATH_ID
0x20AF	PMERR_INV_PATH_MODE
0x20B0	PMERR_INV_PATTERN_ATTR
0x20B1	PMERR_INV_PATTERN_REF_PT_ATTR
0x20B2	PMERR_INV_PATTERN_SET_ATTR
0x20B3	PMERR_INV_PATTERN_SET_FONT
0x20B4	PMERR_INV_PICK_APERTURE_OPTION
0x20B5	PMERR_INV_PICK_APERTURE_POSN
0x20B6	PMERR_INV_PICK_APERTURE_SIZE
0x20B7	PMERR_INV_PICK_NUMBER
0x20B8	PMERR_INV_PLAY_METAFILE_OPTION
0x20B9	PMERR_INV_PRIMITIVE_TYPE
0x20BA	PMERR_INV_PS_SIZE
0x20BB	PMERR_INV_PUTDATA_FORMAT
0x20BC	PMERR_INV_QUERY_ELEMENT_NO
0x20BD	PMERR_INV_RECT
0x20BE	PMERR_INV_REGION_CONTROL
0x20BF	PMERR_INV_REGION_MIX_MODE
0x20C0	PMERR_INV_REPLACE_MODE_FUNC
0x20C1	PMERR_INV_RESERVED_FIELD
0x20C2	PMERR_INV_RESET_OPTIONS
0x20C3	PMERR_INV_RGBCOLOR
0x20C4	PMERR_INV_SCAN_START
0x20C5	PMERR_INV_SEG_ATTR
0x20C6	PMERR_INV_SEG_ATTR_VALUE
0x20C7	PMERR_INV_SEG_CH_LENGTH
0x20C8	PMERR_INV_SEG_NAME
0x20C9	PMERR_INV_SEG_OFFSET
0x20CA	PMERR_INV_SETID
0x20CB	PMERR_INV_SETID_TYPE
0x20CC	PMERR_INV_SET_VIEWPORT_OPTION
0x20CD	PMERR_INV_SHARPNESS_PARM
0x20CE	PMERR_INV_SOURCE_OFFSET
0x20CF	PMERR_INV_STOP_DRAW_VALUE
0x20D0	PMERR_INV_TRANSFORM_TYPE
0x20D1	PMERR_INV_USAGE_PARM
0x20D2	PMERR_INV_VIEWING_LIMITS
0x20D3	PMERR_JFILE_BUSY
0x20D4	PMERR_JNL_FUNC_DATA_TOO_LONG
0x20D5	PMERR_KERNING_NOT_SUPPORTED
0x20D6	PMERR_LABEL_NOT_FOUND
0x20D7	PMERR_MATRIX_OVERFLOW
0x20D8	PMERR_METAFILE_INTERNAL_ERROR
0x20D9	PMERR_METAFILE_IN_USE
0x20DA	PMERR_METAFILE_LIMIT_EXCEEDED
0x20DB	PMERR_NAME_STACK_FULL
0x20DC	PMERR_NOT_CREATED_BY_DEVOPENDC
0x20DD	PMERR_NOT_IN_AREA
0x20DE	PMERR_NOT_IN_DRAW_MODE
0x20DF	PMERR_NOT_IN_ELEMENT
0x20E0	PMERR_NOT_IN_IMAGE
0x20E1	PMERR_NOT_IN_PATH
0x20E2	PMERR_NOT_IN_RETAIN_MODE
0x20E3	PMERR_NOT_IN_SEG
0x20E4	PMERR_NO_BITMAP_SELECTED
0x20E5	PMERR_NO_CURRENT_ELEMENT
0x20E6	PMERR_NO_CURRENT_SEG
0x20E7	PMERR_NO_METAFILE_RECORD_HANDLE
0x20E8	PMERR_ORDER_TOO_BIG
0x20E9	PMERR_OTHER_SET_ID_REFS
0x20EA	PMERR_OVERRAN_SEG
0x20EB	PMERR_OWN_SET_ID_REFS

0x20EC	PMERR_PATH_INCOMPLETE
0x20ED	PMERR_PATH_LIMIT_EXCEEDED
0x20EE	PMERR_PATH_UNKNOWN
0x20EF	PMERR_PEL_IS_CLIPPED
0x20F0	PMERR_PEL_NOT_AVAILABLE
0x20F1	PMERR_PRIMITIVE_STACK_EMPTY
0x20F2	PMERR_PROLOG_ERROR
0x20F3	PMERR_PROLOG_SEG_ATTR_NOT_SET
0x20F4	PMERR_PS_BUSY
0x20F5	PMERR_PS_IS_ASSOCIATED
0x20F6	PMERR_RAM_JNL_FILE_TOO_SMALL
0x20F7	PMERR_REALIZE_NOT_SUPPORTED
0x20F8	PMERR_REGION_IS_CLIP_REGION
0x20F9	PMERR_RESOURCE_DEPLETION
0x20FA	PMERR_SEG_AND_REFSEG_ARE_SAME
0x20FB	PMERR_SEG_CALL_RECURSIVE
0x20FC	PMERR_SEG_CALL_STACK_EMPTY
0x20FD	PMERR_SEG_CALL_STACK_FULL
0x20FE	PMERR_SEG_IS_CURRENT
0x20FF	PMERR_SEG_NOT_CHAINED
0x2100	PMERR_SEG_NOT_FOUND
0x2101	PMERR_SEG_STORE_LIMIT_EXCEEDED
0x2102	PMERR_SETID_IN_USE
0x2103	PMERR_SETID_NOT_FOUND
0x2104	PMERR_STARTDOC_NOT_ISSUED
0x2105	PMERR_STOP_DRAW_OCCURRED
0x2106	PMERR_TOO_MANY_METAFILES_IN_USE
0x2107	PMERR_TRUNCATED_ORDER
0x2108	PMERR_UNCHAINED_SEG_ZERO_INV
0x2109	PMERR_UNSUPPORTED_ATTR
0x210A	PMERR_UNSUPPORTED_ATTR_VALUE
0x210B	PMERR_ENDDOC_NOT_ISSUED
0x210C	PMERR_PS_NOT_ASSOCIATED
0x210D	PMERR_INV_FLOOD_FILL_OPTIONS
0x210E	PMERR_INV_FACENAME
0x210F	PMERR_PALETTE_SELECTED
0x2110	PMERR_NO_PALETTE_SELECTED
0x2111	PMERR_INV_HPAL
0x2112	PMERR_PALETTE_BUSY
0x2113	PMERR_START_POINT_CLIPPED
0x2114	PMERR_NO_FILL
0x2115	PMERR_INV_FACENAMEDESC
0x2116	PMERR_INV_BITMAP_DATA
0x2117	PMERR_INV_CHAR_ALIGN_ATTR
0x2118	PMERR_INV_HFONT
0x2119	PMERR_HFONT_IS_SELECTED
0x2120	PMERR_RASTER_FONT
0x3001	HMERR_DDF_MEMORY
0x3002	HMERR_DDF_ALIGN_TYPE
0x3003	HMERR_DDF_BACKCOLOR
0x3004	HMERR_DDF_FORECOLOR
0x3005	HMERR_DDF_FONTSTYLE
0x3006	HMERR_DDF_REFTYPE
0x3007	HMERR_DDF_LIST_UNCLOSED
0x3008	HMERR_DDF_LIST_UNINITIALIZED
0x3009	HMERR_DDF_LIST_BREAKTYPE
0x300A	HMERR_DDF_LIST_SPACING
0x300B	HMERR_DDF_HINSTANCE
0x300C	HMERR_DDF_EXCEED_MAX_LENGTH
0x300D	HMERR_DDF_EXCEED_MAX_INC
0x300E	HMERR_DDF_INVALID_DDF
0x300F	HMERR_DDF_FORMAT_TYPE
0x3010	HMERR_DDF_INVALID_PARM

0x3011	HMERR_DDF_INVALID_FONT
0x3012	HMERR_DDF_SEVERE
0x4001	PMERR_SPL_DRIVER_ERROR
0x4002	PMERR_SPL_DEVICE_ERROR
0x4003	PMERR_SPL_DEVICE_NOT_INSTALLED
0x4004	PMERR_SPL_QUEUE_ERROR
0x4005	PMERR_SPL_INV_HSPL
0x4006	PMERR_SPL_NO_DISK_SPACE
0x4007	PMERR_SPL_NO_MEMORY
0x4008	PMERR_SPL_PRINT_ABORT
0x4009	PMERR_SPL_SPOOLER_NOT_INSTALLED
0x400A	PMERR_SPL_INV_FORMS_CODE
0x400B	PMERR_SPL_INV_PRIORITY
0x400C	PMERR_SPL_NO_FREE_JOB_ID
0x400D	PMERR_SPL_NO_DATA
0x400E	PMERR_SPL_INV_TOKEN
0x400F	PMERR_SPL_INV_DATATYPE
0x4010	PMERR_SPL_PROCESSOR_ERROR
0x4011	PMERR_SPL_INV_JOB_ID
0x4012	PMERR_SPL_JOB_NOT_PRINTING
0x4013	PMERR_SPL_JOB_PRINTING
0x4014	PMERR_SPL_QUEUE_ALREADY_EXISTS
0x4015	PMERR_SPL_INV_QUEUE_NAME
0x4016	PMERR_SPL_QUEUE_NOT_EMPTY
0x4017	PMERR_SPL_DEVICE_ALREADY_EXISTS
0x4018	PMERR_SPL_DEVICE_LIMIT_REACHED
0x4019	PMERR_SPL_STATUS_STRING_TRUNC
0x401A	PMERR_SPL_INV_LENGTH_OR_COUNT
0x401B	PMERR_SPL_FILE_NOT_FOUND
0x401C	PMERR_SPL_CANNOT_OPEN_FILE
0x401D	PMERR_SPL_DRIVER_NOT_INSTALLED
0x401E	PMERR_SPL_INV_PROCESSOR_DATTYPE
0x401F	PMERR_SPL_INV_DRIVER_DATATYPE
0x4020	PMERR_SPL_PROCESSOR_NOT_INST
0x4021	PMERR_SPL_NO_SUCH_LOG_ADDRESS
0x4022	PMERR_SPL_PRINTER_NOT_FOUND
0x4023	PMERR_SPL_DD_NOT_FOUND
0x4024	PMERR_SPL_QUEUE_NOT_FOUND
0x4025	PMERR_SPL_MANY_QUEUES_ASSOC
0x4026	PMERR_SPL_NO_QUEUES_ASSOCIATED
0x4027	PMERR_SPL_INI_FILE_ERROR
0x4028	PMERR_SPL_NO_DEFAULT_QUEUE
0x4029	PMERR_SPL_NO_CURRENT_FORMS_CODE
0x402A	PMERR_SPL_NOT_AUTHORISED
0x402B	PMERR_SPL_TEMP_NETWORK_ERROR
0x402C	PMERR_SPL_HARD_NETWORK_ERROR
0x402D	PMERR_DEL_NOT_ALLOWED
0x402E	PMERR_CANNOT_DEL_QP_REF
0x402F	PMERR_CANNOT_DEL_QNAME_REF
0x4030	PMERR_CANNOT_DEL_PRINTER_DD_REF
0x4031	PMERR_CANNOT_DEL_PRN_NAME_REF
0x4032	PMERR_CANNOT_DEL_PRN_ADDR_REF
0x4033	PMERR_SPOOLER_QP_NOT_DEFINED
0x4034	PMERR_PRN_NAME_NOT_DEFINED
0x4035	PMERR_PRN_ADDR_NOT_DEFINED
0x4036	PMERR_PRINTER_DD_NOT_DEFINED
0x4037	PMERR_PRINTER_QUEUE_NOT_DEFINED
0x4038	PMERR_PRN_ADDR_IN_USE
0x4039	PMERR_SPL_TOO_MANY_OPEN_FILES
0x403A	PMERR_SPL_CP_NOT_REQD
0x4040	PMERR_UNABLE_TO_CLOSE_DEVICE
0x4FA1	PMERR_SPL_ERROR_1
0x4FA2	PMERR_SPL_ERROR_2

0x4FA3	PMERR_SPL_ERROR_3
0x4FA4	PMERR_SPL_ERROR_4
0x4FA5	PMERR_SPL_ERROR_5
0x4FA6	PMERR_SPL_ERROR_6
0x4FA7	PMERR_SPL_ERROR_7
0x4FA8	PMERR_SPL_ERROR_8
0x4FA9	PMERR_SPL_ERROR_9
0x4FAA	PMERR_SPL_ERROR_10
0x4FAB	PMERR_SPL_ERROR_11
0x4FAC	PMERR_SPL_ERROR_12
0x4FAD	PMERR_SPL_ERROR_13
0x4FAE	PMERR_SPL_ERROR_14
0x4FAF	PMERR_SPL_ERROR_15
0x4FB0	PMERR_SPL_ERROR_16
0x4FB1	PMERR_SPL_ERROR_17
0x4FB2	PMERR_SPL_ERROR_18
0x4FB3	PMERR_SPL_ERROR_19
0x4FB4	PMERR_SPL_ERROR_20
0x4FB5	PMERR_SPL_ERROR_21
0x4FB6	PMERR_SPL_ERROR_22
0x4FB7	PMERR_SPL_ERROR_23
0x4FB8	PMERR_SPL_ERROR_24
0x4FB9	PMERR_SPL_ERROR_25
0x4FBA	PMERR_SPL_ERROR_26
0x4FBB	PMERR_SPL_ERROR_27
0x4FBC	PMERR_SPL_ERROR_28
0x4FBD	PMERR_SPL_ERROR_29
0x4FBE	PMERR_SPL_ERROR_30
0x4FBF	PMERR_SPL_ERROR_31
0x4FC0	PMERR_SPL_ERROR_32
0x4FC1	PMERR_SPL_ERROR_33
0x4FC2	PMERR_SPL_ERROR_34
0x4FC3	PMERR_SPL_ERROR_35
0x4FC4	PMERR_SPL_ERROR_36
0x4FC5	PMERR_SPL_ERROR_37
0x4FC6	PMERR_SPL_ERROR_38
0x4FC7	PMERR_SPL_ERROR_39
0x4FC8	PMERR_SPL_ERROR_40
0x4FC9	PMERR_SPLMSGBOX_INFO_CAPTION
0x4FCA	PMERR_SPLMSGBOX_WARNING_CAPTION
0x4FCB	PMERR_SPLMSGBOX_ERROR_CAPTION
0x4FCC	PMERR_SPLMSGBOX_SEVERE_CAPTION
0x4FCD	PMERR_SPLMSGBOX_JOB_DETAILS
0x4FCE	PMERR_SPLMSGBOX_ERROR_ACTION
0x4FCF	PMERR_SPLMSGBOX_SEVERE_ACTION
0x4FD0	PMERR_SPLMSGBOX_BIT_0_TEXT
0x4FD1	PMERR_SPLMSGBOX_BIT_1_TEXT
0x4FD2	PMERR_SPLMSGBOX_BIT_2_TEXT
0x4FD3	PMERR_SPLMSGBOX_BIT_3_TEXT
0x4FD4	PMERR_SPLMSGBOX_BIT_4_TEXT
0x4FD5	PMERR_SPLMSGBOX_BIT_5_TEXT
0x4FD6	PMERR_SPLMSGBOX_BIT_15_TEXT
0x4FD7	PMERR_SPL_NOPATHBUFFER
0x4FD8	PMERR_SPL_ALREADY_INITIALISED
0x4FD9	PMERR_SPL_ERROR
0x5001	PMERR_INV_TYPE
0x5002	PMERR_INV_CONV
0x5003	PMERR_INV_SEGLEN
0x5004	PMERR_DUP_SEGNAME
0x5005	PMERR_INV_XFORM

0x5006	PMERR_INV_VIEWLIM
0x5007	PMERR_INV_3DCOORD
0x5008	PMERR_SMB_OVFLOW
0x5009	PMERR_SEG_OVFLOW
0x5010	PMERR_PIC_DUP_FILENAME

Appendix C. Error Explanations

This appendix gives an explanation for each PM error. The errors are listed in alphabetic order. The number associated with each error is given in Appendix B, "Error Codes" on page B-1.

Error Constant	Explanation
HMERR_ALLOCATE_SEGMENT	Unable to allocate a segment of memory for memory allocation requests from the help manager.
HMERR_CLOSE_LIB_FILE	The library file cannot be closed.
HMERR_CONTENT_NOT_FOUND	The library file does not have any content.
HMERR_DATABASE_NOT_OPEN	Unable to read the unopened database.
HMERR_DDF_ALIGN_TYPE	The alignment type is not valid.
HMERR_DDF_BACKCOLOR	The background color is not valid.
HMERR_DDF_EXCEED_MAX_INC	The value specified to increment DDF memory is too large.
HMERR_DDF_EXCEED_MAX_LENGTH	The amount of data is too large for the DDF buffer.
HMERR_DDF_FONTSTYLE	The font style is not valid.
HMERR_DDF_FORECOLOR	The foreground color is not valid.
HMERR_DDF_FORMAT_TYPE	The format type specified is invalid.
HMERR_DDF_HINSTANCE	The DDF instance is invalid.
HMERR_DDF_INVALID_DDF	The DDF handle is invalid.
HMERR_DDF_INVALID_FONT	The font value specified is invalid.
HMERR_DDF_INVALID_PARM	One of the DDF parameters specified is invalid.
HMERR_DDF_LIST_BREAKTYPE	The value of BreakType is not valid.
HMERR_DDF_LIST_SPACING	The value for Spacing is not valid.
HMERR_DDF_LIST_UNCLOSED	An attempt was made to nest a list.
HMERR_DDF_LIST_UNINITIALIZED	No definition list has been initialized by DdfBeginList.
HMERR_DDF_MEMORY	Not enough memory is available.
HMERR_DDF_REFTYPE	The reference type is not valid.
HMERR_DDF_SEVERE	Internal error detected by the Help Manager.
HMERR_FREE_MEMORY	Unable to free allocated memory.
HMERR_HELP_INST_CALLED_INVALID	The handle of the instance specified on a call to the help manager does not have the class name of a help manager instance.
HMERR_HELP_INSTANCE_UNDEFINE	The help instance handle specified is invalid.
HMERR_HELPITEM_NOT_FOUND	Context-sensitive help was requested but the ID of the main help item specified was not found in the help table.
HMERR_HELPSUBITEM_NOT_FOUND	Context-sensitive help was requested but the ID of the help item specified was not found in the help subtable.
HMERR_HELPTABLE_UNDEFINE	The application did not provide a help table for context-sensitive help.
HMERR_INDEX_NOT_FOUND	The index is not in the library file.

