Standard ECMA-234
December 1995

ECMA

Standardizing Information and Communication Systems

Application Programming Interface for Windows

Volume 3

Annexes

Standard ECMA-234

December 1995

ECMA

Standardizing Information and Communication Systems

Application Programming Interface for Windows

GL E-234-V3.DOC 07-03-96 16,08

Brief History

The APIW Standard is a functional specification of the Microsoft Windows 3.1 application programming interface. It is based on existing implementations (including Microsoft and others) and behavior. The goal of writing this specification is to define an environment in which:

– applications written to this baseline will be portable to all implementations of the APIW Standard.

– the interface can be enriched through open standards processes to meet current and future user needs in a timely fashion.

APIW uses the current C language binding, and reflects existing coding practices to ensure that current applications will conform to this standard. The APIs documented in this standard shall accurately reflect existing implementations of the windows APIs. If an application that runs with an existing implementation uses one or more APIs contrary to the way it is described in the standard, the standard will be changed to accurately reflect the behavior.

The APIW Standard defines a set of application programming interfaces that allow for the creation of graphical applications spanning a wide range of capabilities. The standard groups these APIs into major functional areas including a window manager interface, a graphics device interface and interfaces necessary for accessing system resources and capabilities. The API requirements of today's major desktop applications are reflected in this specification and are the criteria for determining the APIW content.

The APIW Standard focuses on providing the necessary APIs for writing applications for the desktop, and also allows additional APIs to be bound to an application. This feature enables services outside the scope of a standard desktop application to be provided, for example, database, networking or other system services.

The APIW Standard defines the basic graphical use interface objects, such as buttons, scrollbars, menus, static and edit controls, and the painting functions to draw them, such as area fill, and line and rectangle drawing. Finally, a rich set of text routines in defined, from simple text output to more complex text output routines using multiple founts and font styles, all supporting the use of color.

The APIW Standard is documented in five sections, corresponding loosely to the four functional subsystems represented by the API and the conformance clause. The four APIW sections cover window management, graphical interface, system services and an application support services section. These functions cover window creation and management, graphics routines to paint text and other graphics objects in those windows, functions to access system resources such as files and timers, and finally, common support functions to accelerate the development of graphical window-based applications.

The APIW Window Subsystem section of the standard covers the creation, deletion and management of the window, including window positioning and sizing and the sending and receiving of messages. Within each of these window management subsections are routines that significantly extend the basic functions. With window creation, there are many types of windows that can be created including built-in classes and user-definable classes, that have the ability to modify the style of any one of the built-in classes. Additional functions are defined to affect the display of a window, including functions to modify the windows menu, scrollbars, and the display of carets or cursors within the window. With multiple overlapped windows being displayed simultaneously, functions are defined to manage the position and size of those windows, as well as to control the visibility of a window and its associated icon when it is minimized.

The APIW Window Subsystem section also defines a set of functions for managing a subset of the user interface, referred to as dialog boxes. These functions allow for the creation and management of the dialog box, as well as the user interaction with the dialog box up to its closure. Utility functions are defined to make designing and using a dialog box easier. These utilities provide common dialog box functions, such as group boxes and check boxes, as well as file interface functions to list files and directories. Each of these dialog boxes are controlled by the use of dialog box templates that are stored in resource files.

The APIW Graphics Subsystem section covers all aspects of actually drawing in a window. These aspects include line drawing, text output, graphics primitives, such as rectangles and ellipses, as well as more sophisticated routines such as *floodfill()*, *bitblts()* and *stretchblt()*. The Graphics Device Interface defines bitmaps, icons, cursors and carets, as well as functions to provide for a portable graphics file format called metafiles. The Graphics Device Interface defines a logical coordinate space to further abstract the underlying hardware and has functions to map between the logical and physical coordinate space. The Graphics Device Interface defines utility functions for all drawing routines that use pens, brushes and regions to get precise control over how graphical objects will be drawn.

The APIW System Services section defines platform-independent routines for an application to query the system environment and access system services. System services that may be accessed include memory, timers, the keyboard and the native file system. There are subsections that deal with resources, device I/O and system diagnostic routines. Resource management allows for the loading and unloading of user- and system-defined resources, such as icons, bitmaps and strings. Device I/O includes both parallel and serial port input and output operations. System diagnostic routines enable an application or diagnostic tool to examine the state of an application, including memory utilization, task information and stack usage.

The APIW Application Support Function section defines miscellaneous functions that can be used by a developer in an application. These utility functions define built-in services that a developer does not have to rewrite with each application. These service functions include debugging routines and simple user interface routines to provide graphical feedback to a user. They also include routines for file compression and decompression, standardized routines to retrieve application version information and routines to manage initialization files.

Adopted as an ECMA Standard by the General Assembly of December 1995.

Table of contents

Annex A

Supported Windows 3.1 Functions

A.1 Description

The following table is an alphabetical list of the supported Windows 3.1 functions.

_lclose	BringWindowToTop	CreateBitmap
_lcreate	BuildCommDCB	CreateBitmapIndirect
_llseek	CallMsgFilter	CreateBrushIndirect
_lopen	CallNextHookEx	CreateCaret
_lread	CallWindowProc	CreateCompatibleBitmap
_lwrite	CallWndProc	CreateCompatibleDC
AbortDoc	Catch	CreateCursor
AbortProc	ChangeClipboardChain	CreateDC
AddAtom	CheckDlgButton	CreateDialog
AddFontResource	CheckMenuItem	CreateDialogIndirect
AdjustWindowRect	CheckRadioButton	CreateDialogIndirectParam
AdjustWindowRectEx	ChildWindowFromPoint	CreateDialogParam
AllocResource	ChooseColor	CreateDIBitmap
AnimatePalette	ChooseFont	CreateDIBPatternBrush
AnsiLower	Chord	CreateDiscardableBitmap
AnsiLowerBuf	ClearCommBreak	CreateEllipticRgn
AnsiNext	ClientToScreen	CreateEllipticRgnIndirect
AnsiPrev	ClipCursor	CreateFont
AnsiToOem	CloseClipboard	CreateFontIndirect
AnsiToOemBuff	CloseComm	CreateHatchBrush
AnsiUpper	CloseMetaFile	CreateIC
AnsiUpperBuf	CloseWindow	CreateIcon
AnyPopup	CombineRgn	CreateMenu
AppendMenu	CommDlgExtendedError	CreateMetaFile
Arc	CopyCursor	CreatePalette
ArrangeIconicWindows	CopyIcon	CreatePatternBrush
BeginDeferWindowPos	CopyMetaFile	CreatePen
BeginPaint	CopyRect	CreatePenIndirect
BitBlt	CountClipboardFormats	CreatePolygonRgn

CreatePolyPolygonRgn
CreatePopupMenu
CreateRectRgn
CreateRectRgnIndirect
CreateRoundRectRgn
CreateSolidBrush
CreateWindow
CreateWindowEx
DebugOutput
DefDlgProc
DeferWindowPos
DefFrameProc
DefHookProc
DefMDIChildProc
DefWindowProc
DeleteAtom
DeleteDC
DeleteMenu
DeleteMetaFile
DeleteObject
DestroyCaret
DestroyCursor
DestroyIcon
DestroyMenu
DestroyWindow
DeviceCapabilities
DeviceMode
DialogBox
DialogBoxIndirect
DialogBoxIndirectParam
DialogBoxParam
DialogProc
DirectedYield
DispatchMessage
DlgDirList
DlgDirListComboBox
DlgDirSelect
DlgDirSelectComboBox
DlgDirSelectComboBoxEx
DlgDirSelectEx
DPtoLP
DrawFocusRect
DrawIcon
DrawMenuBar
DrawText
Ellipse
EmptyClipboard
EnableCommNotification
EnableMenuItem
EnableScrollBar
EnableWindow
EndDeferWindowPos
EndDialog
EndDoc
EndPage
EndPaint
EnumChildProc
EnumChildWindows
EnumClipboardFormats
EnumFontFamProc
EnumFontProc
EnumFonts
EnumFontsFamilies
EnumMetaFile
EnumMetaFileProc
EnumObjects
EnumObjectsProc
EnumProps
EnumPropsProc
EnumTaskWindows
EnumTaskWndProc
EnumWindows
EnumWindowsProc
EqualRect
EqualRgn
Escape
EscapeCommFunction
ExcludeClipRect
ExcludeUpdateRgn
ExitWindows
ExtDeviceMode
ExtFloodFill
ExtractIcon
ExtTextOut
FatalAppExit
FatalExit
FillRect
FillRgn
FindAtom
FindExecutable
FindResource
FindText
FindWindow
FlashWindow
FloodFill
FlushComm
FrameRect
FrameRgn
FreeLibrary