HMERR_INVALID_ASSOC_APP_WND	The application window handle specified on the WinAssociateHelpInstance function is not a valid window handle.
HMERR_INVALID_ASSOC_HELP_INST	The help instance handle specified on the WinAssociateHelpInstance function is not a valid window handle.
HMERR_INVALID_DESTROY_HELP_INST	The window handle specified as the help instance to destroy is not of the help instance class.
HMERR_INVALID_HELP_INSTANCE_HDL	The handle specified to be a help instance does not have the class name of a help manager instance.
HMERR_INVALID_HELPSUBITEM_SIZE	The help subtable item size is less than 2.
HMERR_INVALID_LIB_FILE	Improper library file provided.
HMERR_INVALID_QUERY_APP_WND	The application window specified on a WinQueryHelpInstance function is not a valid window handle.
HMERR_LOAD_DLL	Unable to load resource data link library.
HMERR_NO_FRAME_WND_IN_CHAIN	There is no frame window in the window chain from which to find or set the associated help instance.
HMERR_NO_HELP_INST_IN_CHAIN	The parent or owner chain of the application window specified does not have an associated help instance.
HMERR_NO_MEMORY	Unable to allocate the requested amount of memory.
HMERR_OPEN_LIB_FILE	The library file cannot be opened.
HMERR_PANEL_NOT_FOUND	Unable to find the requested help panel.
HMERR_READ_LIB_FILE	The library file cannot be read.
PMERR_ACCESS_DENIED	The memory block was not allocated properly.
PMERR_ALREADY_IN_AREA	An attempt was made to begin a new area while an existing area bracket was already open.
PMERR_ALREADY_IN_ELEMENT	An attempt was made to begin a new element while an existing element bracket was already open.
PMERR_ALREADY_IN_PATH	An attempt was made to begin a new path while an existing path bracket was already open.
PMERR_ALREADY_IN_SEG	An attempt was made to open a new segment while an existing segment bracket was already open.
PMERR_APPL_STRUCTURE_TOO_SMALL	The application buffer length is less than the total length required for the (application) component types.
PMERR_ARRAY_TOO_SMALL	The array specified was too small.
PMERR_AREA_INCOMPLETE	Either: • A segment has been opened, closed, or drawn. • GpiAssociate was issued while an area bracket was open. • A drawn segment has opened an area bracket and ended without closing it.
PMERR_ARRAY_TOO_LARGE	More than 4 bytes was attempted to be inserted or extracted.
PMERR_ATOM_NAME_NOT_FOUND	The specified atom name is not in the atom table.

PMERR_BASE_ERROR	An OS/2 base error has occurred. The base error code can be accessed using the OffBinaryData field of the ERRINFO structure returned by WinGetErrorInfo.
PMERR_BITMAP_IN_USE	An attempt was made either to set a bit map into a device context using GpiSetBitmap while it was already selected into an existing device context, or to tag a bit map with a local pattern set identifier (setid) using GpiSetBitmapId while it was already tagged with an existing setid.
PMERR_BITMAP_IS_SELECTED	An attempt was made to delete a bit map while it was selected into a device context.
PMERR_BITMAP_NOT_FOUND	A attempt was made to perform a bit-map operation on a bit map that did not exist.
PMERR_BITMAP_NOT_SELECTED	A attempt was made to perform an operation on presentation space associated with a memory device context that had no selected bit map.
PMERR_BOUNDS_OVERFLOW	An internal overflow error occurred during boundary data accumulation. This can occur if coordinates or matrix transformation elements (or both) are invalid or too large.
PMERR_BUFFER_TOO_SMALL	The supplied buffer was not large enough for the data to be returned.
PMERR_C_LENGTH_TOO_SMALL	The maximum length of the C structure is less than the total length required for the (C) component types.
PMERR_CALLED_SEG_IS_CHAINED	An attempt was made to call a segment that has a chained attribute set.
PMERR_CAN_NOT_CALL_SPOOLER	An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.
PMERR_CANNOT_DEL_PRINTER_DD_REF	Presentation Manager device driver deletion not possible due to a reference.
PMERR_CANNOT_DEL_PRN_ADDR_REF	Printer port deletion not possible due to a reference.
PMERR_CANNOT_DEL_PRN_NAME_REF	Printer deletion not possible due to a reference.
PMERR_CANNOT_DEL_QNAME_REF	Spooler queue deletion not possible due to a reference.
PMERR_CANNOT_DEL_QP_REF	Spooler queue processor deletion not possible due to a reference.
PMERR_CANNOT_STOP	The session cannot be stopped.
PMERR_CALLED_SEG_IS_CURRENT	An attempt was made to call a segment that is currently open.
PMERR_CALLED_SEG_NOT_FOUND	An attempt was made to call a segment that did not exist.
PMERR_COL_TABLE_NOT_REALIZABLE	An attempt was made to realize a color table that is not realizable.
PMERR_COL_TABLE_NOT_REALIZED	An attempt was made to realize a color table on a device driver that does not support this function.
PMERR_COORDINATE_OVERFLOW	An internal coordinate overflow error occurred. This can occur if coordinates or matrix transformation elements (or both) are invalid or too large.

PMERR_DATA_TOO_LONG	An attempt was made to transfer more than the maximum permitted amount of data (64512 bytes) using GpiPutData, GpiGetData, or GpiElement.
PMERR_DATATYPE_ENTRY_BAD_INDEX	An invalid datatype entry index was specified.
PMERR_DATATYPE_ENTRY_CTL_BAD	An invalid datatype entry control was specified.
PMERR_DATATYPE_ENTRY_CTL_MISS	The datatype entry control was missing.
PMERR_DATATYPE_ENTRY_NOT_NUM	The datatype entry specified was not numerical.
PMERR_DATATYPE_ENTRY_NOT_OFF	The datatype entry specified was not an offset.
PMERR_DATATYPE_INVALID	An invalid datatype was specified.
PMERR_DATATYPE_NOT_UNIQUE	An attempt to register a datatype failed because it is not unique.
PMERR_DATATYPE_TOO_LONG	The datatype specified was too long.
PMERR_DATATYPE_TOO_SMALL	The datatype specified was too small.
PMERR_DC_IS_ASSOCIATED	An attempt was made to associate a presentation space with a device context that was already associated or to destroy a device context that was associated.
PMERR_DEL_NOT_ALLOWED	Deletion not possible.
PMERR_DESC_STRING_TRUNCATED	An attempt was made to supply a description string with GpiBeginElement that was greater then the permitted maximum length (251 characters). The string was truncated.
PMERR_DEV_FUNC_NOT_INSTALLED	The function requested is not supported by the presentation driver.
PMERR_DEVICE_DRIVER_ERROR_1	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_2	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_3	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_4	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_5	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_6	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_7	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_8	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_9	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_10	Miscellaneous error available for use by user written device drivers.
PMERR_DOS_ERROR	A DOS call returned an error.
PMERR_DOSOPEN_FAILURE	A DosOpen call made during GpiLoadMetaFile or GpiSaveMetaFile gave a good return code but the file was not opened successfully.

PMERR_DOSREAD_FAILURE	A DosRead call made during GpiLoadMetaFile gave a good return code. However, it failed to read any more bytes although the file length indicated that there were more to be read.
PMERR_DRIVER_NOT_FOUND	The device driver specified with DevPostDeviceModes was not found.
PMERR_DUP_SEG	During GpiPlayMetaFile, while the actual drawing mode was **draw-and-retain** or **retain**, a metafile segment to be stored in the presentation space was found to have the same segment identifier as an existing segment.
PMERR_DUP_SEGNAME	A called segment has a name that has already been used by another called segment in the input PIF.
PMERR_DUPLICATE_TITLE	The program title specified in the PIBSTRUCT already exists within the same group.
PMERR_DYNAMIC_SEG_SEQ_ERROR	During removal of dynamic segments while processing GpiDrawChain, GpiDrawFrom, or GpiDrawSegment, the internal state indicated that dynamic segment data was still visible after all chained dynamic segments had been processed. This can occur if segments drawn dynamically (including called segments) are modified or removed from the chain while visible.
PMERR_DYNAMIC_SEG_ZERO_INV	An attempt was been made to open a dynamic segment with a segment identifier of zero.
PMERR_ENDDOC_NOT_ISSUED	A request to close the spooled output without first issuing a an ENDDOC was attempted.
PMERR_ESC_CODE_NOT_SUPPORTED	The code specified with DevEscape is not supported by the target device driver.
PMERR_EXCEEDS_MAX_SEG_LENGTH	During metafile creation or generation of retained graphics the system has exceeded maximum segment size.
PMERR_FONT_AND_MODE_MISMATCH	An attempt was made to draw characters with a character mode and character set that are incompatible. For example, the character specifies an image/raster font when the mode calls for a vector/outline font.
PMERR_FONT_FILE_NOT_LOADED	An attempt was made to unload a font file that was not loaded.
PMERR_FONT_NOT_LOADED	An attempt was made to create a font that was not loaded.
PMERR_FUNCTION_NOT_SUPPORTED	The function is not supported.
PMERR_GREATER_THAN_64K	A data item or array dimension is greater than 65 535.
PMERR_HBITMAP_BUSY	An internal bit map busy error was detected. The bit map was locked by one thread during an attempt to access it from another thread.
PMERR_HDC_BUSY	An internal device context busy error was detected. The device context was locked by one thread during an attempt to access it from another thread.
PMERR_HEAP_MAX_SIZE_REACHED	The heap has reached its maximum size (64KB), and cannot be increased.
PMERR_HEAP_OUT_OF_MEMORY	An attempt to increase the size of the heap failed.

PMERR_HFONT_IS_SELECTED
An attempt has been made to either change the owner of a font, or delete when it is currently selected.

PMERR_HRGN_BUSY
An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.

PMERR_HUGE_FONTS_NOT_SUPPORTED
An attempt was made using GpiSetCharSet, GpiSetPatternSet, GpiSetMarkerSet, or GpiSetAttrs to select a font that is larger than the maximum size (64Kb) supported by the target device driver.

PMERR_ID_HAS_NO_BITMAP
No bit map was tagged with the setid specified on a GpiQueryBitmapHandle function.

PMERR_IMAGE_INCOMPLETE
A drawn segment has opened an image bracket and ended without closing it.

PMERR_INCOMPATIBLE_BITMAP
An attempt was made to select a bit map or perform a BitBlt operation on a device context that was incompatible with the format of the bit map.

PMERR_INCOMPATIBLE_METAFILE
An attempt was made to associate a presentation space and a metafile device context with incompatible page units, size or coordinate format; or to play a metafile using the RES_RESET option (to reset the presentation space) to a presentation space that is itself associated with a metafile device context.

PMERR_INCOMPLETE_CONTROL_SEQ
A control data type sequence is incomplete.

PMERR_INCORRECT_DATATYPE
A data type is specified which is incorrect for this function.

PMERR_INCORRECT_DC_TYPE
An attempt was made to perform a bit-map operation on a presentation space associated with a device context of a type that is unable to support bit-map operations.

PMERR_INCORRECT_HSTRUCT
A structure handle is non-NULL, and is invalid for one of the following reasons:

- It is not the handle of a data structure.
- It is the handle of an ERRINFO structure which should not be used on this call.
- A handle block returned by the bindings to the application has been used for an in-line structure handle.

PMERR_INI_FILE_IS_SYS_OR_USER
User or system initialization file cannot be closed.

PMERR_INSUFF_SPACE_TO_ADD
The initialization file could not be extended to add the required program or group.

PMERR_INSUFFICIENT_DISK_SPACE
The operation terminated through insufficient disk space.

PMERR_INSUFFICIENT_MEMORY
The operation terminated through insufficient memory.

PMERR_INTERNAL_ERROR_n
An internal error has occurred. n is a number that identifies the particular error.

PMERR_INV_ANGLE_PARM
An invalid angle parameter was specified with GpiPartialArc.

PMERR_INV_ARC_CONTROL
An invalid control parameter was specified with GpiFullArc.

PMERR_INV_AREA_CONTROL	An invalid options parameter was specified with GpiBeginArea.
PMERR_INV_ATTR_MODE	An invalid mode parameter was specified with GpiSetAttrMode.
PMERR_INV_BACKGROUND_COL_ATTR	An invalid background color attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_BACKGROUND_MIX_ATTR	An invalid background mix attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_BITBLT_MIX	An invalid *lRop* parameter was specified with a GpiBitBlt or GpiWCBitBlt function.
PMERR_INV_BITBLT_STYLE	An invalid options parameter was specified with a GpiBitBlt or GpiWCBitBlt function.
PMERR_INV_BITMAP_DATA	In processing a bit map, the end of the data was unexpectedly encountered.
PMERR_INV_BITMAP_DIMENSION	An invalid dimension was specified with a load bit-map function.
PMERR_INV_BOX_CONTROL	An invalid control parameter was specified with GpiBox.
PMERR_INV_BOX_ROUNDING_PARM	An invalid corner rounding control parameter was specified with GpiBox.
PMERR_INV_CHAR_ALIGN_ATTR	The text alignment attribute specified in GpiSetTextAlignment is not valid.
PMERR_INV_CHAR_ANGLE_ATTR	The default character angle attribute value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_DIRECTION_ATTR	An invalid character direction attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_MODE_ATTR	An invalid character mode attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_POS_OPTIONS	An invalid options parameter was specified with GpiCharStringPos or GpiCharStringPosAt.
PMERR_INV_CHAR_SET_ATTR	An invalid character setid attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_SHEAR_ATTR	An invalid character shear attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CLIP_PATH_OPTIONS	An invalid options parameter was specified with GpiSetClipPath.
PMERR_INV_CODEPAGE	An invalid code-page parameter was specified with GpiSetCp.

PMERR_INV_COLOR_ATTR	An invalid color attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_COLOR_DATA	Invalid color table definition data was specified with GpiCreateLogColorTable.
PMERR_INV_COLOR_FORMAT	An invalid format parameter was specified with GpiCreateLogColorTable.
PMERR_INV_COLOR_INDEX	An invalid color index parameter was specified with GpiQueryRGBColor.
PMERR_INV_COLOR_OPTIONS	An invalid options parameter was specified with a logical color table or color query function.
PMERR_INV_COLOR_START_INDEX	An invalid starting index parameter was specified with a logical color table or color query function.
PMERR_INV_CONV	Invalid conversion-type parameter.
PMERR_INV_COORD_OFFSET	An invalid coordinate offset value was specified.
PMERR_INV_COORD_SPACE	An invalid source or target coordinate space parameter was specified with GpiConvert.
PMERR_INV_COORDINATE	An invalid coordinate value was specified.
PMERR_INV_CORRELATE_DEPTH	An invalid maxdepth parameter was specified with GpiCorrelateSegment, GpiCorrelateFrom, or GpiCorrelateChain.
PMERR_INV_CORRELATE_TYPE	An invalid type parameter was specified with GpiCorrelateSegment, GpiCorrelateFrom, or GpiCorrelateChain.
PMERR_INV_CURSOR_BITMAP	An invalid pointer was referenced with WinSetPointer.
PMERR_INV_DC_DATA	An invalid data parameter was specified with DevOpenDC.
PMERR_INV_DC_TYPE	An invalid type parameter was specified with DevOpenDC, or a function was issued that is invalid for a OD_METAFILE_NOQUERY device context.
PMERR_INV_DEV_MODES_OPTIONS	An invalid options parameter was specified with DevPostDeviceModes.
PMERR_INV_DEVICE_NAME	An invalid devicename parameter was specified with DevPostDeviceModes.
PMERR_INV_DRAW_BORDER_OPTION	An invalid option parameter was specified with WinDrawBorder.
PMERR_INV_DRAW_CONTROL	An invalid control parameter was specified with GpiSetDrawControl or GpiQueryDrawControl.
PMERR_INV_DRAW_VALUE	An invalid value parameter was specified with GpiSetDrawControl.
PMERR_INV_DRAWING_MODE	An invalid mode parameter was specified with GpiSetDrawControl not **draw-and-retain** or **draw**.
PMERR_INV_DRIVER_DATA	Invalid driver data was specified.
PMERR_INV_DRIVER_NAME	A driver name was specified which has not been installed.
PMERR_INV_EDIT_MODE	An invalid mode parameter was specified with GpiSetEditMode.
PMERR_INV_ELEMENT_OFFSET	An invalid off (offset) parameter was specified with GpiQueryElement.

PMERR_INV_ELEMENT_POINTER	An attempt was made to issue GpiPutData with the element pointer not pointing at the last element.
PMERR_INV_END_PATH_OPTIONS	An attempt to create or delete a path out of context of the path bracket was made.
PMERR_INV_ESCAPE_CODE	An invalid code parameter was specified with DevEscape.
PMERR_INV_ESCAPE_DATA	An invalid data parameter was specified with DevEscape.
PMERR_INV_FACENAME	An invalid font family name was passed to GpiQueryFaceString.
PMERR_INV_FACENAMEDESC	The font facename description is invalid.
PMERR_INV_FILL_PATH_OPTIONS	An invalid options parameter was specified with GpiFillPath.
PMERR_INV_FIRST_CHAR	An invalid firstchar parameter was specified with GpiQueryWidthTable.
PMERR_INV_FLOOD_FILL_OPTIONS	Invalid flood fill parameters were specified.
PMERR_INV_FONT_ATTRS	An invalid attrs parameter was specified with GpiCreateLogFont.
PMERR_INV_FONT_FILE_DATA	The font file specified with GpiLoadFonts, GpiLoadPublicFonts, GpiQueryFontFileDescriptions, or GpiQueryFullFontFileDescriptions contains invalid data.
PMERR_INV_FOR_THIS_DC_TYPE	An attempt has been made to issue GpiRemoveDynamics or GpiDrawDynamics to a presentation space associated with a metafile device context.
PMERR_INV_FORMS_CODE	An invalid forms code parameter was specified with DevQueryHardcopyCaps.
PMERR_INV_GEOM_LINE_WIDTH_ATTR	An invalid geometric line width attribute value was specified.
PMERR_INV_GETDATA_CONTROL	An invalid format parameter was specified with GpiGetData.
PMERR_INV_GRAPHICS_FIELD	An invalid field parameter was specified with GpiSetGraphicsField.
PMERR_INV_HBITMAP	An invalid bit-map handle was specified.
PMERR_INV_HDC	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.
PMERR_INV_HFONT	An invalid font handle was specified.
PMERR_INV_HMF	An invalid metafile handle was specified.
PMERR_INV_HPAL	An invalid color palette handle was specified.
PMERR_INV_HPS	An invalid presentation-space handle was specified.
PMERR_INV_HRGN	An invalid region handle was specified.
PMERR_INV_ID	An invalid *lPSid* parameter was specified with GpiRestorePS.
PMERR_INV_IMAGE_DATA_LENGTH	An invalid *lLength* parameter was specified with GpiImage. There is a mismatch between the image size and the data length.

PMERR_INV_IMAGE_DIMENSION	An invalid *psizIImageSize* parameter was specified with GpiImage.
PMERR_INV_IMAGE_FORMAT	An invalid *lFormat* parameter was specified with GpiImage.
PMERR_INV_IN_AREA	An attempt was made to issue a function invalid inside an area bracket. This can be detected while the actual drawing mode is **draw** or **draw-and-retain** or during segment drawing or correlation functions.
PMERR_INV_IN_CURRENT_EDIT_MODE	An attempt was made to issue a function invalid inside the current editing mode.
PMERR_INV_IN_ELEMENT	An attempt was made to issue a function invalid inside an element bracket.
PMERR_INV_IN_IMAGE	An attempt was made to issue a function invalid inside an element bracket.
PMERR_INV_IN_PATH	An attempt was made to issue a function invalid inside a path bracket.
PMERR_INV_IN_RETAIN_MODE	An attempt was made to issue a function (for example, query) that is invalid when the actual drawing mode is not **draw** or **draw-and-retain**.
PMERR_INV_IN_SEG	An attempt was made to issue a function invalid inside a segment bracket.
PMERR_INV_IN_VECTOR_SYMBOL	An invalid order was detected inside a vector symbol definition while drawing a vector (outline) font.
PMERR_INV_INFO_TABLE	An invalid bit-map info table was specified with a bit-map operation.
PMERR_INV_LENGTH_OR_COUNT	An invalid length or count parameter was specified.
PMERR_INV_LINE_END_ATTR	An invalid line end attribute value was specified.
PMERR_INV_LINE_JOIN_ATTR	An invalid line join attribute value was specified.
PMERR_INV_LINE_TYPE_ATTR	An invalid line type attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_LINE_WIDTH_ATTR	An invalid line width attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_LOGICAL_ADDRESS	An invalid device logical address was specified.
PMERR_INV_MARKER_BOX_ATTR	An invalid marker box attribute value was specified.
PMERR_INV_MARKER_SET_ATTR	An invalid marker set attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MARKER_SYMBOL_ATTR	An invalid marker symbol attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MATRIX_ELEMENT	An invalid transformation matrix element was specified.
PMERR_INV_MAX_HITS	An invalid maxhits parameter was specified with GpiCorrelateSegment, GpiCorrelateFrom, or GpiCorrelateChain.

PMERR_INV_METAFILE	An invalid metafile was specified with GpiPlayMetaFile.
PMERR_INV_METAFILE_LENGTH	An invalid length parameter was specified with GpiSetMetaFileBits or GpiQueryMetaFileBits.
PMERR_INV_METAFILE_OFFSET	An invalid length parameter was specified with GpiSetMetaFileBits or GpiQueryMetaFileBits.
PMERR_INV_MICROPS_DRAW_CONTROL	A draw control parameter was specified with GpiSetDrawControl that is invalid in a micro presentation space.
PMERR_INV_MICROPS_FUNCTION	An attempt was made to issue a function that is invalid in a micro presentation space.
PMERR_INV_MICROPS_ORDER	An attempt was made to play a metafile containing orders that are invalid in a micro presentation space.
PMERR_INV_MIX_ATTR	An invalid mix attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MODE_FOR_OPEN_DYN	An attempt was made to open a segment with the ATTR_DYNAMIC segment set, while the drawing mode was set to DM_DRAW or DM_DRAWANDRETAIN.
PMERR_INV_MODE_FOR_REOPEN_SEG	An attempt was made to reopen an existing segment while the drawing mode was set to DM_DRAW or DM_DRAWANDRETAIN.
PMERR_INV_MODIFY_PATH_MODE	An invalid mode parameter was specified with GpiModifyPath.
PMERR_INV_MULTIPLIER	An invalid multiplier parameter was specified with GpiPartialArc or GpiFullArc.
PMERR_INV_NESTED_FIGURES	Nested figures have been detected within a path definition.
PMERR_INV_OR_INCOMPAT_OPTIONS	An invalid or incompatible (with micro presentation space) options parameter was specified with GpiCreatePS or GpiSetPS.
PMERR_INV_ORDER_LENGTH	An invalid order length was detected during GpiPutData or segment drawing.
PMERR_INV_ORDERING_PARM	An invalid order parameter was specified with GpiSetSegmentPriority.
PMERR_INV_OUTSIDE_DRAW_MODE	An attempt was made to issue a GpiSavePS or GpiRestorePS function, or an output only function (for example, GpiPaintRegion) from GpiPlayMetaFile without the drawing mode set to DM_DRAW.
PMERR_INV_PAGE_VIEWPORT	An invalid viewport parameter was specified with GpiSetPageViewport.
PMERR_INV_PATH_CONVERT_OPTIONS	An invalid options parameter was specified with GpiOutlinePath.
PMERR_INV_PATH_ID	An invalid path identifier parameter was specified.
PMERR_INV_PATTERN_ATTR	An invalid pattern symbol attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_PATTERN_REF_PT_ATTR	An invalid refpoint attribute value was specified.

PMERR_INV_PATTERN_SET_ATTR	An invalid pattern set attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_PATTERN_SET_FONT	An attempt was made to use an unsuitable font as a pattern set.
PMERR_INV_PICK_APERTURE_OPTION	An invalid options parameter was specified with GpiSetPickApertureSize.
PMERR_INV_PICK_APERTURE_POSN	An invalid pick aperture position was specified.
PMERR_INV_PICK_APERTURE_SIZE	An invalid size parameter was specified with GpiSetPickApertureSize.
PMERR_INV_PLAY_METAFILE_OPTION	An invalid option parameter was specified with GpiPlayMetaFile.
PMERR_INV_PRIMITIVE_TYPE	An invalid primitive type parameter was specified with GpiSetAttrs or GpiQueryAttrs.
PMERR_INV_PS_SIZE	An invalid size parameter was specified with GpiCreatePS or GpiSetPS.
PMERR_INV_PUTDATA_FORMAT	An invalid format parameter was specified with GpiPutData.
PMERR_INV_QUERY_ELEMENT_NO	An invalid start parameter was specified with DevQueryCaps.
PMERR_INV_RECT	An invalid rectangle parameter was specified.
PMERR_INV_REGION_CONTROL	An invalid control parameter was specified with GpiQueryRegionRects.
PMERR_INV_REGION_MIX_MODE	An invalid mode parameter was specified with GpiCombineRegion.
PMERR_INV_REPLACE_MODE_FUNC	An attempt was made to issue GpiPutData with the editing mode set to SEGEM_REPLACE.
PMERR_INV_RESERVED_FIELD	An invalid reserved field was specified.
PMERR_INV_RESET_OPTIONS	An invalid options parameter was specified with GpiResetPS.
PMERR_INV_RGBCOLOR	An invalid rgb color parameter was specified with GpiQueryNearestColor or GpiQueryColor.
PMERR_INV_SCAN_START	An invalid scanstart parameter was specified with a bit-map function.
PMERR_INV_SEG_ATTR	An invalid attribute parameter was specified with GpiSetSegmentAttrs, GpiQuerySegmentAttrs, GpiSetInitialSegmentAttrs, or GpiQueryInitialSegmentAttrs.
PMERR_INV_SEG_ATTR_VALUE	An invalid attribute value parameter was specified with GpiSetSegmentAttrs or GpiSetInitialSegmentAttrs.
PMERR_INV_SEG_NAME	An invalid segment identifier was specified.
PMERR_INV_SEG_OFFSET	An invalid offset parameter was specified with GpiPutData.
PMERR_INV_SEGLEN	An order length exceeds the remaining segment length in the input PIF.
PMERR_INV_SETID	An invalid setid parameter was specified.
PMERR_INV_SHARPNESS_PARM	An invalid sharpness parameter was specified with GpiPolyFilletSharp.

PMERR_INV_STOP_DRAW_VALUE	An invalid value parameter was specified with GpiSetStopDraw.
PMERR_INV_TRANSFORM_TYPE	An invalid options parameter was specified with a transform matrix function.
PMERR_INV_TYPE	Invalid file-type parameter.
PMERR_INV_USAGE_PARM	An invalid options parameter was specified with GpiCreateBitmap.
PMERR_INV_VIEWING_LIMITS	An invalid limits parameter was specified with GpiSetViewingLimits.
PMERR_INV_VIEWLIM	A set viewing limits order has an inconsistent mask and order length in the input PIF.
PMERR_INV_XFORM	A set (default) viewing transform order has an inconsistent mask and order length in the input PIF.
PMERR_INV_3DCOORD	An order specifying 3-dimensional coordinates has been found in the input PIF.
PMERR_INVALID_ARRAY_COUNT	An array has an invalid count, that is, less than or equal to zero.
PMERR_INVALID_APPL	Attempted to start an application whose type is not recognized by OS/2.
PMERR_INVALID_ARRAY_SIZE	A control data type array size is invalid.
PMERR_INVALID_ASCIIZ	The profile string is not a valid zero-terminated string.
PMERR_INVALID_ATOM	The specified atom does not exist in the atom table.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_INVALID_BUNDLE_TYPE	An invalid bundle type was passed.
PMERR_INVALID_CHARACTER_INDEX	On WinNextChar or WinPrevChar, a character index is invalid, that is, it is less than 1 or is greater than the string length+1.
PMERR_INVALID_CONTROL_DATATYPE	An invalid control data type was specified.
PMERR_INVALID_CONTROL_SEQ_INDEX	There is an invalid index in a control data type sequence (for array, length, offset or MPARAM) that is, the index is to a non-existent or non-numeric entry.
PMERR_INVALID_DATATYPE	An invalid data type was specified.
PMERR_INVALID_DST_CODEPAGE	The destination code page parameter is invalid.
PMERR_INVALID_FLAG	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.
PMERR_INVALID_ERRORINFO_HANDLE	On WinFreeErrorInfo, the ERRINFO is not the handle of an ERRINFO structure, that is, it was not created by WinGetErrorInfo.
PMERR_INVALID_FREE_MESSAGE_ID	An invalid message identifier was specified. The call has completed by assuming the message parameter and reply data types to be ULONG.
PMERR_INVALID_GROUP_HANDLE	An invalid program-group handle was specified.
PMERR_INVALID_HACCEL	An invalid accelerator-table handle was specified.
PMERR_INVALID_HANDLE	An invalid handle was specified.
PMERR_INVALID_HAPP	The application handle passed to WinTerminateApp does not correspond to a valid session.

PMERR_INVALID_HATOMTBL	An invalid atom-table handle was specified.
PMERR_INVALID_HEAP_POINTER	An invalid pointer was found within the heap.
PMERR_INVALID_HEAP_SIZE_PARM	Invalid data was found within the heap.
PMERR_INVALID_HEAP_SIZE_WORD	Invalid data was found within the heap.
PMERR_INVALID_HENUM	An invalid enumeration handle was specified.
PMERR_INVALID_HHEAP	An invalid heap handle was specified.
PMERR_INVALID_HMQ	An invalid message-queue handle was specified.
PMERR_INVALID_HPTR	An invalid pointer handle was specified.
PMERR_INVALID_HSTRUCT	An invalid (null) structure handle was specified.
PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_INI_FILE_HANDLE	An invalid initialization-file handle was specified.
PMERR_INVALID_INTEGER_ATOM	The specified atom is not a valid integer atom.
PMERR_INVALID_MESSAGE_ID	A message identifier is invalid.
PMERR_INVALID_NUMBER_OF_PARMS	The number of parameters is invalid.
PMERR_INVALID_NUMBER_OF_TYPES	The function call has an invalid number (zero) of types.
PMERR_INVALID_PARAMETERS	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range –32,768 to +32,767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.
PMERR_INVALID_PARAMETER_TYPE	A parameter type is invalid for a bundle mask.
PMERR_INVALID_PARM	A parameter to the function contained invalid data.
PMERR_INVALID_PROGRAM_HANDLE	An invalid program handle was specified.
PMERR_INVALID_SESSION_ID	The specified session identifier is invalid. Either zero (for the application's own session) or a valid identifier must be specified.
PMERR_INVALID_SRC_CODEPAGE	The source code page parameter is invalid.
PMERR_INVALID_STRING_PARM	The specified string parameter is invalid.
PMERR_INVALID_SWITCH_HANDLE	An invalid Window List entry handle was specified.
PMERR_INVALID_TARGET_HANDLE	An invalid target program-group handle was specified.
PMERR_INVALID_TITLE	The specified program or group title is too long or contains invalid characters.
PMERR_INVALID_TYPE_FOR_LENGTH	The data type for a control length is invalid.
PMERR_INVALID_TYPE_FOR_MPARAM	The message parameter type for a control MPARAM is invalid, that is, not mparam1, mparam2 or mreply.
PMERR_INVALID_TYPE_FOR_OFFSET	The data type for a control offset is invalid.
PMERR_INVALID_WINDOW	The window specified with a Window List call is not a valid frame window.
PMERR_KERNING_NOT_SUPPORTED	Kerning was requested on GpiCreateLogFont call to a presentation space associated with a device context that does not support kerning.
PMERR_LABEL_NOT_FOUND	The specified element label did not exist.

PMERR_MATRIX_OVERFLOW	An internal overflow error occurred during matrix multiplication. This can occur if coordinates or matrix transformation elements (or both) are invalid or too large.
PMERR_MEMORY_ALLOC	An error occurred during memory management.
PMERR_MEMORY_ALLOCATION_ERR	An error occurred during memory management.
PMERR_MEMORY_DEALLOCATION_ERR	An error occurred during memory management.
PMERR_METAFILE_INTERNAL_ERROR	An internal inconsistency has been detected during metafile unlock processing.
PMERR_METAFILE_IN_USE	An attempt has been made to access a metafile that is in use by another thread.
PMERR_METAFILE_LIMIT_EXCEEDED	The maximum permitted metafile size limit was exceeded during metafile recording.
PMERR_MSGID_TOO_SMALL	The message identifier specified is too small.
PMERR_NEGATIVE_STRCOND_DIM	A negative array dimension was passed for a data type length.
PMERR_NO_BITMAP_SELECTED	An attempt has been made to operate on a memory device context that has no bit map selected.
PMERR_NO_CURRENT_ELEMENT	An attempt has been made to issue GpiQueryElementType or GpiQueryElement while there is no currently open element.
PMERR_NO_CURRENT_SEG	An attempt has been made to issue GpiQueryElementType or GpiQueryElement while there is no currently open segment.
PMERR_NO_FILL	No flood fill occurred because either the starting point color was the same as the input color when a boundary fill was requested, or the starting point color was not the same as the input color when a surface fill was requested.
PMERR_NO_METAFILE_RECORD_HANDLE	The metafile record handle was not found during metafile recording, or DevEscape (DEVESC_STARTDOC) was not issued when drawing to a OD_QUEUED device context with a *pszDataType* field of PM_Q_STD.
PMERR_NO_PALETTE_SELECTED	An attempt to realize a palette failed because no palette was previously selected into the Presentation Space.
PMERR_NO_SPACE	The limit on the number of Window List entries has been reached with WinAddSwitchEntry.
PMERR_NOT_CREATED_BY_DEVOPENDC	An attempt has been made to destroy a device context using DevCloseDC that was not created using DevOpenDC.
PMERR_NOT_CURRENT_PL_VERSION	An unexpected data format was found in the initialization file.
PMERR_NOT_DRAGGING	A drag operation is not in progress at this time.
PMERR_NOT_IN_A_PM_SESSION	An attempt was made to access function that is only available from PM programs from a non-PM session.
PMERR_NOT_IN_AREA	An attempt was made to end an area using GpiEndArea or during segment drawing while not in an area bracket.

PMERR_NOT_IN_DRAW_MODE
An attempt was made to issue GpiSavePS or GpiRestorePS while the drawing mode was not set to DM_DRAW.

PMERR_NOT_IN_ELEMENT
An attempt was made to end an element using GpiEndElement or during segment drawing while not in an element bracket.

PMERR_NOT_IN_IDX
The application name, key-name or program handle was not found.

PMERR_NOT_IN_IMAGE
An attempt was made to end an image during segment drawing while not in an image bracket.

PMERR_NOT_IN_PATH
An attempt was made to end a path using GpiEndPath or during segment drawing while not in a path bracket.

PMERR_NOT_IN_RETAIN_MODE
An attempt was made to issue a segment editing element function that is invalid when the actual drawing mode is not set to **retain**.

PMERR_NOT_IN_SEG
An attempt was made to end a segment using GpiCloseSegment while not in a segment bracket.

PMERR_NOT_SELF_DESCRIBING_DTYP
A data type is not self-describing.

PMERR_OPENING_INI_FILE
Unable to open initialization file (due to lack of disk space for example).

PMERR_ORDER_TOO_BIG
An internal size limit was exceeded while converting orders from short to long format during GpiPutData processing. An order was too long to convert.

PMERR_OWN_SET_ID_REFS
An attempt to unload a font failed because the setid is still being referenced.

PMERR_PALETTE_BUSY
An attempt has been made to reset the owner of a palette when it was busy.

PMERR_PALETTE_SELECTED
Color palette operations cannot be performed on a presentation space while a palette is selected.

PMERR_PARAMETER_OUT_OF_RANGE
The value of a parameter was not within the defined valid range for that parameter.

PMERR_PATH_INCOMPLETE
An attempt was made to open or close a segment either directly or during segment drawing, or to issue GpiAssociate while there is an open path bracket.

PMERR_PATH_LIMIT_EXCEEDED
An internal size limit was exceeded during path or area processing.

PMERR_PATH_UNKNOWN
An attempt was made to perform a path function on a path that did not exist.

PMERR_PEL_IS_CLIPPED
An attempt was made to query a pel that had been clipped using GpiQueryPel.

PMERR_PEL_NOT_AVAILABLE
An attempt was made to query a pel that did not exist in GpiQueryPel (for example, a memory device context with no selected bit map).

PMERR_PROLOG_ERROR
A prolog error was detected during drawing. Segment prologs are used internally within retained segments and also appear in metafiles. This error can also arise from an End Prolog order that is outside a prolog.

PMERR_PRINTER_DD_NOT_DEFINED	The Presentation Manager device driver has not been defined.
PMERR_PRINTER_QUEUE_NOT_DEFINED	The spooler queue for the printer has not been defined.
PMERR_PRN_ADDR_IN_USE	A printer is already defined on the port.
PMERR_PRN_ADDR_NOT_DEFINED	The printer port has not been defined.
PMERR_PRN_NAME_NOT_DEFINED	The printer has not been defined.
PMERR_PS_BUSY	An attempt was made to access the presentation space from more than one thread simultaneously.
PMERR_PS_IS_ASSOCIATED	An attempt was made to destroy a presentation or associate a presentation space that is still associated with a device context.
PMERR_PS_NOT_ASSOCIATED	An attempt was made to access a presentation space that is not associated with a device context.
PMERR_QUEUE_ALREADY_EXISTS	An attempt to create a message queue for a thread failed because one already exists for the calling thread.
PMERR_RASTER_FONT	A request was made for the outline of a bit-map font. Outlines can only be returned for vector font characters.
PMERR_REALIZE_NOT_SUPPORTED	An attempt was made to create a realizable logical color table on a device driver that does not support this function.
PMERR_REGION_IS_CLIP_REGION	An attempt was made to perform a region operation on a region that is selected as a clip region.
PMERR_RESOURCE_DEPLETION	An internal resource depletion error has occurred.
PMERR_RESOURCE_NOT_FOUND	The specified resource identity could not be found.
PMERR_SEG_AND_REFSEG_ARE_SAME	The segid and refsegid specified with GpiSetSegmentPriority were the same.
PMERR_SEG_CALL_STACK_EMPTY	A call stack empty condition was detected when attempting a pop function during GpiPop or segment drawing.
PMERR_SEG_CALL_STACK_FULL	A call stack full condition was detected when attempting to call a segment using GpiCallSegmentMatrix, attempting to preserve an attribute, or during segment drawing.
PMERR_SEG_IS_CURRENT	An attempt was made to issue GpiGetData to a segment that was currently open.
PMERR_SEG_NOT_CHAINED	An attempt was made to issue GpiDrawFrom, GpiCorrelateFrom or GpiQuerySegmentPriority for a segment that was not chained.
PMERR_SEG_NOT_FOUND	The specified segment identifier did not exist.
PMERR_SEG_OVFLOW	The input PIF has more than 1000 called segments. This has overflowed an internal buffer.
PMERR_SEG_STORE_LIMIT_EXCEEDED	The maximum permitted retained segment store size limit was exceeded.
PMERR_SET_ID_REFS	An attempt to unload a font failed because the setid is still being referenced.
PMERR_SETID_IN_USE	An attempt was made to specify a setid that was already in use as the currently selected character, marker or pattern set.