FreeModule	GetCommEventMask	GetKeyNameText
FreeProcInstance	GetCommState	GetKeyState
FreeResource	GetCurrentPosition	GetLastActivePopup
GetActiveWindow	GetCurrentPositionEx	GetMapMode
GetAspectRatioFilter	GetCurrentTask	GetMenu
GetAspectRatioFilterEx	GetCurrentTime	GetMenuCheckMarkDimensions
GetAsyncKeyState	GetCursor	GetMenuItemCount
GetAtomName	GetCursorPos	GetMenuItemID
GetBitmapBits	GetDC	GetMenuState
GetBitMapDimension	GetDCEx	GetMenuString
GetBitMapDimensionEx	GetDCOrg	GetMessage
GetBkColor	GetDeskTopWindow	GetMessageExtraInfo
GetBkMode	GetDeviceCaps	GetMessagePos
GetBoundsRect	GetDialogBaseUnits	GetMessageTime
GetBrushOrg	GetDIBits	GetMetaFile
GetBrushOrgEx	GetDlgCtrlID	GetMetaFileBits
GetCapture	GetDlgItem	GetModuleFileName
GetCaretBlinkTime	GetDlgItemInt	GetModuleHandle
GetCaretPos	GetDlgItemText	GetModuleUsage
GetCharABCWidths	GetDoubleClickTime	GetMsgProc
GetCharWidth	GetDriveType	GetNearestColor
GetClassInfo	GetExpandedName	GetNearestPaletteIndex
GetClassLong	GetFileTitle	GetNextDlgGroupItem
GetClassName	GetFocus	GetNextDlgTabItem
GetClassWord	GetFontData	GetNextWindow
GetClientRect	GetFreeSpace	GetNumTasks
GetClipboardData	GetFreeSystemResources	GetObject
GetClipboardFormatName	GetInputState	GetOpenClipboardWindow
GetClipboardOwner	GetInstanceData	GetOpenFileName
GetClipboardViewer	GetKBCodePage	GetOutlineTextMetrics
GetClipBox	GetKerningPairs	GetPaletteEntries
GetClipCursor	GetKeyboardState	GetParent
GetCommErrror	GetKeyboardType	GetPixel

GetPolyFillMode	GetTextFace	GlobalFlags
GetPriorityClipboardFormat	GetTextMetrics	GlobalFree
GetPrivateProfileInt	GetTickCount	GlobalGetAtomName
GetPrivateProfileString	GetTimerResolution	GlobalHandle
GetProcAddress	GetTopWindow	GlobalLock
GetProfileInt	GetUpdateRect	GlobalLRUNewest
GetProfileString	GetUpdateRgn	GlobalLRUOldest
GetProp	GetVersion	GlobalNotify
GetQueueStatus	GetViewportExt	GlobalReAlloc
GetRasterizerCaps	GetViewportExtEx	GlobalSize
GetRgnBox	GetViewportOrg	GlobalUnfix
GetROP2	GetViewportOrgEx	GlobalUnlock
GetSaveFileName	GetWindow	GrayString
GetScrollPos	GetWindowDC	GrayStringProc
GetScrollRange	GetWindowExt	HideCaret
GetStockObject	GetWindowExtEx	HiLiteMenuItem
GetStretchBltMode	GetWindowLong	InflateRect
GetSubMenu	GetWindowOrg	InitAtomTable
GetSysColors	GetWindowOrgEx	InSendMessage
GetSysModalWindow	GetWindowPlacement	InsertMenu
GetSystemDirectory	GetWindowRect	IntersectClipRect
GetSystemMenu	GetWindowsDir	IntersectRect
GetSystemMetrics	GetWindowTask	InvalidateRect
GetSystemPaletteEntries	GetWindowText	InvalidateRgn
GetSystemPaletteUse	GetWindowTextLength	InvertRect
GetTabbedTextExtent	GetWindowWord	InvertRgn
GetTempDrive	GetWinFlags	IsBadCodePtr
GetTempFileName	GlobalAddAtom	IsBadHugeReadPtr
GetTextAlign	GlobalAlloc	IsBadHugeWritePtr
GetTextCharacterExtra	GlobalCompact	IsBadReadPtr
GetTextColor	GlobalDeleteAtom	IsBadStringPtr
GetTextExtent	GlobalFindAtom	IsBadWritePtr
GetTextExtentPoint	GlobalFix	IsCharAlpha

IsCharAlphaNumeric	LocalCompact	MulDiv
IsCharLower	LocalFirst	NotifyProc
IsCharUpper	LocalFlags	OemKeyScan
IsChild	LocalFree	OemToAnsi
IsClipboardFormatAvailable	LocalHandle	OemToAnsiBuff
IsDBCSLeadByte	LocalInit	OffsetClipRgn
IsDialogMessage	LocalLock	OffsetRect
IsDlgButtonChecked	LocalNext	OffsetRgn
IsGDIObject	LocalRealloc	OffsetViewportOrg
IsIconic	LocalShrink	OffsetViewportOrgEx
IsMenu	LocalSize	OffsetWindowOrg
IsRectEmpty	LocalUnlock	OffsetWindowOrgEx
IsTask	LockInput	OpenClipboard
IsWindow	LockResource	OpenComm
IsWindowEnabled	LockWindowUpdate	OpenFile
IsWindowVisible	LPtoDP	OpenIcon
IsZoomed	lstrcat	OutputDebugString
KillTimer	lstrcmp	PaintRgn
LibMain	lstrcmpi	PatBlt
LineDDA	lstrcpy	PeekMessage
LineDDAProc	lstrcpyn	Pie
LineTo	lstrlen	PlayMetaFile
LoadBitmap	MakeProcInstance	PlayMetaFileRecord
LoadCursor	MapDialogRect	Polygon
LoadIcon	MapVirtualKey	PolyLine
LoadLibrary	MapWindowPoints	PolyPolygon
LoadMenu	MessageBeep	PostAppMessage
LoadMenuIndirect	MessageBox	PostMessage
LoadModule	MessageProc	PostQuitMessage
LoadProc	ModifyMenu	PrintDlg
LoadResource	MoveTo	PtInRect
LoadString	MoveToEx	PtInRegion
LocalAlloc	MoveWindow	PtVisible

QueryAbort	ScaleWindowExtEx	SetDlgItemInt
QuerySendMessage	ScreenToClient	SetDlgItemText
ReadComm	ScrollDC	SetDoubleClickTime
RealizePalette	ScrollWindow	SetErrorMode
Rectangle	ScrollWindowEx	SetFocus
RectInRegion	SelectClipRgn	SetHandleCount
RectVisible	SelectObject	SetKeyboardState
RedrawWindow	SelectPalette	SetMapMode
RegCloseKey	SendDlgItemMessage	SetMapperFlags
RegCreateKey	SendMessage	SetMenu
RegDeleteKey	SetAbortProc	SetMessageQueue
RegEnumKey	SetActiveWindow	SetMetaFileBits
RegisterClass	SetBitmapBits	SetMetaFileBitsBetter
RegisterClipboardFormat	SetBitMapDimension	SetPaletteEntries
RegisterWindowMessage	SetBitMapDimensionEx	SetParent
RegOpenKey	SetBkColor	SetPixel
RegQueryValue	SetBkMode	SetPolyFillMode
RegSetValue	SetBoundsRect	SetProp
ReleaseCapture	SetBrushOrg	SetRect
ReleaseDC	SetCapture	SetRectEmpty
RemoveFontResource	SetCaretBlinkTime	SetRectRgn
RemoveMenu	SetCaretPos	SetResourceHandler
RemoveProp	SetClassLong	SetROP2
ReplaceText	SetClassWord	SetScrollPos
ReplyMessage	SetClipboardData	SetScrollRange
ResetDC	SetClipboardViewer	SetStretchBltMode
ResizePalette	SetCommBreak	SetSysColors
RestoreDC	SetCommEventMask	SetSysModalWindow
RoundRect	SetCommState	SetSystemPaletteUse
SaveDC	SetCursor	SetTextAlign
ScaleViewportExt	SetCursorPos	SetTextCharacterExtra
ScaleViewportExtEx	SetDIBits	SetTextColor
ScaleWindowExt	SetDIBitsToDevice	SetTextJustification

SetTimer	SpoolFile	UnionRect
SetViewportExt	StartDoc	UnrealizeObject
SetViewportExtEx	StartPage	UnregisterClass
SetViewportOrg	StretchBlt	UpdateColors
SetViewportOrgEx	StretchDIBits	UpdateWindow
SetWindowExt	SubtractRect	ValidateRec
SetWindowExtEx	SwapMouseButton	ValidateRgn
SetWindowLong	SysMsgProc	VkKeyScan
SetWindowOrg	SystemParametersInfo	WaitMessage
SetWindowOrgEx	TabbedTextOut	WEP
SetWindowPlacement	TextOut	WindowFromPoint
SetWindowPos	Throw	WindowProc
SetWindowsHook	TimerProc	WinExec
SetWindowsHookEx	ToAscii	WinHelp
SetWindowText	TrackPopupMenu	WinMain
SetWindowWord	TranslateAccelerator	WriteComm
ShowCaret	TranslateMDISysAccel	WritePrivateProfileString
ShowCursor	TranslateMessage	WriteProfileString
ShowOwnedPopups	TransmitCommChar	wsprintf
ShowScrollBar	UngetCommChar	wsvprintf
ShowWindow	UnhookWindowsHook	Yield
SizeofResource	UnhookWindowsHookEx	

Annex B

Unsupported Windows 3.1 Functions

B.1 Description

This annex lists unsupported Windows 3.1 functions by functional group.