PMERR_SETID_NOT_FOUND	An attempt was made to delete a setid that did not exist.
PMERR_SMB_OVFLOW	The input PIF has more than 100 symbol sets defined. This has overflowed an internal buffer.
PMERR_SOURCE_SAME_AS_TARGET	The direct manipulation source and target process are the same.
PMERR_SPL_CANNOT_OPEN_FILE	Unable to open the file.
PMERR_SPL_DD_NOT_FOUND	The Presentation Manager device driver definition could not be found.
PMERR_SPL_DEVICE_ALREADY_EXISTS	The device already exists.
PMERR_SPL_DEVICE_LIMIT_REACHED	The limit on the number of devices has been reached.
PMERR_SPL_DEVICE_NOT_INSTALLED	The device has not been installed.
PMERR_SPL_DRIVER_ERROR	No Presentation Manager device driver supplied or found.
PMERR_SPL_DRIVER_NOT_INSTALLED	The Presentation Manager device driver has not been installed.
PMERR_SPL_FILE_NOT_FOUND	Unable to find the file.
PMERR_SPL_HARD_NETWORK_ERROR	Hard network error.
PMERR_SPL_INI_FILE_ERROR	Error accessing the initialization file.
PMERR_SPL_INV_DATATYPE	The spool file data type is invalid.
PMERR_SPL_INV_DRIVER_DATATYPE	The data type is invalid for the Presentation Manager device driver.
PMERR_SPL_INV_FORMS_CODE	The forms code for the job is invalid.
PMERR_SPL_INV_HSPL	The spooler handle is invalid.
PMERR_SPL_INV_JOB_ID	The job id is invalid.
PMERR_SPL_INV_LENGTH_OR_COUNT	The length or count is invalid.
PMERR_SPL_INV_PRIORITY	The priority for the job is invalid.
PMERR_SPL_INV_PROCESSOR_DATTYPE	The data type is invalid for the spooler queue processor.
PMERR_SPL_INV_QUEUE_NAME	The spooler queue name is invalid.
PMERR_SPL_INV_TOKEN	The token is invalid.
PMERR_SPL_JOB_NOT_PRINTING	The print job is not printing.
PMERR_SPL_JOB_PRINTING	The print job is already printing.
PMERR_SPL_MANY_QUEUES_ASSOC	More than one queue has been associated with the printer.
PMERR_SPL_NO_CURRENT_FORMS_CODE	There is no current forms code defined to the Presentation Manager device driver.
PMERR_SPL_NO_DATA	No data supplied or found.
PMERR_SPL_NO_DEFAULT_QUEUE	There is no default spooler queue for the printer.
PMERR_SPL_NO_DISK_SPACE	There is not enough free disk space.
PMERR_SPL_NO_FREE_JOB_ID	There is no free job id available.
PMERR_SPL_NO_MEMORY	There is not enough free memory.
PMERR_SPL_NO_QUEUES_ASSOCIATED	A queue has not been associated with the printer.
PMERR_SPL_NO_SUCH_LOG_ADDRESS	The logical address does not exist (that is, it is not defined in the initialization file).

PMERR_SPL_NOT_AUTHORISED	Not authorized to perform the operation.
PMERR_SPL_PRINT_ABORT	The job has already been aborted.
PMERR_SPL_PRINTER_NOT_FOUND	The printer definition could not be found.
PMERR_SPL_PROCESSOR_ERROR	No spooler queue processor supplied or found.
PMERR_SPL_PROCESSOR_NOT_INST	The spooler queue processor has not been installed.
PMERR_SPL_QUEUE_ALREADY_EXISTS	The spooler queue already exists.
PMERR_SPL_QUEUE_ERROR	No spooler queue supplied or found.
PMERR_SPL_QUEUE_NOT_EMPTY	The spooler queue contains print jobs.
PMERR_SPL_QUEUE_NOT_FOUND	The spooler queue definition could not be found.
PMERR_SPL_SPOOLER_NOT_INSTALLED	The spooler is not installed.
PMERR_SPL_STATUS_STRING_TRUNC	The print job status string has been truncated.
PMERR_SPL_TEMP_NETWORK_ERROR	Temporary network error.
PMERR_SPL_TOO_MANY_OPEN_FILES	Too many open files.
PMERR_SPOOLER_QP_NOT_DEFINED	The spooler queue processor has not been defined.
PMERR_START_POINT_CLIPPED	The starting point specified for flood fill is outside the current clipping path or region.
PMERR_STARTDOC_NOT_ISSUED	A request to write spooled output without first issuing a STARTDOC was attempted.
PMERR_STARTED_IN_BACKGROUND	The application started a new session in the background.
PMERR_STOP_DRAW_OCCURRED	Segment drawing or GpiPlayMetaFile was stopped prematurely in response to a GpiSetStopDraw request.
PMERR_TOO_MANY_METAFILES_IN_USE	The maximum number of metafiles allowed for a given process was exceeded.
PMERR_TRUNCATED_ORDER	An incomplete order was detected during segment processing.
PMERR_UNABLE_TO_CLOSE_DEVICE	Unable to close the print device (for example, powered off or offline).
PMERR_UNCHAINED_SEG_ZERO_INV	An attempt was made to open segment with segment identifier zero and the ATTR_CHAINED segment attribute not specified.
PMERR_UNKNOWN_BUNDLE_TYPE	Unknown bundle-type primitive.
PMERR_UNSUPPORTED_ATTR	An unsupported attribute was specified in the attrmask with GpiSetAttrs or GpiQueryAttrs.
PMERR_UNSUPPORTED_ATTR_VALUE	An attribute value was specified with GpiSetAttrs that is not supported.
PMERR_WINDOW_LOCK_OVERFLOW	An overflow occurred for the use count of a window.
PMERR_WINDOW_LOCK_UNDERFLOW	An attempt was made to decrement the use count of a window below zero.
PMERR_WINDOW_NOT_LOCKED	The window specified in WinSendMsg was not locked.

Appendix D. Standard Bit-Map Formats

There are four standard bit-map formats. All device drivers have to be able to translate between any of these formats and their own internal formats. The standard formats are:

Bitcount	Planes
1	1
4	1
8	1
24	1

These formats are chosen because they are identical or similar to all formats commonly used by raster devices. Only single-plane formats are standard, but it is very easy to convert these to any multiple-plane format used internally by a device.

Bit-Map Data

The pel data is stored in the bit map in the order that the coordinates appear on a display screen. That is, the pel in the lower-left corner is the first in the bit map. Pels are scanned to the right, and upward, from that position. The bits of the first pel are stored, beginning with the most significant bits of the first byte. The data for pels in each scan line is packed together tightly, but all scan lines are padded at the end, so that each one begins on a ULONG boundary.

Bit-Map Information Tables

Each standard-format bit map must be accompanied by a bit-map information table. Because the standard-format bit maps are intended to be traded between devices, the color indexes in the bit map are meaningless without more information; for a description of this structure, see BITMAPINFO2.

Some calls use a structure that is similar to BITMAPINFO2 but does not have the color table array; for a description of this structure, see BITMAPINFOHEADER2. Wherever BITMAPINFO2 is shown, BITMAPINFO is also allowed. Similarly, wherever BITMAPINFOHEADER2 is shown, BITMAPINFOHEADER is also allowed.

Bit-Map Example

To make the ordering of all the bytes clear, consider this simple example of a 5-by-3 array of colored pels:

```
Red   Green Blue  Red   Green
Blue  Red   Green Blue  Red
Green Blue  Red   Green Blue

ULONG ExampleBitmap[]  {
    0x23,0x12,0x30,0x00              /* bottom line */
    0x31,0x23,0x10,0x00              /* middle line */
    0x12,0x31,0x20,0x00              /* top line    */
};

#define BLACK  0x00000000L
#define RED    0x00FF0000L
#define GREEN  0x0000FF00L
#define BLUE   0x000000FFL

struct BitmapInfoTable ExampleInfo = {
       5,                            /* width       */
       3,                            /* height      */
       1,                            /* planes      */
       4,                            /* bitcount    */
       BLACK,RED,GREEN,BLUE,         /* color table */
       BLACK,BLACK,BLACK,BLACK,
       BLACK,BLACK,BLACK,BLACK,
       BLACK,BLACK,BLACK,BLACK
};
```

Bit-Map File Format

The operating system uses the same file format for bit maps, icons, and pointers in resource files. In the following description, "bit map" refers to bit maps, icons, and pointers unless otherwise specified.

Two formats are supported. In the first, a single-size version of the bit map is defined. This is used whatever the target device.

The second format allows multiple versions of the bit map to be defined, including one or more device-independent versions, and a number of device-dependent versions, each intended for use with a particular device.

In the case of icons and pointers, when more than one version of the bit map exists, the preferred version is one that matches the device size of icon or pointer. Otherwise the device-independent version is used to scale a bit map to the required size.

The operating system provides pointers that match the requirements of the display device in use, typically pointers are 32x32 pels, one bit per plane.

Icons provided with the operating system are designed to match the requirements of the most common display devices. The following versions of each icon are included in each file:

32x32 4 bpp (16 color)
40x40 4 bpp (16 color)
32x32 1 bpp (black and white)
20x20 1 bpp (black and white)
16x16 1 bpp (black and white)

The 32x32 versions are designed for VGA displays and for device-independent use.

The 40x40 version is for 8514/A and XGA displays.

The 20x20 and 16x16 are half-size icons designed for use as mini-icons.

For general bit maps, which may be of arbitrary size, the preferred version is one matching the requested bit map size; otherwise one matching the display size is selected. If neither is available, the device-independent version is used from which to scale a bit map.

For both formats, the definition consists of two sections. The first section contains general information about the type, dimensions, and other attributes of the resource. The second section contains data describing the pels that make up the bit map(s), and is in the format specified in "Bit-Map Data" on page D-1.

In the multiple-version format, the first section contains an array of BITMAPARRAYFILEHEADER structures. or BITMAPARRAYFILEHEADER2 structures. The format of these is as follows:

```
typedef struct _BITMAPARRAYFILEHEADER {    /* bafh */
    USHORT            usType;
    ULONG             cbSize;
    ULONG             offNext;
    USHORT            cxDisplay;
    USHORT            cyDisplay;
    BITMAPFILEHEADER  bfh;
} BITMAPARRAYFILEHEADER;
typedef BITMAPARRAYFILEHEADER *PBITMAPARRAYFILEHEADER;

typedef struct _BITMAPARRAYFILEHEADER2 {    /* bafh */
    USHORT            usType;
    ULONG             cbSize;
    ULONG             offNext;
    USHORT            cxDisplay;
    USHORT            cyDisplay;
    BITMAPFILEHEADER2 bfh2;
} BITMAPARRAYFILEHEADER2;
typedef BITMAPARRAYFILEHEADER2 *PBITMAPARRAYFILEHEADER2;
```

The fields in BITMAPARRAYFILEHEADER and BITMAPARRAYFILEHEADER2 have these meanings:

usType Type of structure. This is:

BFT_BITMAPARRAY (X'4142' - 'BA' for BITMAPARRAYFILEHEADER or BITMAPARRAYFILEHEADER2)

cbSize Size of the BITMAPARRAYFILEHEADER or BITMAPARRAYFILEHEADER2 structure in bytes.

offNext Offset of the next BITMAPARRAYFILEHEADER or BITMAPARRAYFILEHEADER2 structure from the start of the file

cxDisplay, cyDisplay Pel dimensions of the device for which this version is intended (for example, 640 x 480 for VGA).

The device-independent version must be the first BITMAPARRAYFILEHEADER or BITMAPARRAYFILEHEADER2 defined.

In the single-size format, the BITMAPARRAYFILEHEADER or BITMAPARRAYFILEHEADER2 structure is not present. The definition consists of one or two BITMAPFILEHEADER or BITMAPFILEHEADER2 structures.

The format of the BITMAPFILEHEADER and BITMAPFILEHEADER2 structure is :

```
typedef struct _BITMAPFILEHEADER {    /* bfh */
    USHORT            usType;
    ULONG             cbSize;
    SHORT             xHotspot;
    SHORT             yHotspot;
    ULONG             offBits;
    BITMAPINFOHEADER  bmp;
} BITMAPFILEHEADER;
typedef BITMAPFILEHEADER *PBITMAPFILEHEADER;

typedef struct _BITMAPFILEHEADER2 {    /* bfh2 */
    USHORT            usType;
    ULONG             cbSize;
    SHORT             xHotspot;
    SHORT             yHotspot;
    ULONG             offBits;
    BITMAPINFOHEADER2 bmp2;
} BITMAPFILEHEADER2;
typedef BITMAPFILEHEADER2 *PBITMAPFILEHEADER2;
```

BITMAPINFOHEADER2 is a standard data type (see above, and also BITMAPINFOHEADER2).

The fields in BITMAPFILEHEADER and BITMAPFILEHEADER2 have these meanings:

usType Type of resource the file contains. The valid values are:

BFT_BMAP (X'4D42' - 'BM' for bit maps)
BFT_ICON (X'4349' - 'IC' for icons)
BFT_POINTER (X'5450' - 'PT' for pointers).
BFT_COLORICON (X'4943' - 'CI' for color icons).
BFT_COLORPOINTER (X'5043' - 'CP' for color pointers).

cbSize Size of the BITMAPFILEHEADER or BITMAPFILEHEADER2 structure in bytes.

xHotspot, yHotspot Coordinates of the hotspot for icons and pointers. This field is ignored for bit maps.

offBits Offset in bytes to the beginning of the bit-map pel data in the file, from the start of the definition.

For icons and pointers, the *cy* field in **bmp** is actually twice the pel height of the image that appears on the screen. This is because these types actually contain two full bit-map pel definitions. The first bit-map definition is the XOR mask, which contains invert information (0 = no invert, 1 = invert) for the pointer or icon. The second is the AND mask, which determines whether the pointer or the screen is shown (0 = black/white, 1 = screen/inverse screen).

For color icons or pointers, there are two bit-maps involved: one that is black and white and consists of an AND and an XOR mask, and one that is color that defines the color content.

The *cy* field in the BITMAPINFOHEADER2 structure for the color bit-map must be the real height, that is, half the value specified for the black and white bit-map. The *cx* fields must be the same.

The following table shows how these two bit-maps are used for a color icon or pointer:

```
XOR     AND     COLOR
 1       1        x       Invert screen
 0       0        x       Use color x
 0       1        x       Transparency
 1       0        x       Use color x
```

For color icons or pointers, two BITMAPFILEHEADER or BITMAPFILEHEADER2 structures are therefore required:

```
BITMAPFILEHEADER2    with usType BFT_COLORICON or BFT_COLORPOINTER
   BITMAPINFOHEADER2 (part of BITMAPFILEHEADER2)
   Color table
BITMAPFILEHEADER2    with same usType
   BITMAPINFOHEADER2 (part of BITMAPFILEHEADER2)
   Color table
**
bits for one bit-map
**
**
bits for other bit-map
**
```

The usType for the first BITMAPFILEHEADER2 is either BFT_COLORICON or BFT_COLORPOINTER. This means that a second BITMAPFILEHEADER2 is present as part of the definition of a color icon or pointer. The first BITMAPFILEHEADER2 structure contains the information for the black and white AND and XOR masks, while the second BITMAPFILEHEADER2 structure contains the information for the color part of the pointer or icon.

BITMAPFILEHEADER and BITMAPINFOHEADER can occur in place of BITMAPFILEHEADER2 and BITMAPINFOHEADER2 in this example.

For the multiple version format, the file is as follows:

```
BITMAPARRAYFILEHEADER2   for device-independent version
   BITMAPFILEHEADER2     (part of BITMAPARRAYFILEHEADER2)
      BITMAPINFOHEADER2  (part of BITMAPFILEHEADER2)
      Color table

   BITMAPFILEHEADER2     )
      BITMAPINFOHEADER2  )  only if this is a color icon or pointer
      Color table        )

BITMAPARRAYFILEHEADER2   for first device-dependent version
   BITMAPFILEHEADER2     (part of BITMAPARRAYFILEHEADER2)
      BITMAPINFOHEADER2  (part of BITMAPFILEHEADER2)
      Color table

   BITMAPFILEHEADER2     )
      BITMAPINFOHEADER2  )  only if this is a color icon or pointer
      Color table        )
```

```
Further BITMAPARRAYFILEHEADER2 groups occur here as required
for additional device-dependent versions

**
bits for one bit-map
**
**
bits for next bit-map
**
```

And so on for as many bit-maps as necessary.

As before, BITMAPARRAYFILEHEADER, BITMAPFILEHEADER and BITMAPINFOHEADER can occur in place of BITMAPARRAYFILEHEADER2, BITMAPFILEHEADER2 and BITMAPINFOHEADER2.

Appendix E. Fonts Supplied with OS/2

OS/2 Outline Fonts

The following Adobe** Type 1 fonts are supplied with OS/2*:

Family Name	Face Name
Times New Roman**	Times New Roman Times New Roman Bold Times New Roman Bold Italic Times New Roman Italic
Helvetica**	Helvetica Helvetica Bold Helvetica Bold Italic Helvetica Italic
Courier	Courier Courier Bold Courier Bold Italic Courier Italic
Symbol	Symbol

The Courier, Tms Rmn, and Swiss family fonts that were supplied with OS/2 release 1.1 and 1.2 are no longer supplied. Using one of the old names results in one of the new fonts listed above being used, as follows:

Old Family/Face Name Font Used

Roman/Tms Rmn Times New Roman

Swiss/Helv Helvetica

These fonts are provided in an efficient binary format for use by the OS/2 Adobe Type Manager. They are also provided in standard Type 1 format (PFB and AFM) for use with the OS/2 PostScript** printer device driver.

Presentation Manager Bit Map Fonts

The following table lists all system bit map fonts available using the Graphics Programming Interface. Additional device bit map fonts may be available on specific devices. The table also gives the following information about each font:

Points This is the point size of the font, on a device whose resolution matches that of the font, (see "Device" below).

Ave Wid This is the average width in pels of alphabetic characters weighted according to US English letter frequencies.

** Adobe and PostScript are Trademarks of Adobe Systems Incorporated

* Trademark of IBM Corporation

** Times New Roman is a Trademark of Monotype

** Helvetica is a Trademark of Linotype

Max Wid This is the maximum width in pels of all characters in the font. This field is not necessarily the maximum width of any character in the code page. It could be used to ensure that the horizontal space allocated on a display or printer is big enough to handle any character.

Height This is the height in pels of the font. This is the minimum number of rows of pels needed to output any character of the font on a given baseline. This field may be larger than necessary for a given code page. It could be used to ensure that the vertical space allocated on a display or printer is big enough to handle any character.

Device This is the X and Y resolution in pels per inch at which the font is intended to be used. Only those fonts which match the device resolution of the installed display driver are available on the system. If the installed display is changed, the install process will reinstall the proper font sets for the new adapter. The IBM devices whose device drivers report these resolutions are:

96 x 48 CGA
96 x 72 EGA
96 x 96 VGA and XGA (in 640 x 480 mode)
120 x 120 8514/A and XGA (in 1024 x 768 mode)

Note: These values are approximate representations of the actual resolution, which in the case of displays depends on which monitor is attached. Consequently the point size of characters on the screen is also approximate.