B.1.1 Compression Functions

CopyLZFile	LZDone	LZOpenFile	LZSeek
LZClose	LZIni	LZRead	LZStart

B.1.2 Control Panel Functions

CPlApplet

B.1.3 DDE Functions

DdeAbandonTransaction	DdeConnectList	DdeFreeStringHandle	DdeQueryConvInfo
DdeAccessData	DdeCreateDataHandle	DdeGetData	DdeQueryNextServer
DdeAddData	DdeCreateStringHandle	DdeGetLastError	DdeQueryString
DdeCallback	DdeDisconnect	DdeInitialize	DdeReconnect
DdeClientTransaction	DdeDisconnectList	DdeKeepStringHandle	DdeSetUserHandle
DdeCmpStringHandles	DdeEnableCallback	DdeNameService	DdeUnaccessData
DdeConnect	DdeFreeDataHandle	DdePostAdvise	DdeUninitialize

B.1.4 Debugging Functions

DebugBreak	GetWinDebugInfo	SetWinDebugInfo
DebugProc	LogError	ValidateCodeSegments
GetSystemDebugState	LogParamError	ValidateFreeSpaces

B.1.5 Drag and Drop Functions

DragAcceptFiles	DragFinish	DragQueryFile	DragQueryPoint

B.1.6 Driver Functions

CloseDriver	DefDriverProc	DriverProc	GetDriverInfo
GetDriverModuleHandle	GetNextDriver	OpenDriver	SendDriverMessage

B.1.7 Edit Control Functions

WordBreakProc

B.1.8 File I/O Functions

_hread	_hwrite

B.1.9 File Manager Functions

UndeleteFile

B.1.10 Font Functions

CreateScalableFontResource	GetGlyphOutline

B.1.11 Hardware Functions

EnableHardwareInput

B.1.12 Hook Call-Back Functions

CBTProc	JournalPlaybackProc	KeyboardProc	ShellProc
HardwareProc	JournalRecordProc	MouseProc	

B.1.13 Memory Management Functions

GlobalDosAlloc	hmemcopy	SetSelectorLimit	UnlockSegment
GlobalDosFree	LimitEmsPages	SetSwapAreaSize	
GetSelectorBase	LockSegment	SwitchStackBack	
GetSelectorLimit	SetSelectorBase	SwitchStackTo	

B.1.14 Module Management Functions

GetCodeHandle

B.1.15 Message Functions

hardware_event

B.1.16 Networking Functions

WNetAddConnection	WNetCancelConnection	WNetGetConnection

B.1.17 OLE Functions

OleActivate
OleBlockServer
OleClone
OleClose
OleCopyFromLink
OleCopyToClipboard
OleCreate
OleCreateFromClip
OleCreateFromFile
OleCreateFromTemplate
OleCreateInvisible
OleCreateLinkFromClip
OleCreateLinkFromFile
OleDelete
OleDraw
OleEnumFormats
OleEnumObjects
OleEqual
OleExecute
OleGetData
OleGetLinkUpdateOptions
OleIsDcMeta
OleLoadFromStream
OleLockServer
OleObjectConvert
OleQueryBounds
OleQueryClientVersion
OleQueryCreateFromClip
OleQueryLinkFromClip
OleQueryName
OleQueryOpen
OleQueryOutOfDate
OleQueryProtocol
OleQueryReleaseError
OleQueryReleaseMethod
OleQueryReleaseStatus
OleQueryServerVersion
OleQuerySize
OleQueryType
OleReconnect
OleRegisterClientDoc
OleRegisterServer
OleRegisterServerDoc
OleRelease
OleRename
OleRenameClientDoc
OleRenameServerDoc
OleRequestData
OleRevertClientDoc
OleRevertServerDoc
OleRevokeClientDoc
OleRevokeObject
OleRevokeServer
OleRevokeServerDoc
OleSavedClientDoc
OleSavedServerDoc
OleSaveToStream
OleSetBounds
OleSetColorScheme
OleSetData
OleSetHostNames
OleSetLinkUpdateOptions
OleSetTargetDevice
OleUnblockServer
OleUnlockServer
OleUpdate
Open

B.1.18 Profiler Functions

ProfClear
ProfSampRate
ProfFinish
ProfSetup
ProfFlush
ProfStart
ProfInsChk
ProfStop

B.1.19 Program Manager Functions

FMExtensionProc

B.1.20 Process Management Functions

GetCurrentPDB

B.1.21 Resource Manager Functions

AccessResource

B.1.22 Segment Functions

AllocDStoCSAlias
AllocSelector
FreeSelector
GetCodeInfo
GlobalPageLock
GlobalPageUnlock
PrestoChangoSelector

B.1.23 Shell Functions

ShellExecute

B.1.24 Stress Functions

AllocDiskSpace	AllocMem	FreeAllMem	UnAllocDiskSpace
AllocFileHandles	AllocUserMem	FreeAllUserMem	UnAllocFileHandles
AllocGDIMem	FreeAllGDIMem	GetFreeFileHandles	

B.1.25 System Services Functions (General)

DOS3Call	NetBIOSCall
GetDOSEnvironment	WaitEvent

B.1.26 ToolHelp Functions

ClassFirst	InterruptUnRegister	SystemHeapInfo	TaskFirst
ClassNext	LocalFirst	StackTraceFirst	TaskNext
GlobalFirst	LocalNext	StackTraceCSIPFirst	TaskGetCSIP
GlobalNext	LocalInfo	StackTraceNext	TaskSetCSIP
GlobalEntryHandle	MemManInfo	ModuleFirst	TaskSwitch
GlobalEntryModule	MemoryRead	ModuleNext	TerminateApp
GlobalHandleToSel	MemoryWrite	ModuleFindHandle	TimerCount
GlobalInfo	NotifyRegister	ModuleFindName	
InterruptRegister	NotifyUnregister	TaskFindHandle	

B.1.27 Version Functions

GetFileResource	GetFileVersionInfoSize	VerFindFile	VerQueryValue
GetFileResourceSize	GetSystemDir	VerInstallFile	
GetFileVersionInfo	GetWindowsDir	VerLanguageName	

B.1.28 WINMEM32 DLL Functions

GetWinMem32Version	Global16PointerAlloc	Global16PointerFree	Global32Alloc
Global32CodeAlias	Global32CodeAliasFree	Global32Free	Global32Realloc

Annex C

Data Structures

C.1 Description

This annex describes data structures.

C.1.1 BITMAP

C.1.1.1 Synopsis

```
typedef struct tagBITMAP {
        int bmType;
        int bmWidth;
        int bmHeight;
        int bmWidthBytes;
        BYTE bmPlanes;
        BYTE bmBitsPixel;
        void *bmBits;
} BITMAP;
```

C.1.1.2 Description

The **BITMAP** structure contains information about a bitmap.

Element	**Description**
bmType	The type of bitmap. The value is zero for a logical bitmap.
bmWidth	The pixel width of bitmap. The value is greater than zero.
bmHeight	The raster line height of bitmap. The value is greater than zero.
bmWidthBytes	The number of bytes in each of the bitmap's raster lines. The value must be an even number. When *bmWidthBytes* is multiplied by 8, the resulting value must be the next multiple of 16 that is greater than or equal to the value of the *bmWidth* * *bmBitsPixel*.
bmPlanes	The number of color planes in the bitmap.
bmBitsPixel	The number of contiguous color bits on each color plane that are used to define a pixel.
bmBits	The pointer to an array of one-byte values representing the bitmap's bit values.

Only two types of bitmap formats, monochrome and color, are currently used.

A monochrome bitmap has one bit per pixel, uses one color plane, and each scan line has a multiple of 16 bits. A monochrome bitmap's pixel color is either black or white. If a bit in the **bmBits** array has a value of 1, the pixel that it represents is colored white. If a bit in the **bmBits** array has a value of 0, the pixel that it represents is colored black.

Use the *GetDeviceCaps()* function with the RASTERCAPS value to determine if a device supports bitmaps. If the device supports bitmaps, the RC_BITBLT bit is set in the *GetDeviceCaps()* function's return value. Use the *GetDIBits()* and *SetDIBits()* functions to transfer a bitmap from one device to another.

C.1.1.3 Cross-References

CreateBitmapIndirect(), GetDIBits(), SetDIBits()

C.2 BITMAPCOREHEADER

C.2.1 Synopsis

```
typedef struct tagBITMAPCOREHEADER {
```

DWORD bcSize;

short bcWidth;

short bcHeight;

WORD bcPlanes;

WORD bcBitCount;

} BITMAPCOREHEADER;

C.2.2 Description

The **BITMAPCOREHEADER** structure contains information about a device-independent bitmap's (DIB) dimensions and color format.

Element	Description
bcSize	The size of the **BITMAPCOREHEADER** structure in bytes.
bcWidth	The pixel width of the bitmap.
bcHeight	The pixel height of the bitmap.
bcPlanes	The number of color planes for the destination device. This value should always be one.
bcBitCount	The number of contiguous color bits on each color plane that are used to define each pixel. The value of the **bcBitCount** element also defines the maximum number of colors in the DIB. The value of the **bcBitCount** element should always be 1, 4, 8, or 24.

Value	Meaning
1	The monochrome bitmap containing two entries in the DIB's color table. Each pixel in the bitmap is represented by a single bit in the bitmap array. If the bit has a value of zero, the pixel has the color specified in the first entry of the DIB's color table. If the bit has a value of one, the pixel has the color specified in the second entry of the DIB's color table.
4	The 16 color bitmap. Each pixel in the bitmap is represented by a four-bit index value of the DIB's color table. For example, if the first byte in the bitmap is 0x3F, the byte represents two pixels. The first pixel has the color specified in the fourth entry of the color table entry. The second pixel has the color specified in the sixteenth entry of the color table entry.
8	The 256 color bitmap. Each pixel in the bitmap is represented by a byte index value of the DIB's color table. For example, if the first byte in the bitmap is 0x3F, the byte represents one pixel. The first pixel has the color specified in the sixty-fourth entry of the color table entry.
24	The 2^24 color bitmap. There is no color table for the bitmap. Every three bytes in the bitmap array specify the RGB color value for a pixel.

C.2.3 Cross-References

BITMAPCOREINFO, BITMAPINFOHEADER

C.3 BITMAPCOREINFO

C.3.1 Synopsis

typedef struct tagBITMAPCOREINFO {

BITMAPCOREHEADER bmciHeader;

RGBTRIPLE bmciColors[1];

} BITMAPCOREINFO;

C.3.2 Description

The **BITMAPCOREINFO** structure contains information about a device-independent bitmap's (DIB) dimensions, color format, and colors used in the bitmap.

Element	**Description**
bmciHeader	The **BITMAPCOREHEADER** structure containing the device-independent bitmap's (DIB) dimensions and color format.
bmciColors	The array containing either **RGBTRIPLE** structures that specify each color used in the bitmap or 16-bit unsigned integers that are indexes into the currently realized logical palette. The colors should be in the order of their importance. The **bmciColors** array should not contain palette indexes if the bitmap is to be transferred to another application or stored in a file. The **bmciColors** array should only contain palette indexes when the application that is using it has exclusive and complete control over it. The number of entries in the array depends on the value of the **BITMAPCOREHEADER** structure's **bcBitCount** element. If the value is set to 1, the DIB is monochrome and the bmciColors array should contain two entries. If the value is set to 4, the DIB uses a maximum of 16 colors and the bmciColors array should contain 16 entries. If the value is set to 8, the DIB uses a maximum of 256 colors and the **bmciColors** array should contain 256 entries. If the value is set to 24, the DIB uses a maximum of 2^24 colors and the **bmciColors** array should be assigned a value of NULL.

The **BITMAPCOREINFO** structure is followed immediately in memory by an array of bytes that specify the bitmap's pixels.

C3.3 Cross-References

BITMAPINFO, BITMAPCOREHEADER, RGBTRIPLE

C.4 BITMAPINFO

C.4.1 Synopsis

typedef struct tagBITMAPINFO {

BITMAPINFOHEADER bmiHeader;

RGBQUAD bmiColors[1];

} BITMAPINFO;

C.4.2 Description

The **BITMAPINFO** structure contains all information about a device-independent bitmap's (DIB) dimensions and colors.

Element	**Description**
bmiHeader	The **BITMAPINFOHEADER** structure containing the device-independent bitmap's (DIB) dimensions and color format.
bmiColors	The array containing either **RGBQUAD** structures that specify each color used in the DIB or 16-bit unsigned integers that are indexes into the currently realized logical palette. The colors should be in the order of their importance. The **bmiColors** array should not contain palette indexes if the DIB is to be transferred to another application or stored in a file. The **bmiColors** array should only contain palette indexes when the application that is using it has exclusive and complete total control over it. If the value of the given **BITMAPINFOHEADER** structure's **biClrUsed** element is set to zero, the DIB uses the maximum number of colors corresponding to the value of the structure's **biBitCount** element. In this case, if the value of the **biBitCount** element is set to 1, the DIB is monochrome and the **bmiColors** array should contain two entries. If the value is set to 4, the DIB uses a maximum of 16 colors and the **bmiColors** array should contain 16

entries. If the value is set to 8, the DIB uses a maximum of 256 colors and the **bmiColors** array should contain 256 entries. If the value is set to 24, the DIB uses a maximum of 2^24 colors and the **bmiColors** array should be assigned a value of NULL.

C.4.3 Cross-References

BITMAPINFOHEADER, RGBQUAD

C.5 BITMAPINFOHEADER

C.5.1 Synopsis

```
typedef struct tagBITMAPINFOHEADER {
        DWORD biSize;
        LONG biWidth;
        LONG biHeight;
        WORD biPlanes;
        WORD biBitCount;
        DWORD biCompression;
        DWORD biSizeImage;
        LONG biXPelsPerMeter;
        LONG biYPelsPerMeter;
        DWORD biClrUsed;
        DWORD biClrImportant;
} BITMAPINFOHEADER;
```

C.5.2 Description

The **BITMAPINFO** structure contains all information about a device-independent bitmap's (DIB) dimensions and colors.

Element	Description
biSize	The size of the **BITMAPINFOHEADER** structure in bytes.
biWidth	The pixel width of bitmap.
biHeight	The pixel height of bitmap.
biPlanes	The number of color planes for the destination device. This value should always be one.
biBitCount	The number of contiguous color bits on each color plane that are used to define each pixel. The value of the **biBitCount** element also defines the maximum number of colors in the DIB. The value the **biBitCount** element should always be 1, 4, 8, or 24.

Value	Description
1	The monochrome bitmap containing two entries in the DIB's color table. Each pixel in the bitmap is represented by a single bit in the bitmap array. If the bit has a value of zero, the pixel has the color specified in the first entry of the DIB's color table. If the bit has a value of one, the pixel has the color specified in the second entry of the DIB's color table.
4	The 16 color bitmap. Each pixel in the bitmap is represented by a four-bit index value into the DIB's color table. For example, if the first byte in the bitmap is 0x3F, the byte represents two pixels. The first pixel has the color specified in the fourth entry of the color table entry. The second pixel has the color specified in the sixteenth entry of the color table entry.

8	The 256 color bitmap. Each pixel in the bitmap is represented by a byte index value into the DIB's color table. For example, if the first byte in the bitmap is 0x3F, the byte represents one pixel. The first pixel has the color specified in the sixty-fourth entry of the color table entry.
24	The 2^24 color bitmap. There is no color table for the bitmap. Every three bytes in the bitmap array specifies the RGB color value for a pixel.

biCompression The type of compression used to compress the bitmap image. It can be one of the following constant values:

Value	**Description**
BI_RGB	Bitmap is not compressed.
BI_RLE8	Bitmap is compressed using the run-length encoded format for bitmaps with 8 bits per pixel. The algorithm uses a 2-byte format consisting of a count byte followed by a byte containing a color index.
BI_RLE4	Bitmap is compressed using the run-length encoded format for bitmaps with 4 bits per pixel. The algorithm uses a 2-byte format consisting of a count byte followed by two word-length color indexes.

biSizeImage The size in bytes of the decompressed bitmap image. The value can be zero if the image is not compressed.

BiXPelsPerMeter
The horizontal resolution of the DIB's destination device (in pixels per meter). This value can be used to determine if a given bitmap best matches a given destination device.

BiYPelsPerMeter
The vertical resolution of the DIB's destination device (in pixels per meter). This value can be used to determine if a given bitmap best matches a given destination device.

biClrUsed The number of entries in the DIB's color table.

If the value of the **biClrUsed** element is zero, the DIB uses the maximum number of colors corresponding to the value of the structure's **biBitCount** element.

If the value of the **biClrUsed** element is not zero and the **biBitCount** element's value is less than 24, the value is the number of colors that the graphics engine or device driver will access.

If the value of the **biClrUsed** element's value is not zero and the **biBitCount** element's value is 24, **biClrUsed** element's value is the size of the reference color table used to optimize performance of color palettes.

If the DIB is a packed DIB (bitmap bit array follows the **BITMAPINFO** header and which is referenced by a single pointer), the **biClrUsed** element's value must be zero or the actual size of the color table.

biClrImportant The number of colors that are considered important when displaying the bitmap. If the value is zero, it is assumed that all of the colors are important when displaying the bitmap.