Family	Face Name	Points	Av Wid	Max Wid	Height	Device
Courier	Courier	8	8	8	7	96x48
			8	8	10	96x72
			8	8	13	96x96
			9	9	16	120x120
		10	9	9	8	96x48
			9	9	12	96x72
			9	9	16	96x96
			12	12	20	120x120
		12	12	12	10	96x48
			12	12	15	96x72
			12	12	20	96x96
			15	15	25	120x120
System Proportional	System Proportional	8	6	20	8	96x48
		10	6	20	12	96x96
		10	6	20	16	96x96
		10	8	23	20	120x120
		11	10	23	23	120x120
System Monospaced	System Monospaced	8	8	8	8	96x48
		10	8	8	12	96x72
		10	8	8	16	96x96
		10	9	9	20	120x120
Helv	Helv	8	5	13	6	96x48

Family	Face Name	Points	Av Wid	Max Wid	Height	Device
			5	13	10	96x72
			5	13	13	96x96
			6	14	16	120x120
		10	6	15	8	96x48
			6	14	12	96x72
			6	14	16	96x96
			7	20	20	120x120
		12	7	17	10	96x48
			7	17	15	96x72
			7	17	20	96x96
			9	21	25	120x120
		14	8	21	12	96x48
			8	21	18	96x72
			8	21	24	96x96
			11	26	29	120x120
		18	11	26	15	96x48
			10	26	22	96x72
			11	26	29	96x96
			13	34	36	120x120
		24	14	35	19	96x48
			14	35	28	96x72
			14	35	37	96x96
			18	45	46	120x120
Tms Rmn	Tms Rmn	8	4	12	6	96x48
			4	13	10	96x72
			4	12	13	96x96
			5	14	16	120x120
		10	6	15	8	96x48
			5	14	12	96x72
			5	14	16	96x96
			7	19	20	120x120
		12	7	18	10	96x48
			6	18	15	96x72
			6	16	19	96x96
			8	23	23	120x120
		14	7	21	11	96x48
			7	21	16	96x72
			7	20	21	96x96
			10	26	27	120x120

Family	Face Name	Points	Av Wid	Max Wid	Height	Device
		18	10	26	14	96x48
			10	26	20	96x72
			10	26	27	96x96
			12	34	33	120x120
		24	14	35	18	96x48
			13	35	26	96x72
			13	35	35	96x96
			16	46	43	120x120

During system installation, the operating system determines the type of display adapter available on your computer and installs only the fonts which match the device resolution.

If you change your display device after the operating system is installed, you may also have to install the correct bit map fonts.

Appendix F. The Font-File Format

The OS/2 font-file format consists of two sections. The first section contains the general attributes of the font, and describes features such as its typeface, style, and nominal size. The second section contains the actual definitions of the characters belonging to the font.

The font resource is a set of self-defining records of the form:

```
typedef struct _RECORD {
        ULONG   ulIdentity;     /* structure identity code */
        ULONG   ulSize;         /* structure size in bytes */
         .                      /* data */
         .
         .
        } RECORD;
```

A font starts with a special font-signature structure and ends with an ending structure. The font signature has the form:

```
typedef struct _FONTSIGNATURE {
        ULONG   ulIdentity;
        ULONG   ulSize;
        CHAR    achSignature [12]
        } FONTSIGNATURE;
```

where:

```
ulIdentity   = X'FFFFFFFE'
ulSize       = 20
achSignature = "OS/2 FONT"   for an OS/2 1.x format font, or
             = "OS/2 FONT 2" for an OS/2 2.0 format font.
```

A 2.0 format font includes additional font description information in the PANOSE structure. This structure will be added to the end of the .FNT file (prior to the ENDFONT record).

The font end structure has the form:

```
typedef struct _ENDFONT{
        ULONG   ulIdentity;
        ULONG   ulSize;
        }ENDFONT
```

where:

```
ulIdentity   = X'FFFFFFFF'
ulSize       = 8
```

All records should be in the order of their identity fields.

There are three or four records in a font resource between the font signature and the font end:

- The font metrics
- The font character definitions
- The pair kerning table.
- The PANOSE description (for "OS/2 FONT 2" fonts).

Following compilation, the records in the resource are in the order defined above.

Metric Information Contained in Fonts

This section gives an explanation of how to set the fields of the FOCAMETRICS structure when developing:

- A bit map or outline font for general use by PM graphics applications

- A description of a bit map or outline device font that is built in to a device or can be downloaded to a device.

The following structure contains the physical font metrics used when creating fonts. It is defined in the file \INCLUDE\PMFONT.H.

```
typedef struct _FOCAMETRICS {
    ULONG   ulIdentity;
    ULONG   ulSize;
    CHAR    szFamilyname[32];
    CHAR    szFacename[32];
    SHORT   usRegistryId;
    SHORT   usCodePage;
    SHORT   yEmHeight;
    SHORT   yXHeight;
    SHORT   yMaxAscender;
    SHORT   yMaxDescender;
    SHORT   yLowerCaseAscent;
    SHORT   yLowerCaseDescent;
    SHORT   yInternalLeading;
    SHORT   yExternalLeading;
    SHORT   xAveCharWidth;
    SHORT   xMaxCharInc;
    SHORT   xEmInc;
    SHORT   yMaxBaselineExt;
    SHORT   sCharSlope;
    SHORT   sInlineDir;
    SHORT   sCharRot;
    USHORT  usWeightClass;
    USHORT  usWidthClass;
    SHORT   xDeviceRes;
    SHORT   yDeviceRes;
    SHORT   usFirstChar;
    SHORT   usLastChar;
    SHORT   usDefaultChar;
    SHORT   usBreakChar;
    SHORT   usNominalPointSize;
    SHORT   usMinimumPointSize;
    SHORT   usMaximumPointSize;
    SHORT   fsTypeFlags;
    SHORT   fsDefn;
    SHORT   fsSelectionFlags;
    SHORT   fsCapabilities;
    SHORT   ySubscriptXSize;
    SHORT   ySubscriptYSize;
    SHORT   ySubscriptXOffset;
    SHORT   ySubscriptYOffset;
    SHORT   ySuperscriptXSize;
    SHORT   ySuperscriptYSize;
    SHORT   ySuperscriptXOffset;
    SHORT   ySuperscriptYOffset;
    SHORT   yUnderscoreSize;
    SHORT   yUnderscorePosition;
    SHORT   yStrikeoutSize;
    SHORT   yStrikeoutPosition;
    SHORT   usKerningPairs;
    SHORT   sFamilyClass;
    PSZ     pszDeviceNameOffset;
  } FOCAMETRICS;
```

Note: FOCAMETRICS is a parallel structure with FONTMETRICS as returned to applications in the GpiQueryFonts and GpiQueryFontMetrics function calls.

The FONTMETRICS fields are derived from FOCAMETRICS by the Presentation Manager graphics engine. Most values are passed though unchanged. The exceptions are:

- The **Identity** field. This must be 1. This field is not a part of the FONTMETRICS structure.

- The **Size** field. This must be set to the size of the FOCAMETRICS structure. This field is not a part of the FONTMETRICS structure.
- The **Codepage** field. Ignore the description in FONTMETRICS, and use the following:

  Place 850 in this field if the font is intended to support any PM supported code page. The list of Presentation Manager supported code pages is given in Chapter 34, "Code Pages" on page 34-1.

  Place 65400 in this field if the font has special glyphs, for example if it is a Symbol font.

  Place other valid code pages in this field if the font is specific to this code page.

  Do not place other values in this field.
- FONTMETRICS fields which contain values in world coordinates. The corresponding field in FOCAMETRICS should contain pel values for bit-map fonts, and notional units for outline fonts.

See FONTMETRICS on page A-52 for a detailed explanation of the fields.

Font Character Definitions

Two formats of font character definition are supported. These are:

Image format
The character glyphs are represented as pel images.

Outline format
The character glyphs are represented by vector data that traces the outline of the character.

Note: Intelligent Font Technology fonts (such as ATM Type-1 fonts) may be stored in a technology specific format, and thus will not conform to this definition for outline fonts.

The definition consists of a header portion and a portion carrying the characters themselves.

The header portion contains information about the format of the character definitions and data about each character including width data and the offset into the definition section at which the character definition begins. (See "a-space, b-space, c-space" on page F-12.)

1. Proportional characters (a+b+c = character increment) for each character:

   $a,b,c \geq 0$

2. Characters where a, b, and c are definitions for all characters:

   $b \geq 0$
   a, c any integer

Raster fonts contain a "null character." The character definition record for this occurs after the one for the last character. Thus the format has usLastChar+2 characters, although the null character is not counted in the range returned. The null character is composed of zeros and is always eight pels wide.

Font Definition Header

This structure defines the format or the character definition records that follow it:

```
typedef struct_FONTDEFINITIONHEADER {
 ULONG        ulIdentity;
 ULONG        ulSize;
 SHORT        fsFontdef;
 SHORT        fsChardef;
 SHORT        usCellSize;
 SHORT        xCellWidth;
 SHORT        yCellHeight;
 SHORT        xCellIncrement;
 SHORT        xCellA;
 SHORT        xCellB;
 SHORT        xCellC;
 SHORT        pCellBaseOffset;
 } FONTDEFINITIONHEADER;
typedef FONTDEFINITIONHEADER FAR *PFONTDEFINITIONHEADER;
```

ulIdentity

4 bytes.

Must be equal to 2.

ulSize

4 bytes.

Size of this structure in bytes.

fsFontdef

2 bytes of flags.

Indicates which fields are present in the font definition data in the header.

Type 1

Bit 0	1 = width defined in header
Bit 1	1 = height defined in header
Bit 2	1 = char increment same as width, so that it is defined for the whole font
Bit 3	0 = a-space not defined
Bit 4	0 = b-space not defined
Bit 5	0 = c-space not defined
Bit 6	1 = base offset same for all characters.

Type 2

Bit 0	0 = width for each character unique
Bit 1	1 = height defined in header
Bit 2	0 = char increment same as width, so that it is unique for each character
Bit 3	0 = a-space not defined
Bit 4	0 = b-space not defined
Bit 5	0 = c-space not defined
Bit 6	1 = base offset same for all characters.

Type 3

Bit 0	0 = width for each character unique
Bit 1	1 = height defined in header
Bit 2	0 = char increment same as width, so that it is unique
Bit 3	0 = a-space not defined
Bit 4	0 = b-space not defined
Bit 5	0 = c-space not defined
Bit 6	1 = base offset same for all characters.

FsChardef

2 bytes of flags.

Indicates which fields are present on a per character basis.

Type 1

Bit 0	1 = width defined for each character (performance op)
Bit 1	0 = height is in header
Bit 2	0 = char increment is in header
Bit 3	0 = a-space not defined
Bit 4	0 = b-space not defined
Bit 5	0 = c-space not defined
Bit 6	0 = base offset defined in header
Bit 7	1 = offset to glyph defined.

Type 2

Bit 0	1 = width defined for each character
Bit 1	0 = height is in header
Bit 2	0 = char increment same as width
Bit 3	0 = a-space not defined
Bit 4	0 = b-space not defined
Bit 5	0 = c-space not defined
Bit 6	0 = base offset defined in header
Bit 7	1 = offset to glyph defined.

Type 3

Bit 0	1 = width not defined, use a, b, c
Bit 1	0 = height is in header
Bit 2	0 = char increment same as width
Bit 3	1 = a-space defined
Bit 4	1 = b-space defined
Bit 5	1 = c-space defined
Bit 6	0 = base offset defined in header
Bit 7	1 = offset to glyph defined.

usCellSize

2-byte integer.

Indicates the length in bytes of each character definition record (the per character data).

Type 1	6 bytes
Type 2	6 bytes
Type 3	10 bytes.

xCellWidth

2-byte integer

The width of the characters, in pels for image fonts, and relative units for outline fonts.

Type 1	Width of the characters
Type 2	Zero
Type 3	Zero.

yCellHeight

2-byte integer.

The height of the characters, in pels for image fonts, and relative units for outline fonts.

Type 1	Height of the characters
Type 2	Height of the characters
Type 3	Height of the characters.

xCellIncrement

2-byte integer.

The distance along the character baseline required to step from one character to the next (when forming a character string).

Type 1	Width of the characters
Type 2	Zero
Type 3	Zero.

xCellA

2-byte signed integer.

The width of the space before a character in the inline direction (the a-space).

Type 1	Zero
Type 2	Zero
Type 3	a-space for all characters.

xCellB

2-byte integer.

The width of a character (inline direction). The b-space.

Type 1	Zero
Type 2	Zero
Type 3	b-space for all characters.

xCellC

2-byte signed integer.

The width of the space after a character in the inline direction (the c-space).

Type 1	Zero
Type 2	Zero
Type 3	c-space for all characters.

pCellBaseOffset

2-byte signed integer.

The position of the top of a character definition relative to the baseline in the direction perpendicular to the baseline.

Type 1	Baseline offset for all characters
Type 2	Baseline offset for all characters
Type 3	Baseline offset for all characters.

Character Definition Record

xCellSize bytes per record.

The following fields may or may not be present, according to the font character definition fields flags. If a field is present, it is present for *each* character and the value applies to that character only.

There are usLastChar+2 such records for raster fonts. The final one is for the null character.

- Character Definition Offset: 4-byte integer.

  The offset into the Font File at which the character definition begins.

  Data for a single character raster or vector should not span two segments; that is, if a character is too big to fit into a segment it should be put in the next segment.

  This field should be set to zero if the character being defined is a blank character.

- Character Cell Width: 2-byte integer.

  The width of the character definition in pels.

- Character Cell Height: 2-byte integer.

  The height of the character definition in pels.

- Character Increment: 2-byte integer.

  The length along the character baseline required to step from this character to the next (when forming a character string).

- Character a-space: 2-byte signed integer.

  The width of the space before the character in the inline direction.

- Character b-space: 2-byte integer.

  The width of the character shape (inline direction).

- Character c-space: 2-byte signed integer.

  The width of the space after the character in the inline direction.

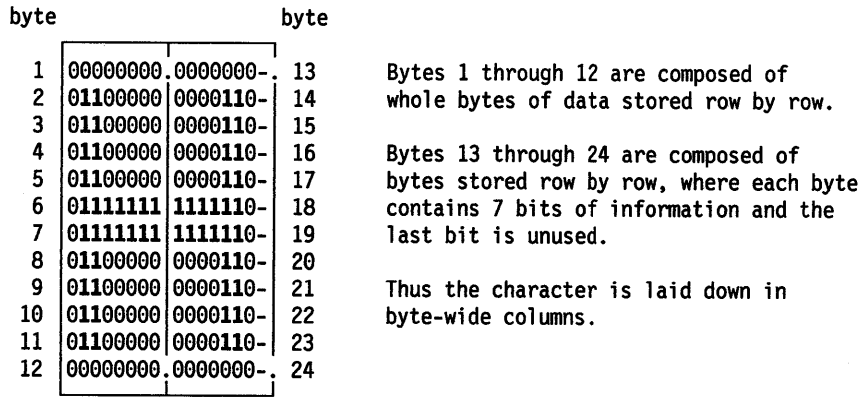
- Character Baseline Offset: 2-byte signed integer.

  The position of the top of a character definition relative to the baseline in the direction perpendicular to the baseline.

Note: Type 1 fonts have offset/width pairs (like type 2); however, the usCellSize and xCellIncrement are nonzero. In the fsType field of the font metrics, the proportional-space flag, bit 0, is set.

Image Data Format

The bits for each character are stored separately, and start on a byte boundary. Sequential bytes represent vertical pieces of the character image. For example, a 15-bit-wide H is stored as follows:



Notes:

1. There is always an additional (null) character defined in an Image Font (defined at character position LastChar+2) which is 8 bits wide, the height of the font character, and set to all zeros.
2. The maximum size of each individual Image Font must not exceed 64KB.

The Kerning Pair Table

The kerning pair table record is not present if the _KerningPairs record in the metrics is zero. If it is present, the code points are words, not bytes. This table should be sorted by kpChar1 and kpChar2 order to allow binary searches.

```
typedef struct _KERNPAIRTABLE {
        ULONG   ulIdentity;
        ULONG   ulSize;
        CHAR    cFirstpair;
}KERNPAIRTABLE;

typedef struct _KERNPAIRS {
        SHORT   sFirstChar;
        SHORT   sSecondChar;
        SHORT   sKerningAmount;
}KERNINGPAIRS;
```

where:

```
ulIdentity      = 3
ulSize          = 10
sFirstChar      = First character of the kerning pair
sSecondChar     = Second character of the kerning pair
sKerningAmount  = Kerning value.  Positive values increase the
                  inter-character spacing while negative values
                  bring the characters closer together.
```

Outline Data Format

Fonts defined by outlines (vectors) may contain any of these graphics orders:

- Line at given position (GLINE)
- Line at current position (GCLINE)
- Relative line at given position (GRLINE)
- Relative line at current position (GCRLINE)
- Fillet at given position (GFLT)
- Fillet at current position (GCFLT)
- Sharp fillet at given position (GSFLT)
- Sharp fillet at current position (GCSFLT)
- Bézier curve at given position (GBEZ)
- Bézier curve at current position (GCBEZ)
- No operation (GNOP1)
- Comment (GCOMT)
- End of symbol definition (GESD).

The maximum length of the data in these orders is 255 bytes. The drawing order code and the length fields are not included in the length count.

The size of each outline font definition must not be longer than 64KB.