C.5.3 Cross-References

BITMAPINFO

C.6 CHOOSECOLOR

C.6.1 Synopsis

```
typedef struct tagCHOOSECOLOR {
    DWORD lStructSize;
    HWND hwndOwner;
    HWND hInstance;
    COLORREF rgbResult;
    COLORREF *lpCustColors;
    DWORD Flags;
    LPARAM lCustData;
    UINT (CALLBACK *lpfnHook)(HWND, UINT, WPARAM, LPARAM);
    LPCSTR lpTemplateName;
} CHOOSECOLOR;
```

C.6.2 Description

The **CHOOSECOLOR** structure contains information that is used by the system to initialize the Color common dialog box and to return the user's Color common dialog box selections.

Element	Description
lStructSize	The size of the **CHOOSECOLOR** structure in bytes. A value must be assigned to this element before the structure is passed to the *ChooseColor()* function.
hwndOwner	The handle of the window that owns the Color common dialog box. A value must be assigned to this element before the structure is passed to the *ChooseColor()* function. If there is no owner, the element's value should be NULL. If the CC_SHOWHELP flag is set in the **Flags** element, a valid window handle must be assigned to the **hwndOwner** element. If the user selects the dialog box's Help button, the window is sent a notification message. The message's ID is registered at runtime and can be retrieved by calling the *RegisterWindowMessage()* function with the constant HELPMSGSTRING.
hInstance	Should be assigned the handle of the data block containing the dialog box template given in the **lpTemplateName** element. The value of the hInstance element is used only when the CC_ENABLETEMPLATE or CC_ENABLETEMPLATEHANDLE constants are used in the **Flags** element. When the CC_ENABLETEMPLATE constant is used, **hInstance** is an instance handle; when the CC_ENABLETEMPLATEHANDLE constant is used, hInstance is a handle to a dialog resource. If either of these two constants are used, a value must be assigned to the **hInstance** element before the structure is passed to the *ChooseColor()* function.
rgbResult	When the **CHOOSECOLOR** structure is passed to the *ChooseColor()* function, the **rgbResult** element can contain the color that should be initially selected when the dialog box is initialized. After the user closes the Color common dialog box with the OK button, the **rgbResult** element contains the color that the user selected. If the CC_RGBINIT constant is set in the **Flags** element, a value must be assigned to the **hInstance** element before the structure is passed to the *ChooseColor()* function. If the color value is not available, the system selects the nearest solid color that is available. If the value of the **hInstance** element is NULL, the initially-selected color is black.
lpCustColors	This element is the pointer to an array of 16 doubleword values that specify the intensity of a red, green, and blue (RGB) component in the custom color box. A value must be assigned to this element before the structure is passed to the *ChooseColor()* function. If an RGB color value is changed in the dialog box, the corresponding entry in the array is updated with the modified color value.

Flags	These flags determine how the color common dialog box is initialized. A value must be assigned to this element before the structure is passed to the *ChooseColor()* function. The value of this element can be one or more of the following constant values OR'ed together:

CC_ENABLEHOOK	This value uses the hook function given in the structure's **lpfnHook** element.
CC_ENABLETEMPLATE	This value uses the dialog box template given in the **hInstance** and **lpTemplateName** elements.
CC_ENABLETEMPLATEHANDLE	The **hInstance** element is a data block that has a pre-loaded dialog box template; the **lpTemplateName** element is ignored.
CC_FULLOPEN	This value displays the entire Color common dialog box including the part that allows the creation of custom colors. If this constant is not used, the custom colors section of the dialog box is not visible initially, and the user will have to press the "Define Custom Color" button to see the custom colors section of the dialog box
CC_PREVENTFULLOPEN	This value disables the "Define Custom Colors" button.
CC_RGBINIT	This value uses the default color given in the **rgbResult** element.
CC_SHOWHELP	This value displays the Help button in the dialog box.

lCustData	This element is the application-defined data that the system passes to the hook function specified in the structure's **lpfnHook** element when the Color dialog box is initialized.
lpfnHook	This element is the pointer to a hook function that processes messages for the Color dialog box. The hook function is used only when the CC_ENABLEHOOK constant is specified in the structure's **Flags** element. The hook function is sent all of the messages that the Color dialog box receives. When the dialog box is created, the hook function is sent a WM_INITDIALOG message whose *lParam* contains a pointer to the **CHOOSECOLOR** structure. This is the only time that the hook function can access the application-defined data specified in the **lCustData** element and to the rest of the values stored in the **CHOOSECOLOR** structure. The hook function must return TRUE when it processes a message it is sent, or zero when it does not process a message it is sent.
lpTemplateName	This element is a null-terminated string containing the name of the resource file that has an application-defined dialog box template that is to be substituted for the standard Color common dialog box's template. This element is used only when the CC_ENABLETEMPLATE constant is specified in the structure's **Flags** element. The MAKEINTRESOURCE macro can be used if the dialog box resource is numbered.

C.6.3 Cross-References

ChooseColor(), MAKEINTRESOURCE, RGB

C.7 CHOOSEFONT

C.7.1 Synopsis

typedef struct tagCHOOSEFONT {

DWORD lStructSize;

HWND hwndOwner;

HDC hdc;

LOGFONT *lpLogFont;

```
    int iPointSize;
    DWORD Flags;
    COLORREF rgbColors;
    LPARAM lCustData;
    UINT (CALLBACK *lpfnHook)(HWND, UINT, WPARAM, LPARAM);
    LPCSTR lpTemplateName;
    HINSTANCE hInstance;
    LPSTR lpszStyle;
    UINT nFontType;
    int nSizeMin;
    int nSizeMax;
} CHOOSEFONT;
```

C.7.2 Description

The **CHOOSEFONT** structure contains information that is used by the system to initialize the Font common dialog box and to return the user's Font common dialog box selections.

Element	Description
lStructSize	This element is the size of the **CHOOSEFONT** structure in bytes. A value must be assigned to this element before the structure is passed to the *ChooseFont()* function.
hwndOwner	This element is the handle of the window that owns the Font common dialog box. A value must be assigned to this element before the structure is passed to the *ChooseFont()* function. If there is no owner, the element's value should be NULL. If the CF_SHOWHELP flag is set in the **Flags** element, a valid window handle must be assigned to the **hwndOwner** element. If the user selects the dialog box's Help button, the window sends a notification message. The message's ID is registered at runtime and can be retrieved by calling the *RegisterWindowMessage()* function with the constant HELPMSGSTRING.
hdc	This element is the device-context or information context of the printer for which fonts are to be listed in the Font common dialog box. A value must be assigned to this element before the structure is passed to the *ChooseFont()* function. The value of this element is used only when the constant CF_PRINTERFONTS is set in the structure's **Flags** element.
lpLogFont	This element is the pointer to a **LOGFONT** structure the describes the font that should be initially displayed when the Font common dialog box is shown. If the font is not available, its closest match is shown instead. A value must be assigned to this element before the structure is passed to the *ChooseFont()* function. The value of this element is used only when the constant CF_INITTOLOGFONTSTRUCT is set in the structure's **Flags** element. After the user closes the Font common dialog box with the OK button, the **lpLogFont** element contains information about the last font that the user selected.
iPointSize	This element is the size of the last selected font, in tenths of a point, is stored in this element after the user closes the Font common dialog box with the OK button.
Flags	These flags determine how the Font common dialog box is initialized. After the user closes the Font common dialog box with the OK button, the **Flags** element will contain information about the user's font selection. A value must be assigned to this element before the structure is passed to the *ChooseFont()* function. The value may the one or more of the following constant values OR'ed together:

CF_APPLY	This value enables the "Apply" button. in the Font common dialog box.

CF_ANSIONLY	This value only allows selection of fonts that use the Windows character set. For example, the user cannot select a font that contains only symbols.
CF_BOTH	This value shows the available screen and printer fonts using the context given in the structure's **hdc** element.
CF_TTONLY	This value only shows TrueType fonts.
CF_EFFECTS	This value allows strikeout, underline, and color effects. If this constant is used, the **LOGFONT** structure's **lfStrikeOut** and **lfUnderline** elements and the **CHOOSEFONT** structure's **rgbColors** element can be set before calling the *ChooseFont()* function. If this constant is not used, the *ChooseFont()* function can set the values of these elements after the user closes the Font common dialog box with the OK button.
CF_ENABLEHOOK	This value uses the hook function given in the structure's **lpfnHook** element.
CF_ENABLETEMPLATE	This value uses the dialog box template given in the **hInstance** and **lpTemplateName** elements.
CF_ENABLETEMPLATEHANDLE	The **hInstance** element is a data block that has a pre-loaded dialog box template. The **lpTemplateName** element should be ignored.
CF_FIXEDPITCHONLY	This value selects only monospace fonts.
CF_FORCEFONTEXIST	This value reports an error if the user tries to select a font or font style that does not exist.
CF_INITTOLOGFONTSTRUCT	This value initializes the Font common dialog box by using the information in the **LOGFONT** structure specified in the structure's **lpLogFont** element.
CF_LIMITSIZE	This value selects only those font sizes that are within the range given in the structure's **nSizeMin** and **nSizeMax** elements.
CF_NOFACESEL	This value means that there is no selection in the "face name" combo box. This flag can be used to support multiple font selections. After the user closes the Font common dialog box with the OK button, the CF_NOFACESEL constant is set in the **Flags** element if there was no face name selection.
CF_NOOEMFONTS	This value means that there are no vector-font selections. It is the same as CF_NOVECTORFONTS.
CF_NOSIMULATIONS	This value does not allow graphics-device-interface (GDI) font simulations.
CF_NOSIZESEL	This value means that there is no selection in the "Size" combo box. This flag can be used to support multiple size selections. After the user closes the Font common dialog box with the OK button, the CF_NOSIZESEL constant is set in the **Flags** element, if there was no size selection.
CF_NOSTYLESEL	This value means that there is no selection in the "Font Style" combo box. This flag can be used to support multiple style selections. After the user closes the Font common dialog box with the OK button, the CF_NOSTYLESEL constant is set in the **Flags** element if there was no style selection.

CF_NOVECTORFONTS	This value means that there are no vector-font selections. It is the same as CF_NOOEMFONTS.
CF_PRINTERFONTS	This value shows only the fonts supported by the printer associated with the context given in the structure's **hdc** element.
CF_SCALABLEONLY	This value selects only scalable fonts (for example, vector fonts, some printer fonts, TrueType fonts, and fonts that are scaled by other algorithms or technologies).
CF_SCREENFONTS	This value shows only the screen fonts supported by the system.
CF_SHOWHELP	This value displays the Help button in the dialog box.
CF_USESTYLE	When the Font common dialog box is created, this value uses the font style specified by the **lpszStyle** element.
CF_WYSIWYG	This value selects only fonts that are available on both the printer and the screen. The CF_BOTH and CF_SCALABLEONLY constants should be used as well.

rgbColors These elements are the red, green, and blue (RGB) values to use when setting the initial text color. The value of this element is used when the CF_EFFECTS constant is set in the **Flags** element. After the user closes the Font common dialog box with the OK button, the RGB values for the selected font's color are copied to the **rgbColors** element.

lCustData This element is the application-defined data that the system passes to the hook function specified in the structure's **lpfnHook** element when the Font dialog box is initialized.

lpfnHook This element is the pointer to a hook function that processes messages for the Color dialog box. The hook function is used only when the CF_ENABLEHOOK constant is specified in the structure's **Flags** element.

The hook function is sent all of the messages that the Color dialog box receives. When the dialog box is created, the hook function is sent a WM_INITDIALOG message whose *lParam* contains a pointer to the **CHOOSECOLOR** structure. This is the only time that the hook function has access to the application-defined data specified in the **lCustData** element and to the rest of the values stored in the **CHOOSECOLOR** structure. The hook function must return TRUE when it processes a received message, or FALSE when it does not process a received message.

lpTemplateName
This element is the null-terminated string containing the name of the resource file that has an application-defined dialog box template that is to be substituted for the standard Font common dialog box's template. This element is used only when the CF_ENABLETEMPLATE constant is specified in the structure's **Flags** element. The MAKEINTRESOURCE macro can be used if the dialog box resource is numbered.

hInstance This is the value that should be assigned the handle of the data block containing the dialog box template given in the **lpTemplateName** element.

The value of the **hInstance** element is used only when the CF_ENABLETEMPLATE or CF_ENABLETEMPLATEHANDLE constants are used in the **Flags** element. When the CC_ENABLETEMPLATE constant is used, **hInstance** is an instance handle; when the CC_ENABLETEMPLATEHANDLE constant is used, **hInstance** is a handle to a dialog resource. If either of these two constants are used, a value must be assigned to the **hInstance** element before the structure is passed to the *ChooseFont()* function.

lpszStyle This element is the buffer containing a null-terminated string that is the description of the initial font style. This element is used only when the CF_USESTYLE constant is specified in the structure's **Flags** element. After the user closes the Font common dialog box with the OK button, the description of the selected style is copied to the buffer. The buffer should be at least LF_FACESIZE bytes in size.

nFontType This element is the type of the selected font. This value may the one or more of the following constant values OR'ed together:

BOLD_FONTTYPE	This value means that the font is bold. This constant only impacts TrueType fonts and corresponds to the **NEWTEXTMETRIC** structure's **ntmFlags** element.
ITALIC_FONTTYPE	This value means that the font is italic. This constant only impacts TrueType fonts and corresponds to the **NEWTEXTMETRIC** structure's **ntmFlags** element.
PRINTER_FONTTYPE	This value means that the font is a printer font.
REGULAR_FONTTYPE	This value means that the font is not bold or italic. This constant only impacts TrueType fonts and corresponds to the **NEWTEXTMETRIC** structure's **ntmFlags** element.
SCREEN_FONTTYPE	This value means that the font is a screen font.
SIMULATED_FONTTYPE	This value means that the font is simulated by graphics device inteface. This is not used if the CF_NOSIMULATIONS constant is used in the **CHOOSEFONT** structure's **Flags** element.

nSizeMin	This element is the minimum point size that can be selected by a user. The value of this element is used only when the constant CF_LIMITSIZE is assigned to the structure's **Flags** element. A value must be assigned to this element before the structure is passed to the *ChooseFont()* function.
nSizeMax	This element is the maximum point size that can be selected by a user. The value of this element is used only when the constant CF_LIMITSIZE is assigned to the structure's **Flags** element. A value must be assigned to this element before the structure is passed to the *ChooseFont()* function.

C.7.3 Cross-References

ChooseFont(), **LOGFONT,** MAKEINTRESOURCE, **NEWTEXTMETRIC**

C.8 CLASSENTRY

C.8.1 Synopsis

typedef struct tagCLASSENTRY {

DWORD dwSize;

HMODULE hInst;

char szClassName[MAX_CLASSNAME + 1];

WORD wNext;

} CLASSENTRY;

C.8.2 Description

The **CLASSENTRY** structure contains the handle to the owner and name of a class.

Element	**Description**
dwSize	This element is the size of the **CLASSENTRY** structure in bytes.
hInst	This element is the handle of the module that owns the class. The handle can be used in calls to the *GetClassInfo()* function. The hInst element is really a handle to a module, since Windows classes are owned by modules.
szClassName	This element is the null-terminated string containing the name of the class. The name can be used in calls to the *GetClassInfo()* function.
wNext	This element is the next class in the class list. It is reserved for use by the system.

C.8.3 Cross-References

ClassFirst(), ClassNext(), GetClassInfo()

C.9 CLIENTCREATESTRUCT

C.9.1 Synopsis

```
typedef struct tagCLIENTCREATESTRUCT {
    HANDLE hWindowMenu;
    UINT idFirstChild;
} CLIENTCREATESTRUCT;
```

C.9.2 Description

The **CLIENTCREATESTRUCT** structure contains information about a multiple document interface (MDI) client window's menu and first MDI child window.

Element	Description
hWindowMenu	This element is the handle of the Window menu. This handle can be retrieved from the menu of the MDI frame window by calling the *GetSubMenu()* function.
idFirstChild	This element is the initial identifier for the first MDI child window that is created. As each new MDI child window is created, the system increments the identifier. When a MDI child window is destroyed, and another is created, the system reuses the identifier. A new MDI child window's identifier should not conflict with any other WM_COMMAND identifiers since the identifiers are used in WM_COMMAND messages to the application's MDI frame window.

C.9.3 Cross-References

CreateWindow(),GetSubMenu()

C.10 COMPAREITEMSTRUCT

C.10.1 Synopsis

```
typedef struct tagCOMPAREITEMSTRUCT {
    UINT CtlType;
    UINT CtlID;
    HWND hwndItem;
    UINT itemID1;
    DWORD itemData1;
    UINT itemID2;
    DWORD itemData2;
} COMPAREITEMSTRUCT;
```

C.10.2 Description

The **COMPAREITEMSTRUCT** structure contains the identifiers and application-defined data for two items in a sorted, owner-drawn combo box or list box control.

Element	Description	
CtlType	This element is the type of control. The element contains one of the following values:	
	ODT_LISTBOX	This value is the owner-drawn list box.
	ODT_COMBOBOX	This value is the owner-drawn combo box.
CtlID	This element is the control's identifier.	
hwndItem	This element is the control's window handle.	
itemID1	Index value of the first item in the control that is being compared.	
itemData1	Application-defined data associated with the first item in the control that is being compared.	

itemID2	Index value of the second item in the control that is being compared.
itemData2	Application-defined data associated with the second item in the control that is being compared.