The Additional Metrics

The additional metrics structure extends the metrics describing the font to include the PANOSE fields. The fields allow for quantitative descriptions of the visual properties of font faces. The format of the ADDITIONALMETRICS structure is:

```
typedef struct {
   ULONG       ulIdentity;
   ULONG       ulSize;
   PANOSE      panose;
} ADDITIONALMETRICS;

where:

  ulIdentity      = 4
  ulSize          = 20
  panose          = The ten digit PANOSE number with two bytes
                    of padding.
```

The PANOSE definition consists of ten digits, each of which describes one of up to sixteen variations. The current digits are:

1. Family Kind (6 variations)

   0 = Any
   1 = No Fit
   2 = Text and Display
   3 = Script
   4 = Decorative
   5 = Pictorial

2. Serif Style (16 variations)

   0 = Any
   1 = No Fit
   2 = Cove
   3 = Obtuse Cove
   4 = Square Cove
   5 = Obtuse Square Cove
   6 = Square
   7 = Thin
   8 = Bone
   9 = Exaggerated
   10 = Triangle
   11 = Normal Sans
   12 = Obtuse Sans
   13 = Perp Sans
   14 = Flared
   15 = Rounded

3. Weight (12 variations)

   0 = Any
   1 = No Fit
   2 = Very Light
   3 = Light
   4 = Thin
   5 = Book
   6 = Medium
   7 = Demi
   8 = Bold
   9 = Heavy
   10 = Black
   11 = Nord

4. Proportion (10 variations)

   0 = Any
   1 = No Fit
   2 = Old Style
   3 = Modern
   4 = Even Width
   5 = Expanded
   6 = Condensed
   7 = Very Expanded
   8 = Very Condensed
   9 = Monospaced

5. Contrast (10 variations)

   0 = Any
   1 = No Fit
   2 = None
   3 = Very Low
   4 = Low
   5 = Medium Low
   6 = Medium
   7 = Medium High
   8 = High
   9 = Very High

6. Stroke Variation (9 variations)

   0 = Any
   1 = No Fit
   2 = Gradual/Diagonal
   3 = Gradual/Transitional
   4 = Gradual/Vertical
   5 = Gradual/Horizontal
   6 = Rapid/Vertical
   7 = Rapid/Horizontal
   8 = Instant/Vertical

7. Arm Style (12 variations)

   0 = Any
   1 = No Fit
   2 = Straight Arms/Horizontal
   3 = Straight Arms/Wedge
   4 = Straight Arms/Vertical
   5 = Straight Arms/Single Serif
   6 = Straight Arms/Double Serif
   7 = Non-Straight Arms/Horizontal
   8 = Non-Straight Arms/Wedge
   9 = Non-Straight Arms/Vertical
   10 = Non-Straight Arms/Single Serif
   11 = Non-Straight Arms/Double Serif

8. Letterform (16 variations)

   0 = Any
   1 = No Fit
   2 = Normal/Contact
   3 = Normal/Weighted
   4 = Normal/Boxed
   5 = Normal/Flattened
   6 = Normal/Rounded
   7 = Normal/Off Center
   8 = Normal/Square
   9 = Oblique/Contact
   10 = Oblique/Weighted
   11 = Oblique/Boxed
   12 = Oblique/Flattened

13 = Oblique/Rounded
14 = Oblique/Off Center
15 = Oblique/Square

9. Midline (14 variations)

0 = Any
1 = No Fit
2 = Standard/Trimmed
3 = Standard/Pointed
4 = Standard/Serifed
5 = High/Trimmed
6 = High/Pointed
7 = High/Serifed
8 = Constant/Trimmed
9 = Constant/Pointed
10 = Constant/Serifed
11 = Low/Trimmed
12 = Low/Pointed
13 = Low/Serifed

10. X-height (8 variations)

0 = Any
1 = No Fit
2 = Constant/Small
3 = Constant/Standard
4 = Constant/Large
5 = Ducking/Small
6 = Ducking/Standard
7 = Ducking/Large

When using the PANOSE number to match fonts, the ordering of the PANOSE digit is the key to finding the closest match. The most significant digit is the first digit, and the least significant digit is number ten. To find matches, the digits need to be compared, in the order given. A font mapper may want to change the precedence of the digits, to give higher weightings to other font features.

Font Directory

This section describes the directory section of a font resource. A font resource contains a directory consisting of a set of structures each containing the metrics of a font and a pointer to the font itself. This font directory is generated by the resource compiler.

The format of the font directory is:

```
typedef struct {
   USHORT      usHeaderSize;
   USHORT      usnFonts;
   USHORT      usiMETRICS;
   FONTENTRY   fntEntry[1];
} FONTDIRECTORY;

typedef struct {
   USHORT          usIndex;
   FONTFILEMETRICS metrics;
} FONTENTRY;
```

Where:

usHeaderSize The size of the header, in bytes.

usnFonts The number of fonts in the resource.

usiMetrics The size of the FOCAMETRICS structures that follow the header. Note that the set of metrics for all the fonts in the resource follow the header.

usIndex	The index of a particular font; an identifier assigned to the font when the resource was created (defined in the .RC file).
metrics	The font metrics structure for the font. This is identical to a FOCAMETRICS structure with the addition of the PANOSE fields to the end.

Definitions of Terms Used When Describing Fonts

a-space, b-space, c-space
The a-space is the distance from the left of the character frame to the left edge of the character. The b-space is the width of the character. The c-space is the distance from the right edge of the character to the right of the character frame. Negative values of a and c allow adjacent character frames to overlap. See also *character increment*, and *space default values*.

average char width
The average horizontal distance from the left edge of one character to the left edge of the next. Contrast with *max char increment*.

baseline
The line on which the bottom of a character rests, and below which a descender extends.

break char code point
The *code point* of the space or break character. Contrast with *default char code point*, *first char code point*, and *last char code point*.

character increment
A set of three values (*a-space*, *b-space*, and *c-space*) that define the proportions of a character. The sum of the three values (a+b+c) specifies only one value for the entire character increment. See also *font width* and *space default values*.

character rotation
The angle by which each character is rotated around its own center, increasing clockwise from vertical. Contrast with *character slope* and *inline direction*.

character slope
The angle by which a character is slanted, increasing clockwise from vertical. Contrast with *character rotation* and *inline direction*.

default char code point
The *code point* of the character to be used if a *code point* outside the range of a font is passed to an application using that font. Contrast with *break char code point*, *first char code point*, and *last char code point*.

em height
The maximum distance above the *baseline* reached by an uppercase symbol. Contrast with *x height*.

external leading
The vertical distance from the bottom of one character to the top of the character below it. Contrast with *internal leading* and *max baseline extent*.

first char code point
The *code point* of the first character. All numbers between the *first char code point* and the *last char code point* must represent a character in the font. Contrast with *break char code point*, *default char code point*, and *last char code point*.

fixed spacing
The same amount of space separates each character. Contrast with *proportional spacing*.

font weight
The line-thickness of a character relative to its size. Contrast with *font width*.

font width
The relative width of a character to its height; condensed fonts are very narrow while expanded fonts are very wide. See also *character increment*. Contrast with *font weight*.

inline direction
The angle of a line of type, increasing clockwise from horizontal. Contrast with *character rotation* and *character slope*.

internal leading
The vertical distance from the top or bottom of a character to any accent marks that may appear with it. Contrast with *external leading*.

last char code point
The *code point* of the last character. All numbers between the *first char code point* and the *last char code point* must represent a character in the font. Contrast with *break char code point*, *default char code point*, and *first char code point*.

lowercase ascent
The maximum distance above the *baseline* reached by any part of any lowercase character. Contrast with *maximum ascender* and *x height*.

lowercase descent
The maximum distance below the *baseline* reached by any part of any lowercase character. Contrast with *maximum descender*.

max baseline extent
The maximum space occupied by the font (typically, the sum of the *maximum ascender* and *maximum descender*). Contrast with *external leading* and *max char increment*.

max char increment
The maximum horizontal distance from the left edge of one character to the left edge of the next character to the right. Contrast with *average char width* and *max baseline extent*.

maximum ascender
The maximum distance that any part of any character may extend above the *x height* of a font. Contrast with *lowercase ascent* and *maximum descender*.

maximum descender
The maximum distance that any part of any character may extend below the *x height* of a font. Contrast with *lowercase descent* and *maximum ascender*.

maximum vert point size
The maximum vertical dimensions to which a font can be resized. Contrast with *minimum vert point size* and *nominal vert point size*.

minimum vert point size
The minimum vertical dimensions to which a font can be resized. Contrast with *maximum vert point size* and *nominal vert point size*.

nominal vert point size
The normal display size of a font. Contrast with *maximum vert point size* and *minimum vert point size*.

pel
The smallest element of a display surface that can be independently assigned color and density.

point
Printer's unit of measurement. There are 72 points to an inch (approximately 3.5 points to a millimeter).

proportional spacing
The space that each character occupies is in proportion to its width. See also *font width*. Contrast with *fixed spacing*.

Registry ID
A code number that Presentation Manager uses to register a font file as a resource.

space default values
Values that specify the space to be left between characters. Once defined, they are used for the entire font, and do not have to be specified for each character. However, they can be changed for characters that require more or less spacing than the defaults provide, by giving values for the *a Space* and the *c Space*. See also *character increment*.

strikeout position
The distance of the strikeout character above the *baseline* (in *pels*). See also *strikeout size* and *underscore position*.

strikeout size
The size of the strikeout character (in *points*). See also *strikeout position* and *underscore size*.

subscript position
The distance of a subscript character of a font below the *baseline* (in *pels*). See also *subscript size* and *superscript position*.

subscript size
The size of a subscript character (in *points*). See also *subscript position* and *superscript size*.

superscript position
The distance of a superscript character above the *baseline* (in *pels*). See also *subscript position* and *superscript size*.

superscript size
The size of a superscript character (in *points*). See also *subscript size* and *superscript position*.

target dev resolution X
The number of *pels* per inch in the horizontal axis of a display device on which a font is to be displayed. Contrast with *target dev resolution Y*.

target dev resolution Y
The number of *pels* per inch in the vertical axis of a display device on which a font is to be displayed. Contrast with *target dev resolution X*.

underscore position
The distance in *pels* of the first underscore stroke from the *baseline* of a font. Successive strokes below this create a heavier underscore. See also *strikeout position* and *underscore size*.

underscore size
The size of the underscore character measured in single strikeout strokes. See also *strikeout size* and *underscore position*.

x height
The maximum distance above the *baseline* reached by a lowercase character. Contrast with *em height* and *lowercase ascent*.

Appendix G. Format of Interchange Files

A metafile is a file in which graphics are stored. The file is application-created, and it contains the graphics orders generated from those GPI calls that are valid in a metafile. Metafiled graphics can be reused by the application that created them. They can also be made available to other applications at the same, or at a different, workstation.

This chapter describes the restrictions which apply when generating the metafile and gives detail of the overall structure. For the graphics orders descriptions, see Chapter 33, "Graphics Orders" on page 33-1.

Metafile Restrictions

The following restrictions apply to the generation of all metafiles, and also to the generation of a PM_Q_STD print file to a OD_QUEUED device:

- If GpiWCBitBlt or GpiBitBlt is used to copy a bit map to a device context in an application, the application should not delete that bit map handle with GpiDeleteBitmap before the device context is closed (metafile is closed).
- GpiSetPS must not be used.
- GpiSetPageViewport is ignored.

The following section lists some general rules that must be followed when creating a metafile that is to be acceptable to SAA-conforming implementations, or replayed into a presentation space that is in **draw-and-retain** or **retain** mode (see GpiSetDrawingMode).

- These items must be established or defaulted before any drawing occurs to the graphics presentation space, and not changed subsequently:
  - The graphics field (GpiSetGraphicsField). For an SAA-conforming metafile, the graphics field must be defaulted or set to no clipping.
  - The code page for the default character set (GpiSetCp).
  - The color table or palette (GpiCreateLogColorTable or GpiCreatePalette). The size of the color table must not exceed 31KB (KB equals 1024 bytes).
  - The default viewing transform (GpiSetDefaultViewMatrix).
  - The setting of the draw controls (GpiSetDrawControl). DCTL_DISPLAY must be defaulted or set ON.
  - The default values of attributes (see GpiSetDefAttrs), viewing limits (see GpiSetDefViewingLimits), primitive tag (see GpiSetDefTag) and arc parameters (see GpiSetDefArcParams).
- These calls should not be used:
  - GpiBitBlt
  - GpiDeleteSetId (note that this means that local identifiers cannot be used again within the picture)
  - GpiErase
  - GpiExcludeClipRectangle
  - GpiIntersectClipRectangle
  - GpiOffsetClipRegion
  - GpiPaintRegion
  - GpiResetPS
  - GpiSetClipRegion
  - GpiSetPel
  - GpiSetPS
  - DevEscape (for an escape which is metafiled).
- GpiCreateLogFont must not redefine a local identifier that has previously been used within the picture.
- The metafile context must not be reassociated.

- If a bit map is used as the source of a GpiWCBitBlt operation, or as an area-fill pattern, it must not be modified or deleted (GpiDeleteBitmap) before the metafile is closed.
- Only these foreground mixes must be used (see GpiSetMix):
  - FM_DEFAULT
  - FM_OR
  - FM_OVERPAINT
  - FM_LEAVEALONE.
- Only these background mixes must be used (see GpiSetBackMix):
  - BM_DEFAULT
  - BM_OVERPAINT
  - BM_LEAVEALONE.
- If palettes are used (see GpiCreatePalette), the palette that is metafiled is the one in force when the metafile device context is dissociated from the (final) presentation space. If the palette is changed during the course of the picture (using GpiSetPaletteEntries), it must therefore only be with incremental additions.

Note: There is no restriction concerning the use of primitives outside segments. These are metafiled in segment(s) with zero identifier.

Metafile Data Format

This section describes the format of the data in a metafile, as it would be stored in an OS/2 Version 2.0 disk file.

Metafile data is stored as a sequence of **structured fields**. Each structured field starts with an eight-byte header consisting of a two-byte **length** field and a three-byte **identifier** field. These are followed by a one-byte **flags** field and a two-byte **segment sequence number** field.

The length field contains a count of the total number of bytes in the structured field, including the length field. The identifier field uniquely identifies the type of the structured field.

The flags and segment sequence number fields are always zero.

Following the header are positional parameters that are optional and dependent on the particular structured field.

Following the positional parameters are non-positional parameters called *triplets*. These are self-defining parameters and consist of a one-byte **length** field, followed by a one-byte **identifier** field, followed by the data of the parameter.

The length field contains a count of the total number of bytes in the triplet, including the length and identifier fields. The identifier field identifies uniquely the type of the triplet.

A metafile is structured into a number of different functional components; for example, document and graphics object. Each component comprises a number of structured fields, and is delimited by "begin-component" and "end-component" structured fields. Structured fields marked as **required**, inside an **optional** structured field bracket, are required if the containing bracket is present.

The graphics orders that describe a picture occur in the **graphics data** structured field. See page G-16.

Structured Field Formats

The format of the various structured fields is given below:

Begin Document

Structured Field Introducer (BDT): required

```
0-1   Length X'n+1E'
2-4   BDT X'D3A8A8'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7   Document name   C'0000 0001'
8     Architecture version X'00'
9     Document security X'00'
```

Triplets (all required)

```
0     Length X'05'
1     Triplet Id X'18'
2     Interchange set type X'03' (resource document)
3-4   Base set definition X'0C00' (level 12, version 0)

0     Length X'06'
1     Triplet Id X'01'
2-5   GCID

0     Length X'n+1'
1     Triplet Id X'65'
2-n   Comment, used for metafile description of
      up to 252 bytes.
```

Begin Resource Group (BRG): required

Structured Field Introducer

```
0-1   Length X'0010'
2-4   BRG X'D3A8C6'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7   Resource group name   C'0000 0002'
```

Begin Color Attribute (BCA) Table: required

Structured Field Introducer

```
0-1   Length X'0010'
2-4   BCA X'D3A877'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7   Color table name C'0000 0004'
```

Color Attribute Table (CAT): required

Structured Field Introducer

```
0-1   Length X'n+8'
2-4   CAT X'D3B077'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

Base Part (required)

```
0  Flags
   0   Reserved B'0'
   1   Reset
       B'0' Do not reset to default
       B'1' Do reset to default
   2-7 Reserved B'000000'
1  Reserved X'00'
2  LCTID X'00'
```

Element list(s) (triple generating) are mutually-exclusive. One or other is required.

Element List (repeating)

```
0     Length of this parameter
1     Type X'01': element list
2     Flags X'00': reserved
3     Format
      X'01' RGB
4-6   Starting Index
      (Top Byte Truncated)
7     Size of RGB component1 X'08'
8     Size of RGB component2 X'08'
9     Size of RGB component3 X'08'
10    Number of bytes in each
      following color triple X'04'
11-m  Color triples
```

Triple Generating

```
0     Length of this parameter X'0A'
1     Type X'02': bit generator
2     Flags
      0 ABFlag
        B'0' Normal
      1-7 Reserved B'0000000'
3     Format
      X'01' RGB
4-6   Starting index (top byte truncated)
7     Size of RGB component1 X'08'
8     Size of RGB component2 X'08'
9     Size of RGB component3 X'08'
```

End Color Attribute (ECA) Table: required

Structured Field Introducer

```
0-1   Length X'0010'
2-4   ECA X'D3A977'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7   Color table name C'0000 0004'
```

Begin Image Object (BIM): optional, repeating

Structured Field Introducer
0-1 Length X'0010'
2-4 BIM X'D3A8FB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Image name C'xxxx xxxx'

Begin Resource Group (BRG): optional

Structured Field Introducer
0-1 Length X'0010'
2-4 BRG X'D3A8C6'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Resource group name C'xxxx xxxx'

Color Attribute Table (BCA): optional

Structured Field Introducer
0-1 Length X'0010'
2-4 BCA X'D3A877'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Color table name C'xxxx xxxx'

Color Attribute Table (CAT): required

Structured Field Introducer
0-1 Length
2-4 CAT X'D3B077'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters

Base Part
0 Flags X'00'
1 Reserved X'00'
2 LUTID

Element List (repeating)
0 Length of this parameter
1 Type X'01': element list
2 Flags X'00': reserved
3 Format X'01': RGB
4-6 Starting index
(top byte truncated)
7 Size of RGB component1 X'08'
8 Size of RGB component2 X'08'
9 Size of RGB component3 X'08'
10 Number of bytes in each
following color triple X'03'
11-n Color triples

End Color Attribute Table (ECA): required if BCA present

Structured Field Introducer
0-1 Length X'0010'
2-4 ECA X'D3A977'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Color Table name C'xxxx xxxx'

End Resource Group (ERG): required if BRG present

Structured Field Introducer
0-1 Length X'0010'
2-4 ERG X'D3A9C6'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Resource Group name C'xxxx xxxx'

Begin Object Environment Group (BOG): optional

Structured Field Introducer
0-1 Length X'0010'
2-4 BOG X'D3A8C7'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Object environment group
name C'xxxx xxxx'

Map Color Attribute (MCA) Table: required

Structured Field Introducer
0-1 Length X'001A'
2-4 MCA X'D3AB77'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-1 Length

Triplet (required)
0 Length X'0C'
1 Triplet type:
fully qualified name X'02'
2 Type: ref to
Begin Resource Object X'84'
3 ID X'00'
4-11 Color table name C'xxxx xxxx'

lcid (required)
0 Length X'04'
1 Triplet type:
resource local ID X'24'
2 Type color table resource X'07'
3 Local identifier (LUT-ID) X'01'

End Object Environment Group (EOG): required if BOG present

Structured Field Introducer
0-1 Length X'0010'
2-4 EOG X'D3A9C7'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Object Environment Group
name C'xxxx xxxx'

Image Data Descriptor (IDD): required

Structured Field Introducer
0-1 Length X'0011'
2-4 IDD X'D3A6FB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0 Unit of measure:
X'00' tens of inches
X'01' tens of centimeters
1-2 X resolution image points / UOM
3-4 Y resolution image points / UOM
5-6 X extent of image PS
7-8 Y extent of image PS

Image Picture Data (IPD): required

Structured Field Introducer
0-1 Length
2-4 IPD X'D3EEFB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters (all required and in this order, except that only one of Image LUT-ID and IDE structure is present)

Begin Segment
0 Type X'70': begin segment
1 Length of following X'00'

Begin Image Content
0 Type X'91': Begin Image Content
1 Length of following X'01'
2 Format X'FF'

Image Size
0 Type X'94': image size
1 Length of following X'09'
2 Units of measure X'02': logical
3-4 Horizontal resolution
5-6 Vertical resolution
7-8 Height in pels
9-10 Width in pels

Image Encoding
0 Type X'95': image encoding
1 Length of following X'02'
2 Compression algorithm X'03': none
3 Recording algorithm X'03':
bottom-to-top

Image IDE-Size
0 Type X'96': image IDE-Size
1 Length of following X'01'
2 Number of bits per element

Image LUT-ID
(For bit maps with other than 24 bits per pel)

0 Type X'97' Image LUT-ID
1 Length of following X'01'
2 LUT-ID

IDE Structure
(For bit maps with 24 bits per pel)

0 Type X'9B': IDE structure
1 Length of following X'08'
2 Flags:
0 ABFlag
B'0' Normal (Additive)
1-7 Reserved B'0000000'
3 Format
X'01' RGB
4-6 Reserved X'000000'
7 Size of element 1
8 Size of element 2
9 Size of element 3

Image Picture Data (IPD): required, repeating

Structured Field Introducer

0-1 Length
2-4 IPD X'D3EEFB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters

Image Data

0-1 Type X'FE92': image data
2-3 Length of following
4-n Image data (scan lines of bit maps)

End Image Content
(required, only present in last Image Picture Data)

0 Type X'93': End Image Content
1 Length of following X'00'

End Segment
(required, only present in last Image Picture Data)

0 Type X'71': end segment
1 Length of following X'00'

End Image Object (EIM): required if BIM present

Structured Field Introducer

0-1 Length X'0010'
2-4 EIM X'D3A9FB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters

0-7 Image name C'xxxx xxxx'

Begin Graphics Object (BGR): required

Structured Field Introducer
0-1 Length X'0010'
2-4 BGR X'D3A8BB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Graphics object name C'0000 0007'

Begin Object Environment Group (BOG): optional

Structured Field Introducer
0-1 Length X'0010'
2-4 BOG X'D3A8C7'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Object Environment Group
name C'0000 0007'

Map Color Attribute Table (MCA): required

Structured Field Introducer
0-1 Length X'0016'
2-4 MCA X'D3AB77'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-1 Length

Triplet (required)
0 Length X'0C'
1 Triplet type:
fully qualified name X'02'
2 Type: ref to
Begin Resource Object X'84'
3 ID X'00'
4-11 Color table name C'0000 0004'

Map Coded Font (MCF): required, for default font

Structured Field Introducer
0-1 Length X'20'
2-4 MCF X'D3AB8A'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-1 Length

Triplets (required)

Font name
0 Length X'0C'
1 Triplet type:
fully qualified name X'02'
2 Type: ref to coded font X'84'
3 ID X'00'
4-11 Coded font name: C'nnxx xxxx'
where n is X'FF'

lcid
0 Length X'04'
1 Triplet type:
Resource Local ID X'24'
2 Type: Coded Font Resource X'05'
3 Local identifier (LCID) X'00'

Font Binary GCID
0 Length X'06'
1 Triplet type: Font Binary GCID X'20'
2-5 GCID

Map Coded Font (MCF): optional, repeating, for loaded fonts

Structured Field Introducer
0-1 Length X'58'
2-4 MCF X'D3AB8A'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-1 Length

Triplets (required)

Font name
0 Length X'0C'
1 Triplet type:
fully qualified name X'02'
2 Type: ref to coded font X'84'
3 ID X'00'
4-11 Coded font name

lcid
0 Length X'04'
1 Triplet type:
Resource Local ID X'24'
2 Type: coded font resource X'05'
3 Local identifier (LCID)

Font Attributes
0 Length X'14'
1 Triplet type:
Font Descriptor X'1F'
2 Weight Class
3 Width Class