C.10.3 Cross-References

WM_COMPAREITEM

C.11 CREATESTRUCT

C.11.1 Synopsis

```
typedef struct tagCREATESTRUCT {
    void *lpCreateParams;
    HINSTANCE hInstance;
    HMENU hMenu;
    HWND hwndParent;
    int cy;
    int cx;
    int y;
    int x;
    LONG style;
    LPCSTR lpszName;
    LPCSTR lpszClass;
    DWORD dwExStyle;
} CREATESTRUCT;
```

C.11.2 Description

The **CREATESTRUCT** structure contains initialization information that is passed to a new window's window procedure.

Element	**Description**
lpCreateParams	This element is the pointer to data to use when creating the new window.
hInstance	This element is the module-instance handle of the module that owns the new window.
hMenu	This element is the new window's menu.
hwndParent	This element is the handle of the window that owns the new window. The element's value is NULL if the new window is a top-level window.
cy	This element is the new window's height.
cx	This element is the new window's width.
y	This element is the Y-coordinate of the new window's upper-left corner. If the new window is a child window, the coordinate is relative to its parent window. If the new window is not a child window, the coordinate is relative to the screen's origin.
x	This element is the X-coordinate of the new window's upper-left corner. If the new window is a child window, the coordinate is relative to its parent window. If the new window is not a child window, the coordinate is relative to the screen's origin.
style	This element is the new window's style.
lpszName	This element is the pointer to a null-terminated string that contains the new window's name.
lpszClass	This element is the pointer to a null-terminated string that contains the new window's class name.
dwExStyle	This element is the new window's extended style.

C.11.3 Cross-References

CreateWindow()

C.12 DELETEITEMSTRUCT

C.12.1 Synopsis

```
typedef struct tagDELETEITEMSTRUCT {
    UINT CtlType;
    UINT CtlID;
    UINT itemID;
    HWND hwndItem;
    DWORD itemData;
} DELETEITEMSTRUCT;
```

C.12.2 Description

The **DELETEITEMSTRUCT** structure contains information associated with an item that is deleted from an owner-drawn list-box or combo-box control.

Element	Description
CtlType	This element is the type of control from which the item was deleted. The element contains one of the following values: ODT_LISTBOX — This value is the owner-drawn list box. ODT_COMBOBOX — This value is the owner-drawn combo box.
CtlID	This element is the control's identifier.
itemID	This element is the index value of the item in the control that was deleted.
hwndItem	This element is the control's window handle.
itemData	This element is the application-defined data associated with the item that was deleted.

C.12.3 Cross-References

WM_DELETEITEM

C.13 DRAWITEMSTRUCT

C.13.1 Synopsis

```
typedef struct tagDRAWITEMSTRUCT {
    UINT CtlType;
    UINT CtlID;
    UINT itemID;
    UINT itemAction;
    UINT itemState;
    HWND hwndItem;
    HDC hdc;
    RECT rcItem;
    DWORD itemData;
} DRAWITEMSTRUCT;
```

C.13.2 Description

The **DRAWITEMSTRUCT** structure contains information that the control's owner needs to determine how to paint an owner-drawn control.

Element	**Description**	
CtlType	This element is the type of control from which the item was deleted. The element contains one of the following values:	
	ODT_BUTTON	This value is the owner-drawn button.
	ODT_COMBOBOX	This value is the owner-drawn combo box.
	ODT_LISTBOX	This value is the owner-drawn list box.
	ODT_MENU	This value is the owner-drawn menu.
CtlID	This element is the control's identifier. It is not used for menu controls.	
itemID	This element is the index value of the item in the combo box or list box, or the menu-item identifier for a menu control. If the combo box or list box is empty, the value of **itemID** is negative.	
itemAction	This element is the type of drawing to perform. The element will contain one of the following values:	
	ODA_DRAWENTIRE	This value means the entire control needs to be drawn.
	ODA_FOCUS	This value means the control has lost or obtained focus.
	ODA_SELECT	This value means the selection status has changed.
itemState	This element is the state of the control after the current drawing action is performed. The element contains one of the following constant values:	
	ODS_CHECKED	This value means the menu item is to be checked. It is only used for menu controls.
	ODS_DISABLED	This value means the item is to be drawn as disabled.
	ODS_FOCUS	This value means the item has input focus.
	ODS_GRAYED	This value means the item is to be grayed. It is only used for menu controls.
	ODS_SELECTED	This value means the item's status is selected.
hwndItem	This element is the window handle of the button, combo box or list box, or the handle of the menu.	
hdc	This element is the device context to use when performing drawing operations on the control.	
rcItem	This element is the **RECT** structure containing the boundaries of the control to be drawn. Anything that the owner draws in the device context for combo boxes, list boxes, and buttons is clipped by the system. Clipping is not performed for menu items. When menu items are drawn, the system must ensure that the owner does not draw outside the boundaries of the **rcItem**.	
i**temData**	This element is the last value assigned to the combo box or list box through an LB_SETITEMDATA or CB_SETITEMDATA message. If the LBS_HASSTRINGS or CBS_HASSTRINGS style is set in the combo box or list box, the value for itemData is zero initially. If the LBS_HASSTRINGS or CBS_HASSTRINGS style is not set in the combo box or list box, the initial value of **itemData** is the value passed to the control in the lParam parameter of the CB_ADDSTRING, CB_INSERTSTRING, LB_ADDSTRING, or LB_INSERTSTRING message.	

C.13.3 Cross-References

CB_ADDSTRING, CB_INSERTSTRING, CB_SETITEMDATA, CBS_HASSTRINGS, **RECT**, LB_ADDSTRING, LB_INSERTSTRING, LB_SETITEMDATA, LBS_HASSTRINGS, WM_DRAWITEM

C.14 FINDREPLACE

C.14.1 Synopsis

```
typedef struct tagFINDREPLACE {
    DWORD lStructSize;
    HWND hwndOwner;
    HINSTANCE hInstance;
    DWORD Flags;
    LPSTR lpstrFindWhat;
    LPSTR lpstrReplaceWith;
    UINT wFindWhatLen;
    UINT wReplaceWithLen;
    LPARAM lCustData;
    UINT (CALLBACK *lpfnHook)(HWND, UINT, WPARAM, LPARAM);
    LPCSTR lpTemplateName;
} FINDREPLACE;
```

C.14.2 Description

The **FINDREPLACE** structure contains information that is used by the system to initialize the Find and Replace common dialog boxes and to return the user's dialog box selections.

Element	Description
lStructSize	This element is the size of the **FINDREPLACE** structure in bytes. A value must be assigned to this element before the structure is passed to the *FindText()* or *ReplaceText()* functions.
hwndOwner	This element is the handle of the window that owns the Color common dialog box. A value must be assigned to this element before the structure is passed to the *FindText()* or *ReplaceText()* functions. If there is no owner, the element's value should be NULL. If the FR_SHOWHELP flag is set in the **Flags** element, a valid window handle must be assigned to the **hwndOwner** element. If the user selects the dialog box's Help button, the window is sent a notification message. The message's ID is registered at runtime and can be retrieved by calling the *RegisterWindowMessage()* function with the constant HELPMSGSTRING.
hInstance	This element should be assigned the handle of the data block containing the dialog box template given in the **lpTemplateName** element. The value of the **hInstance** element is used only when the FR_ENABLETEMPLATE or FR_ENABLETEMPLATEHANDLE constants are used in the **Flags** element. When the CC_ENABLETEMPLATE constant is used, **hInstance** is an instance handle; when the CC_ENABLETEMPLATEHANDLE constant is used, **hInstance** is a handle to a dialog resource. If either of these two constants are used, a value must be assigned to the **hInstance** element before the structure is passed to the *FindText()* or *ReplaceText()* functions.
Flags	These flags determine how the common dialog box is initialized. A value must be assigned to this element before the structure is passed to the *FindText()* or *ReplaceText()* functions. The value of this element may the one or more of the following constant values OR'ed together:

FR_DIALOGTERM	This value means the dialog box is closing and the window handle returned by the *FindText()* or *ReplaceText()* functions is no longer valid. This constant is set by the system.
FR_DOWN	This value initially selects the search "down" button and searches down through the document. If this value is not used, the search direction is up and the "up" button is selected. After the user

	closes the dialog box with the OK button, the FR_DOWN constant can be used to determine the last search direction.
FR_ENABLEHOOK	This value uses the hook function given in the structure's **lpfnHook** element.
FR_ENABLETEMPLATE	This value uses the dialog box template given in the **hInstance** and **lpTemplateName** elements.
FR_ENABLETEMPLATEHANDLE	The **hInstance** element is a data block that has a pre-loaded dialog box template; the **lpTemplateName** element should be ignored.
FR_FINDNEXT	This value searches for the next occurrence of the string given in the structure's **lpstrFindWhat** element. This constant is set by the system.
FR_HIDEMATCHCASE	This value initially hides and disables the dialog box's "Match Case" check box.
FR_HIDEWHOLEWORD	This value initially hides and disables the dialog box's "Match Only Whole Word" check box.
FR_HIDEUPDOWN	This value initially hides the dialog box's "Up" and "Down" radio buttons.
FR_MATCHCASE	This value initially a search is to be case sensitive. This constant may be changed due to user input.
FR_NOMATCHCASE	This value initially disables the dialog box's "Match Case" check box.
FR_NOUPDOWN	This value initially disables the dialog box's "Up" and "Down" buttons.
FR_NOWHOLEWORD	This value initially disables the dialog box's "Match Whole Word Only" check box.
FR_REPLACE	This value replaces the current occurrence of the string given in the structure's **lpstrFindWhat** element with the string given in the structure's **lpstrReplaceWith** element. This flag is set by the system.
FR_REPLACEALL	This value replaces all occurrences of the string given in the structure's **lpstrFindWhat** element with the string given in the structure's **lpstrReplaceWith** element. This flag is set by the system.
FR_SHOWHELP	This value displays the Help button in the dialog box.
FR_WHOLEWORD	This value initially checks the dialog box's "Match Whole Word Only" check box. Only whole words that match the search string are considered during a search. This constant may be changed due to user input.