```
4-5   Font Height
6-7   Char Width
8     Descript Flags
9     Usage Codes
10    Family
11    Activity Class
12    Font Quality
13-14 CAP Height
15-16 X Height
17-18 Line Density
19    Use Flags

Font Binary GCID
0     Length X'06'
1     Triplet type:
      Font Binary GCID X'20'
2-5   GCID

Font Typeface
0     Length X'24'
1     Triplet type:
      fully qualified name X'02'
2     Type: ref to font typeface X'08'
3     ID X'00'
4-35  Font typeface C'xxx..xxx'
```

Map Data Resource (MDR): optional, repeating

```
Structured Field Introducer
0-1   Length X'1D'
2-4   MDR X'D3ABC3'
5     Flags X'00'
6-7   Segment sequence number X'0000'

Parameters
0-1 Length

Triplets (required)

Bit-map Name
0     Length X'0C'
1     Triplet type:
      fully qualified name X'02'
2     Type: ref to Image Object X'84'
3     ID X'00'
4-11  Image name C'xxxx xxxx'

Extended Resource lcid
0     Length X'07'
1     Triplet type:
      Extended Resource Local ID X'22'
2     Type: Image Resource X'10'
3-6   Bit-map handle
```

End Object Environment Group (EOG): required if BOG present

```
Structured Field Introducer
0-1   Length X'0010'
2-4   EOG X'D3A9C7'
5     Flags X'00'
6-7   Segment sequence number X'0000'

Parameters
0-7 Object Environment Group name C'0000 0007'
```

Graphics Data Descriptor (GDD): required

Structured Field Introducer

```
0-1   Length X'nnnn'
2-4   GDD X'D3A6BB'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters (all required and in this order)

```
0   X'F7' Specify GVM Subset
1   Length of following data X'07'
2   X'B0' drawing order subset
3-4 X'0000'
5   X'23' Level 3.2
6   X'01' Version 1
7   Length of following field X'01'
8   Coordinate types in data
    X'04' Intel16
    X'05' Intel32

0   X'F6' Set Picture Descriptor
1   Length of following data
2   Flags
    0 B'0' Picture in 2D
    1 Picture Dimensions
      B'0' Not absolute (PU_ARBITRARY PS)
      B'1' Absolute (example: PU_TWIPS PS)
    2 Picture Elements
      B'0' Not pels
      B'1' Pels (PU_PELS PS)
           (Bit 1 must also be set)
    3-7 B'00000'
3   X'00' Reserved
4   Picture frame size coordinate type
    X'04' Intel16
    X'05' Intel32
5   UnitsOfMeasure
    X'00' Ten inches
    X'01' Decimeter
6-11 or 6-17 (2 or 4 bytes) Resolution.
    GPS Units / UOM on x axis
    GPS Units / UOM on y axis
    GPS Units / UOM on z axis
12-23 or 18-41 (2 or 4 bytes) Window Size.
    GPS X left, X right
    GPS Y bottom, Y top
    GPS Z near, Z far

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Parameter Format X'08'
3-4 Mask X'E000'
5   Names X'8F'
6   Coordinates
    X'00' Picture in 2D
7   Transforms
    X'04' Intel16
    X'05' Intel32
8   Geometrics
    X'04' Intel16
    X'05' Intel32

0   X'21' Set Current Defaults
1   Length of following data
2   Set default viewing transform X'07'
3-4 Mask X'CC0C'
```

```
5   Names X'8F'
6-n M11, M12, M21, M22, M41, M42   Matrix
                                   elements


0   X'21' Set Current Defaults
1   Length of following data
2   Set default line attributes X'01'
3-4 Mask - OR of as many of the following
    bits as are required:
    X'8000'  Line type
    X'4000'  Line width
    X'2000'  Line end
    X'1000'  Line join
    X'0800'  Stroke width
    X'0008'  Line color
    X'0002'  Line mix
5   Flags
    X'0F' Set indicated default
                attributes to initial values.
                (Data field is not present
                in this instance).
    X'8F' Set indicated default attributes
                to specified values.
6-n Data - data values as required, in the
                following order if present.
    No space is reserved for attributes for
    which the corresponding mask flag was not set.
    (1 byte)  - Line type
    (1 byte)  - Line width
    (1 byte)  - Line end
    (1 byte)  - Line join
    (G bytes) - Stroke width
    (4 bytes) - Line color
    (1 byte)  - Line mix
              (G=2 or 4 depending on the Geometrics
               parameter of Set Default Parameter Format)

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Character Attributes X'02'
3-4 Mask - OR of as many of the following
    bits as are required:
    X'8000'  Character angle
    X'4000'  Character box
    X'2000'  Character direction
    X'1000'  Character precision
    X'0800'  Character set
    X'0400'  Character shear
    X'0040'  Character break extra
    X'0020'  Character extra
    X'0008'  Character color
    X'0004'  Character background color
    X'0002'  Character mix
    X'0001'  Character background mix
5   Flags
    X'0F' Set indicated default attributes to initial values.
                (Data field is not present in this case).
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in the following order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (2*G bytes) - Character angle
    (2*G + 4 bytes) - Character box
    (1 byte)    - Character direction
    (1 byte)    - Character precision
    (1 byte)    - Character set
    (2*G bytes) - Character shear
```

```
    (4 bytes)   - Character break extra
    (4 bytes)   - Character extra
    (4 bytes)   - Character color
    (4 bytes)   - Character background color
    (1 byte)    - Character mix
    (1 byte)    - Character background mix
        (G=2 or 4 depending on the Geometrics parameter of Set Default
         Parameter Format)


0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Marker Attributes X'03'
3-4 Mask - OR of as many of the following bits as are required:
    X'4000'  Marker box
    X'1000'  Marker precision
    X'0800'  Marker set
    X'0100'  Marker symbol
    X'0008'  Marker color
    X'0004'  Marker background color
    X'0002'  Marker mix
    X'0001'  Marker background mix
5   Flags
    X'0F' Set indicated default attributes to initial values.
                  (Data field is not present in this instance)
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in this order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (2*G bytes) - Marker box
    (1 byte)    - Marker precision
    (1 byte)    - Marker set
    (1 byte)    - Marker symbol
    (4 bytes)   - Marker color
    (4 bytes)   - Marker background color
    (1 byte)    - Marker mix
    (1 byte)    - Marker background mix
        (G=2 or 4 depending on the Geometrics parameter of Set Default
         Parameter Format)

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Pattern Attributes X'04'
3-4 Mask - OR of as many of the following bits as are required:
    X'0800'  Pattern set
    X'0100'  Pattern symbol
    X'0080'  Pattern reference point
    X'0008'  Pattern color
    X'0004'  Pattern background color
    X'0002'  Pattern mix
    X'0001'  Pattern background mix
5   Flags
    X'0F' Set indicated default attributes to initial values.
                  (Data field is not present in this instance)
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in this order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (1 byte)    - Pattern set
    (1 byte)    - Pattern symbol
    (2*G bytes) - Pattern reference point
    (4 bytes)   - Pattern color
    (4 bytes)   - Pattern background color
    (1 byte)    - Pattern mix
    (1 byte)    - Pattern background mix
        (G=2 or 4 depending on the Geometrics parameter of Set Default
         Parameter Format)
```

```
0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Image Attributes X'06'
3-4 Mask - OR of as many of these bits as are required:
    X'0008'  Image color
    X'0004'  Image background color
    X'0002'  Image mix
    X'0001'  Image background mix
5   Flags
    X'0F' Set indicated default attributes to initial values.
                  (Data field is not present in this instance)
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in this order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (4 bytes)   - Image color
    (4 bytes)   - Image background color
    (1 byte)    - Image mix
    (1 byte)    - Image background mix

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Viewing Window X'05'
3-4 Mask - OR of as many of the following bits as are required:
    X'8000'  x left limit
    X'4000'  x right limit
    X'2000'  y bottom limit
    X'1000'  y top limit
5   Flags
    X'0F' Set indicated default attributes to initial values.
                  (Data field is not present in this case).
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in the following order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (2*G bytes) - x left limit
    (2*G bytes) - x right limit
    (2*G bytes) - y bottom limit
    (2*G bytes) - y top limit
          (G=2 or 4 depending on the Geometrics parameter of Set
           Default Parameter Format)

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Arc Parameters X'0B'
3-4 Mask - OR of as many of the following bits as are required:
    X'8000'  P value
    X'4000'  Q value
    X'2000'  R value
    X'1000'  S value
5   Flags
    X'0F' Set indicated default attributes to initial values.
                  (Data field is not present in this case).
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in the following order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (G bytes)   - P value
    (G bytes)   - Q value
    (G bytes)   - R value
    (G bytes)   - S value
        (G=2 or 4 depending on the Geometrics parameter of Set
         Default Parameter Format)

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Pick Identifier X'0C'
3-4 Mask - OR of as many of the following bits as are required:
```

X'8000' Pick identifier
5 Flags
X'0F' Set indicated default attributes to initial values.
(Data field is not present in this case).
X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in the following order if present.
No space is reserved for attributes for which the corresponding Mask flag was not set.
(4 bytes) - Pick identifier

0 X'E7' Set Bit-map Identifier
1 Length of following data X'07'
2-3 Usage Flags X'8000'
4-7 Bit-map handle
8 Lcid

Graphics Data (GAD): optional, repeating

Structured Field Introducer
0-1 Length X'n+9'
2-4 GAD X'D3EEBB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters (maximum length in one structured field is 32759)

Graphics Segment (optional, repeating)
Segment data (including the Begin Segment parameter) can be split at any point between successive Graphics Data structured fields.
0 X'70' Begin Segment
1 Length of following data X'0E'
2-5 Segment identifier
6 Segment attributes (1)
0 B'1' Invisible
1 B'1' Propagate invisibility
2 B'1' Detectable
3 B'1' Propagate detectability
6 B'1' Dynamic
7 B'1' Fast chaining
7 Segment attributes (2)
0 B'1' Non-chained
3 B'1' Prolog
8-9 Segment data length (low-order 2 bytes)
10-13 Reserved
14-15 Segment data length (high-order 2 bytes)
16-n Graphics orders (see page 33-1)

End Graphics Object (EGR)

Structured Field Introducer
0-1 Length X'0010'
2-4 EGR X'D3A9BB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Graphics object name C'0000 0007'

End Resource Group (ERG): required

Structured Field Introducer
0-1 Length X'0010'
2-4 ERG X'D3A9C6'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Resource Group name C'0000 0002'

End Document (EDT): required

Structured Field Introducer
0-1 Length X'0010'
2-4 EDT X'D3A9A8'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Document name C'0000 0001'

Appendix H. Initialization File Information

Initialization files include information about printers, queues, and system preferences set by the user from the control panel. Applications can query this information by using the PrfQueryProfileData, PrfQueryProfileInt, PrfQueryProfileSize, and PrfQueryProfileString functions.

All data in initialization files is accessed by a two-level hierarchy of application name, and key name within an application. Presentation Manager system data is keyed off "applications" that have names starting with PM_.

The application name/key name combinations that applications may need to use are listed below, together with the definition of the corresponding data.

Note: Information that is prefixed with PM_SPOOLERxxxx can not always be modified directly: The spooler validates all attempts to write information to the INI file that it depends on.

Application name "PM_ControlPanel"
Key name "Beep"
Type integer
Content/value 1 or 0.

Application name "PM_ControlPanel"
Key name "LogoDisplayTime"
Type integer
Content/value -1 ≤ time ≤ 32767 milliseconds.

Indefinite display	-1
No display	0
Timed display	>0

Application name "PM_National"
Key name "iCountry"
Type integer
Content/value country code:

Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358
French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82
Latin-American	3
Netherlands	31
Norwegian	47
Portuguese	351
Simpl. Chinese	86
Spanish	34
Swedish	46
Swiss	41
Trad. Chinese	88
UK-English	44
US-English	1
Other country	0.

Application name "PM_National"
Key name "iDate"
Type integer
Content/value 0=MDY; 1=DMY; 2=YMD.

Application name "PM_National"
Key name "iCurrency"
Type integer
Content/value Values have the following meanings:

- **0** Prefix, no separator
- **1** Suffix, no separator
- **2** Prefix, 1 character separator
- **3** Suffix, 1 character separator.

Application name "PM_National"
Key name "iDigits"
Type integer
Content/value n = number of decimal digits.

Application name "PM_National"
Key name "iTime"
Type integer
Content/value 0 = 12-hour clock; 1 = 24-hour clock.

Application name "PM_National"
Key name "iLzero"
Type integer
Content/value 0 = no leading zero; 1 = leading zero.

Application name "PM_National"
Key name "s1159"
Type string
Content/value "am" for example. 3 chars max.

Application name "PM_National"
Key name "s2359"
Type string
Content/value "pm" for example. 3 chars max.

Application name "PM_National"
Key name "sCurrency"
Type string
Content/value "$" for example. 3 chars max.

Application name "PM_National"
Key name "sThousand"
Type string
Content/value "," for example. 1 char max.

Application name "PM_National"
Key name "sDecimal"
Type string
Content/value "." for example. 1 char max.

Application name "PM_National"
Key name "sDate"
Type string
Content/value "/" for example. 1 char max.

Application name "PM_National"
Key name "sTime"
Type string
Content/value ":" for example. 1 char max.

Application name "PM_National"
Key name "sList"
Type string
Content/value "," for example. 1 char max.

Application name PM_Fonts
Key name <Font module name>
Type string
Content/value fully-qualified drive:\path\filename.ext.

Application name "PM_SPOOLER"
Key name "QUEUE"
Type string
Content/value <Queue name>;

where:

- <Queue name> is the name of the default queue (might be NULL). This must be a key name for the PM_SPOOLER_QUEUE application.

Application name "PM_SPOOLER"
Key name "PRINTER"
Type string
Content/value <Printer name>;

where:

- <Printer name> is the name of the default printer (might be NULL).

Note: Use the SplQueryDevice and SplQueryQueue functions to retrieve the spooler configuration data.

Appendix I. Virtual Key Definitions

The PC VKEY set is shown in the following table:

Symbol	Personal Computer AT Keyboard	Enhanced Keyboard
VK_BUTTON1 VK_BUTTON2 VK_BUTTON3	These values are only used to access the up/down and toggled states of the pointing device buttons; they never actually appear in a WM_CHAR message.	These values are only used to access the up/down and toggled states of the pointing device buttons; they never actually appear in a WM_CHAR message.
VK_BREAK	Ctrl + Scroll Lock	Ctrl + Pause
VK_BACKSPACE	Backspace	Backspace
VK_TAB	Tab	Tab
VK_BACKTAB	Shift + Tab	Shift + Tab
VK_NEWLINE	Enter	Enter
VK_SHIFT *	Left and Right Shift	Left and Right Shift
VK_CTRL *	Ctrl	Left and Right Ctrl
VK_ALT *	Alt	Left and Right Alt
VK_ALTGRAF *	None	Alt Graf (if available)
VK_PAUSE	Ctrl + Num Lock	Pause
VK_CAPSLOCK	Caps Lock	Caps Lock
VK_ESC	Esc	Esc
VK_SPACE *	Space	Space
VK_PAGEUP *	Numpad 9	Pg Up and Numpad 9
VK_PAGEDOWN *	Numpad 3	Pg Dn and Numpad 3
VK_END *	Numpad 1	End and Numpad 1
VK_HOME *	Numpad 7	Home and Numpad 7
VK_LEFT *	Numpad 4	Left and Numpad 4
VK_UP *	Numpad 8	Up and Numpad 8
VK_RIGHT *	Numpad 6	Right and Numpad 6
VK_DOWN *	Numpad 2	Down and Numpad 2
VK_PRINTSCRN	Shift + Print Screen	Print Screen
VK_INSERT *	Numpad 0	Ins and Numpad 0
VK_DELETE *	Numpad .	Del and Numpad .
VK_SCRLLOCK	Scroll Lock	Scroll Lock
VK_NUMLOCK	Num Lock	Num Lock
VK_ENTER	Shift + Enter	Shift + Enter and Numpad Enter
VK_SYSRQ	SysRq	Alt + Print Screen
VK_F1 *	F1	F1
VK_F2 *	F2	F2

VK_F3 *	F3	F3
VK_F4 *	F4	F4
VK_F5 *	F5	F5
VK_F6 *	F6	F6
VK_F7 *	F7	F7
VK_F8 *	F8	F8
VK_F9 *	F9	F9
VK_F10 *	F10	F10
VK_F11 *	None	F11
VK_F12 *	None	F12
VK_F13	None	None
VK_F14	None	None
VK_F15	None	None
VK_F16	None	None
VK_F17	None	None
VK_F18	None	None
VK_F19	None	None
VK_F20	None	None
VK_F21	None	None
VK_F22	None	None
VK_F23	None	None
VK_F24	None	None
VK_MENU *	F10	F10

Notes:

1. VKEYs marked with an asterisk (*) are generated irrespective of other shift states (Shift, Ctrl, Alt, and Alt Graf).
2. VK_CAPSLOCK is **not** generated for any of the Ctrl shift states, for PC-DOS compatibility.
3. Wherever possible, the VK_ name is derived from the legend on the key top of the 101-key Enhanced PC keyboard.

Glossary

A

accelerator. A single key stroke that invokes an application-defined function.

accelerator table. Used to define which key strokes are treated as *accelerators* and the commands they are translated into.

access permission. All access rights that a user has regarding an object.

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. The area at the top of a window that contains the choices currently available in the application program.

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. See also *interactive program*, *noninteractive program*, and *foreground program*.

active window. The window with which the user is currently interacting.

address space. (1) The range of addresses available to a program. (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.

anchor block. An area of Presentation Manager-internal resources allocated to a 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. The formally-defined programming language that is between an IBM application program and the user of the program. See also *GPI*.

area. In computer graphics, a filled shape such as a solid rectangle.

ASCII. American National Standard Code for Information Interchange. A coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

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. (1) Without regular time relationship. (2) Unexpected or unpredictable with respect to the execution of a program's instructions. See also *synchronous*.

atom. A constant that represents a string. Once 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. 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.

attributes. Characteristics or properties that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes*.

AVIO. Advanced Video Input/Output.

B

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*.

Bézier curves. A mathematical technique of specifying smooth continuous 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. Bezier.

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. (2) To combine two or more data elements in one block.

border. A visual indication (for example, a separator line or a background color) of the boundaries of a window.

breakpoint. (1) An instruction in a program for halting execution. Breakpoints are usually established at positions in a program where halts, caused by external intervention, are convenient for restarting. (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.

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.

button. A mechanism on a *pointing device*, such as a mouse, used to request or initiate an action. Contrast with *pushbutton* and *radio button*.

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.

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. (2) To transfer control to a procedure, program, routine, or subroutine.

calling order. A sequence of instructions together with any associated data necessary to perform a call. Also known as *calling sequence*.

cancel. An action that removes the current window or menu without processing it, and returns the previous window.

CASE statement. In C, provides the body of a window procedure. There is one CASE statement for each message type written to take specific actions.

cell. See *character cell*.

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. Synonym for *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 mode. The character mode, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

check box. A control window, shaped like a square button on the screen, that can be in a checked or unchecked state. It is used to select one or more items from a list. Contrast with *radio button*.

check mark. The symbol that is used to indicate a selected item on a pull-down.

child process. A process that is loaded and started by another process. Contrast with *parent process*.

child window. A window that is positioned relative to another window (either a main window or another child window). Contrast with *parent window*.

choice. 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).

class. See *window class*.

class style. The set of properties that apply to every window in a window class.

client area. The area in the center of a window that contains the main information of the window.

clipboard. An area of main storage that can hold data being passed from one PM application to another. Various data formats can be stored.

clipping. In computer graphics, removing those parts of a display image that lie outside a given boundary.

clip limits. The area of the paper that can be reached by a printer or plotter.

clipping path. A clipping boundary in world-coordinate space.

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. Synonym for *character code*.

code segment. An executable section of programming code within a load module.

color dithering. See *dithering*.

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 usually at the bottom of the screen, in which only commands can be entered.

command prompt. A field prompt showing the location of the command entry field in a panel.

Common Programming Interface (CPI). A consistent set of specifications for languages, commands, and calls to enable applications to be developed across all SAA environments. See also *Systems Application Architecture*.

Common User Access (CUA). A set of rules that define the way information is presented on the screen, and the techniques for the user to interact with the information.

compile. To translate a program written in a higher-level programming language into a machine language program.

COM1, COM2, COM3. Character-device names reserved for serial ports 1 through 3.

CON. Character-device name reserved for the console keyboard and screen.

contiguous. Touching or joining at a common edge or boundary, for example, an unbroken consecutive series of storage locations.

control. The means by which an operator gives input to an application. A *choice* corresponds to a control.

Control Panel. In PM, a program used to set up user preferences that act globally across the system.

Control Program. The basic function of OS/2, including DOS emulation and the support for keyboard, mouse, and video input/output.

control window. A class of window used to handle a specific kind of user interaction. Radio buttons and check boxes are examples.

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.

CPI. Common Programming Interface.

critical extended attribute. An extended attribute that is necessary for the correct operation of the system or a particular application.

CUA. Common User Access.

current position. The point from which the next primitive will be drawn.

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*.

D

data structure. (ISO) The syntactic structure of symbolic expressions and their storage-allocation characteristics.

DBCS. See *double-byte character set*.

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.

decipoint. In printing, one tenth of a point. There are 72 points in an inch.

default procedure. Function provided by the Presentation Interface that may be used to process standard messages from dialogs or windows.

default value. A value used when no value is explicitly specified by the user. For example, in the graphics programming interface, the default line-type is 'solid'.

descendant. A process or session that is loaded and started by a parent process or parent session.

Desktop Manager. In PM, 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.

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. Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units.

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. A type of window that contains one or more controls for the formatted display and entry of data. Also known as a *pop-up window*. A modal dialog box is used to implement 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 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 *File Manager*.

direct memory access (DMA). The transfer of data between main storage and input/output devices without intervention by the processor.

directory. A type of file containing the names and controlling information for other files or other directories.

display point. Synonym for *pel*.

dithering. 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 on narrow lines, for example.

DMA. Direct memory access.

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.

dragging. In computer graphics, moving an object on the display screen as if it were attached to the pointer.

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.

DTL. See *dialog tag language*.

dual-boot function. A feature of OS/2 that allows the user to start DOS from within OS/2, or OS/2 from within DOS.

duplex. Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full duplex*.

dynamic linking. The process of resolving external references in a program module at load time or run time rather than during linking.

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-link module. A module that is linked at load time or run time.

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. (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. (3) See also *static storage*.

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.

EGA. Extended graphics adapter.

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*.

element. An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

entry field. An area on the screen, usually highlighted in some manner, in which users type information.

entry-field control. The means by which the application receives data entered by the user in an entry field. When it has the input focus, it displays a flashing pointer at the position where the next typed character will go.

entry panel. A defined panel type containing one or more entry fields and protected information such as headings, prompts, and explanatory text.

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.

exit. The action that terminates the current function and returns the user to a higher level function. Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel*.

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-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*.

extended help. A facility that provides users with information about an entire application panel rather than a particular item on the panel.

extent. Continuous space on a disk or diskette that is occupied by or reserved for a particular data set, data space, or file.

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.

file. A named set of records stored or processed as a unit.

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 PM, 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 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*.

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.

focus. See *input focus*.

font. A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

foreground program. The program with which the user is currently interacting. Also known as *interactive program*. Contrast with *background program*.

frame. The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by PM. The frame encloses the client area.

frame styles. Different standard window layouts provided by PM.

FSD. File system driver.

full duplex. Synonym for *duplex*.

full-screen application. An application program that occupies the whole 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). 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. A special character used to refer to a set of file objects with a common base name. The asterisk (*) and question mark (?) are used as global file-name characters. For example, *.EXE can be used to refer to a set of files with the extension EXE.

glyph. A graphic symbol whose appearance conveys information.

GPI. Graphics programming interface. The formally-defined programming language that is between an IBM graphics program and the user of the program. See also *API*.

graphics. A picture defined in terms of graphic primitives and graphics attributes.

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 model space. The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as *model space*.

graphic primitive. A single item of drawn graphics, such as a line, arc, or graphics text string. See also *graphics segment*.

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. See also *program group*.

H

handle. An identifier that represents an object, such as a device or window, to the Presentation Interface.

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.

help. A function that provides information about a specific field, an application panel, or information about the help facility.

help index. A facility that allows the user to select topics for which help is available.

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.

heap. An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

hit testing. The means of identifying which window is associated with which input device event.

hook. A mechanism by which procedures are called when certain events occur in the system. For example, the filtering of mouse and keyboard input before it is received by an application program.

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*.

I

icon. A pictorial representation of an item the user can select. 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 PM, icons are used for data objects, system actions, and minimized programs.

icon area. In PM, 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 the symbol. Contrast with *outline font*.

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.

input focus. The area of the screen that will receive input from an input device (typically the keyboard).

input router. An internal OS/2 process that removes messages from the system queue.

integer atom. A special kind of *atom* that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by PM are expressed as integer atoms.

interactive graphics. Graphics that can be moved or manipulated by a user at a terminal.

interactive program. A program that is running (active) and is ready to receive (or is receiving) input from the user. Compare with *active program* and contrast with *noninteractive program*.

Also known as a *foreground program*.

interchange file. Data that can be sent from one Presentation Interface application to another.

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. A device-specific command that requests a function of a device driver through the *DosDevIOCtl* function.

I/O operation. An input operation to, or output operation from a device attached to a computer.

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.*

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.

keys help. A facility that gives users a listing of all the key assignments for the current application.

L

label. In a graphics segment, an identifier of one or more elements that is used when editing the segment.

language support procedure. Function provided by the Presentation Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

LDT. Local Descriptor Table.

LIFO stack. A data stack from which data is retrieved in last-in, first-out order.

linked list. Synonym for *chained list.*

list box. A control window containing a vertical list of selectable descriptions.

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-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.

load time. The point in time at which a program module is loaded into main storage for execution.

local area network (LAN). A data network located on the user's premises in which serial transmission is used for direct data communication among data stations.

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.

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.*

map. (1) A set of values having a defined correspondence with the quantities or values of another set. (2) To establish a set of values having a defined correspondence with the quantities or values of another set.

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.

maximize. A window-sizing action that makes the window the largest size possible.

media window. The part of the physical device (display, printer, or plotter) on which a picture is presented.

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.

menu. A type of panel that consists of one or more selection fields. Also called a *menu panel.*

message. (1) In PM, a packet of data used for communication between the Presentation Interface and windowed 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 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.

metafile. The generic name for the definition of the contents of a picture. Metafiles are used to allow pictures to be used by other applications.

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. A window-sizing action that makes the window the smallest size possible. In PM, minimized windows are represented by icons.

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. A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item.

modal dialog box. The type of control that allows the operator to perform input operations on only the current dialog box or one of its child windows. Also known as a *serial dialog box*. Contrast with *parallel dialog box*.

modeless dialog box. The type of control that allows the operator to perform input operations on any of the application's windows. Also known as a *parallel dialog box*. Contrast with *modal dialog box*.

model space. See *graphics model space*.

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. A hand-held device that is moved around to position the pointer on the screen.

MOUSE$. Character-device name reserved for a mouse.

multiple-choice selection. A mode that allows users to select any number of choices, including none at all. See also *check box*. Contrast with *extended-choice selection*.

multitasking. The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

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*.

noncritical extended attribute. An extended attribute that is not necessary for the function of an application.

nondestructive read. A read process that does not erase the data in the source location.

non-8.3 file-name format. A file-naming convention in which path names can consist of up to 255 characters. See also *8.3 file-name format*.

noninteractive program. A program that is running (active) but is not ready to receive input from the user. Compare with *active program*, and contrast with *interactive program*.

nonretained graphics. Graphic primitives that are not remembered by the Presentation Interface once they have been drawn. Contrast with *retained graphics*.

NUL. Character-device name reserved for a nonexistent (dummy) device.

null-terminated string. A string of (n + 1) characters where the (n + 1)th character is the 'null' character (X'00'), and is used to represent an n-character string with implicit length. Also known as 'zero-terminated' string and 'ASCIIZ' string.

O

object window. A window that does not have a parent, but which may have child windows. An object window cannot be presented on a device.

open. To start working with a file, directory, or other object.

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. The area of the output device within which the picture is to be displayed, printed, or plotted.

owner window. A window into which specific events that occur in another (owned) window are reported.

owning process. The process that owns the resources that may be shared with other processes.

P

page. A 4KB segment of contiguous physical memory.

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. The action of drawing or redrawing the contents of a window.

panel. A particular arrangement of information grouped together for presentation to the user in a window.

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 body. The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body may contain protected information, selection fields, and entry fields. The layout and content of the panel body determine the panel type.

panel body area. The part of a window not occupied by the action bar or function key area. The panel body area may contain information, selection fields, and entry fields. Also known as *client area*.

panel body area separator. A line or color boundary that provides users with a visual distinction between two adjacent areas of a panel.

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. A panel element located in the upper left-hand corner of a panel body that identifies that particular panel within the application.

panel title. A panel element that identifies the information in the panel.

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*.

parent process. A process that loads and starts other processes. Contrast with *child process*.

parent window. The window relative to which one or more child windows are positioned. 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.

path. The part of a file specification that lists a series of directory names. Each directory name is separated by the backslash character. In the file specification C:\MYFILES\MISC\GLOSSARY.SCR, the path consists of MYFILES\MISC\.

pel. 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*.

pick. To select part of a displayed object using the pointer.

picture chain. See *segment chain*.

picture element. Synonym for *pel*.

PID. Process identification.

pipe. 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*.

pixel. Synonym for *pel*.

plotter. An output device that uses pens to draw its output on paper or on transparency foils.

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.

POINTER$. Character-device name reserved for a pointer device (mouse screen support).

pointing device. A device (such as a mouse) 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. It 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*.

polyline. A sequence of adjoining lines.

pop. To retrieve an item from a last-in-first-out stack of items. Contrast with *push*.

pop-up window. 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.

Presentation Manager (PM). The visual component of OS/2 that presents, in windows, a graphics-based interface to applications and files installed and running in OS/2.

presentation page. The coordinate space in which a picture is assembled for display.

presentation space (PS). Contains the device-independent definition of a picture.

primary window. The window in which the main dialog 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. See *graphic primitive*.

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 PM, 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 details. Information about a program that is specified in the *Program Manager* window and is used when the program is started.

program group. In PM, 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. The user must respond to the prompt in order to proceed.

protocol. A set of semantic and syntactic rules that determines the behavior o functional units in achieving communication.

pseudocode. An artificial language used to describe computer program algorithms without using the syntax of any particular programming language.

pull-down. 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.

push. To add an item to a last-in-first-out stack of items. Contrast with *pop*.

pushbutton. A control window, shaped like a round-cornered rectangle and containing text, that invokes an immediate action, such as 'enter' or 'cancel'.

Q

queue. A linked list of elements waiting to be processed. For example, a queue may be a list of print jobs waiting to be printed.

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. 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 list. Contrast with *check box*.

RAS. Reliability, availability, and serviceability.

raster. (1) In computer graphics, a predetermined pattern of lines that provides uniform coverage of a display space. (2) The coordinate grid that divides the display area of a display device.

read-only file. A file that may be read from but not written to.

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 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. 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.

reference phrase help. Provides help information for a selectable word or phrase.

refresh. To update a window, with changed information, to its current status.

region. A clipping boundary in device space.

register. A storage device having a specified storage capacity such as a bit, byte, or computer word, and usually intended for a special purpose.

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 Interface after they have been drawn. Contrast with *nonretained graphics*.

return code. (1) A code used to influence the execution of succeeding instructions. (2) A value returned to a program to indicate the results of an operation requested by that program.

reverse video. A form of alphanumeric highlighting for a character, field, or cursor, in which its color is

exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

RGB. Red-green-blue. For example, "RGB display".

roman. Relating to a type style with upright characters.

root segment. In a hierarchical database, the highest segment in the tree structure.

run time. (1) Any instant at which a program is being executed. (2) The time during which an instruction in an instruction register is decoded and performed.

S

SAA. Systems Application Architecture.

scheduler. A computer program designed to perform functions such as scheduling, initiation, and termination of jobs.

screen. The physical surface of a work station or terminal upon which information is presented to users.

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. A control window, horizontally or vertically aligned, that allows the user to scroll additional data into an associated panel area.

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 type of window associated with the primary window in a dialog. A secondary window begins a secondary and parallel dialog that runs at the same time as the primary dialog.

sector. An addressable subdivision of a track used to record one block of program code or data on a disk or diskette.

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. A type of cursor used to indicate the choice or entry field users want to interact with. It is represented by highlighting the item that it is currently positioned on.

selection field. A field containing a list of choices from which the user can select one or more.

semaphore. An object used by multi-threaded applications for signalling purposes and for controlling access to serially reusable resources.

separator. See *panel body area separator*.

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. 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.

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. Memory that is used by two or more programs.

shear. The tilt of graphics text when each character leans to the left or right while retaining a horizontal baseline.

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 procedure required before the computer is switched off to ensure that data is not lost.

sibling processes. Child processes that have the same parent process.

sibling windows. Child windows that have the same parent window.

slider box. An area on the scroll bar that indicates the size and position of the visible information in a panel area in relation to the information available. Also known as *thumb mark*.

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.

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. Static storage may use dynamic addressing or sensing circuits. (2) Storage other than *dynamic storage*.

style. See *window style*.

suballocation. The allocation of a part of one extent for occupancy by elements of a component other than the one occupying the remainder of the extent.

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. (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) An action that moves the input focus from one area to another. This can be within the same window or from one window to another. (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, that is used to identify a programming object.

synchronous. Pertaining to events or operations that are predictable or occur at the same time. See also *asynchronous*.

System Menu. In PM, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

system queue. This is the master queue for all pointer device or keyboard events.

Systems Application Architecture (SAA). A formal set of rules that enables applications to be run without modification in different computer environments.

T

tag. One or more characters attached to a set of data that defines the formatting or other characteristics of the set, including its definition.

Task List. In PM, 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.

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&hyphn.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*.

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.

title bar. The area at the top of a window that contains the window title. The title bar is highlighted when that window has the input focus. Contrast with *panel title*.

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*.

Tree. In PM, the window in the *File Manager* that shows the organization of drives and directories.

truncate. (1) To end a computational process in accordance with some rule. (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.

U

unnamed pipe. A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with *named pipe*.

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 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. (2) A program designed to perform an everyday task such as copying data from one storage device to another.

V

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. Clipping boundary that defines the visible part of model space.

VIO. Video Input/Output.

virtual memory (VM). Addressable space that is apparent to the user as the processor storage space, but not having a fixed physical location.

virtual storage. 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. (2) A portion of data, together with its data carrier, that can be handled conveniently as a unit.

W

wild-card character. The global file-name characters asterisk (*) and question mark (?).

window. A rectangular 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. The means by which a window position or size is defined; measured in device units, or *pels*.

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 style. The set of properties that influence how events related to a particular window will be processed.

workstation. A display screen together with attachments such as a keyboard, a local copy device, or a tablet.

world coordinates. Application-convenient coordinates used for drawing graphics.

world-coordinate space. Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG. What You See Is What You Get. A capability that enables text to be displayed on a screen in the same way it will be formatted on a printer.

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. In graphics applications, the process of increasing or decreasing the size of picture.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

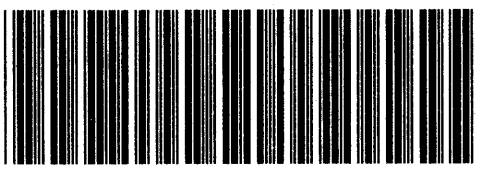
T

U

V

W

X



S10G-6272-00

International Business
Machines Corporation

Printed in the
United States of America

10G6272

P10G6272