lpstrFindWhat	This element is a pointer to a buffer containing a null-terminate string for which to search. If **lpstrFindWhat** contains a valid value when the dialog box is created, the string is placed in the "Find What" edit control. If the FR_FINDNEXT constant is specified in the structure's **Flags** element when the dialog box is created, a search is performed for the string. The size of the buffer should be at least eighty bytes. The value of the **lpstrFindWhat** element may be changed due to user input.

lpstrReplaceWith	This element is a pointer to a buffer containing a null-terminate string that will replace search strings. The *FindText()* function does not use this element. If the **lpstrReplaceWith** element contains a valid value when the Replace common dialog box is created, the string is placed in "Replace With" edit control. The value of the **lpstrReplaceWith** element may be changed due to user input.
wFindWhatLen	This element is the size, in bytes, of the buffer pointed to by the structure's **lpstrFindWhat** element.
wReplaceWithLen	This element is the size, in bytes, of the buffer pointed to by the structure's **lpstrReplaceWith** element.
lCustData	This element is the application-defined data that the system passes to the hook function specified in the structure's **lpfnHook** element when the dialog box is initialized.
lpfnHook	This element is the pointer to a hook function that processes messages for the Color dialog box. The hook function is used only when the FR_ENABLEHOOK constant is specified in the structure's **Flags** element. The hook function is sent all of the messages that the dialog box receives. When the dialog box is created, the hook function is sent a WM_INITDIALOG message whose *lParam* contains a pointer to the **FINDREPLACE** structure. This is the only time that the hook function will have access to the application-defined data specified in the **lCustData** element and to the rest of the values stored in the **FINDREPLACE** structure. The hook function must return TRUE when it processes a message that is sent to it, or FALSE when it does not process a message that is sent to it.
lpTemplateName	This element is the null-terminated string containing the name of the resource file that has an application-defined dialog box template that is to be substituted for the standard common dialog box's template. This element is used only when the FR_ENABLETEMPLATE constant is specified in the structure's **Flags** element. The MAKEINTRESOURCE macro can be used if the dialog box resource is numbered.

C.14.3 Cross-References

FindText(), ReplaceText(), MAKEINTRESOURCE

C.15 HELPWININFO

C.15.1 Synopsis

```
typedef struct {
      int wStructSize;
      int x;
      int y;
      int dx;
      int dy;
      int wMax;
      char rgchMember[2];
} HELPWININFO;
```

C.15.2 Description

The **HELPWININFO** structure contains the secondary help window's size and position information.

Element	Description	
wStructSize	This element is the size, in bytes, of the **HELPWININFO** structure.	
x	This element is the X-coordinate of the help window's upper-left corner.	
y	This element is the Y-coordinate of the help window's upper-left corner.	
cx	This element is the Help window's width.	
cy	This element is the Help window's height.	
wMax	This element determines whether the window should be maximized or set to the specified position and size. The element can be assigned one of the following values:	
	TRUE	This value means that the window should be maximized.
	FALSE	This value means that the window's position and size should be set using the values in the structure's **x, y, cx,** and **cy** elements.
rgchMember	This element is the buffer containing a null-terminated string that is the name of the help window.	

The Help file viewer uses a logical screen coordinate system of 1024x1024 when sizing and positioning help windows. For example, a secondary window with the following position information would fill the upper-right quadrant of the display:

x	512
y	0
cx	512
cy	1024

C.15.3 Cross-References

WinHelp()

C.16 LOGBRUSH

C.16.1 Synopsis

```
typedef struct tagLOGBRUSH {
    UINT lbStyle;
    COLORREF lbColor;
    int lbHatch;
} LOGBRUSH;
```

C.16.2 Description

The **LOGBRUSH** structure contains a physical brush's style, color, and pattern.

Element	Description	
lbStyle	This element is the brush's style. One of the following constant values may be assigned to this element:	
	BS_DIBPATTERN	This value is a pattern brush defined by a device-independent bitmap (DIB).
	BS_HATCHED	This value is a hatched brush.
	BS_HOLLOW	This value is a hollow brush.
	BS_PATTERN	This value is a pattern brush defined by a memory bitmap.
	BS_NULL	This value is the same as BS_HOLLOW.

BS_SOLID	This value is a solid brush.

lbColor This element is the brush's color. In some cases, the meaning of this element depends on the value of the **lbStyle** element.

If the **LOGBRUSH** structure's **lbStyle** element is the value BS_HOLLOW or BS_PATTERN, the **lbColor** element is ignored.

If the **LOGBRUSH** structure's **lbStyle** element is the value BS_ DIBPATTERN, the **lbColor** element should specify whether the pattern bitmap's **BITMAPINFO** structure's **bmiColors** element contains explicit RGB values or indexes into the currently realized logical palette. In this case, the low-order word of **lbColor** should contain one of the following values:

DIB_PAL_COLORS	This value is the pattern bitmap's color table is an array of 16-bit indexes into the currently realized logical palette.
DIB_RGB_COLORS	This value is the pattern bitmap's color table contains RGB values.

lbHatch This element is the brush's hatch style. The meaning of this element depends on the value of the **lbStyle** element.

If the **LOGBRUSH** structure's **lbStyle** element is the value BS_ DIBPATTERN, the **lbHatch** element is a handle to a packed DIB. A packed DIB is a **BITMAPINFO** structure followed by the array of bytes that define the pixels of the bitmap.

If the **LOGBRUSH** structure's **lbStyle** element is the value BS_HATCHED style, the **lbHatch** element determines the orientation of the hatch lines and can be one of the following values:

HS_BDIAGONAL	This value is the left to right, 45-degree upward hatch.
HS_CROSS	This value is the horizontal and vertical cross-hatch.
HS_DIAGCROSS	This value is the 45-degree cross-hatch.
HS_FDIAGONAL	This value is the left to right, 45-degree downward hatch.
HS_HORIZONTAL	This value is the horizontal hatch.
HS_VERTICAL	This value is the vertical hatch.

If the **LOGBRUSH** structure's **lbStyle** element is the value BS_ PATTERN, the **lbHatch** element is a handle to a bitmap that defines the pattern.

If the **LOGBRUSH** structure's **lbStyle** element is the value BS_ SOLID or BS_HOLLOW, the **lbHatch** element is not used.

C.16.3 Cross-References

BITMAPINFO, *CreateBrushIndirect()*, *CreateBrushIndirect()*

C.17 LOGFONT

C.17.1 Synopsis

typedef struct tagLOGFONT {

int lfHeight;

int lfWidth;

int lfEscapement;

int lfOrientation;

int lfWeight;

BYTE lfItalic;

```
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE lfClipPrecision;
    BYTE lfQuality;
    BYTE lfPitchAndFamily;
    BYTE lfFaceName[LF_FACESIZE];
} LOGFONT;
```

C.17.2 Description

The **LOGFONT** structure contains a logical font's attributes.

Element	Description
lfHeight	This element is the height of the font in logical units. If the value of **lfHeight** is less than zero, it is assumed to be the font's character height (cell height minus the internal leading). If the value of **lfHeight** is zero, the system maps the font using the default height. If all of the fonts are larger than the requested font size, the system picks the smallest font. Otherwise, the system chooses the largest physical font that is not larger than the requested font size. The absolute value of **lfHeight** must not be greater than 16,384 after the value is converted into device units.
lfWidth	This element is the average width of font characters in logical units. If the value of **lfWidth** is zero, the system chooses a default font width that is reasonable when considering the font's height. This is done by matching the output device's aspect ratio with the available fonts' digitization aspect ratio. Each character in a TrueType font is scaled by dividing the value of **lfWidth** by the character's average character width.
lfEscapement	This element is the angle between a character's base line and the x-axis in tenths of degrees. The way in which the angle is measured depends on the orientation of the coordinate system. When the y direction is down (left-handed coordinate system), the angle is measured in a counterclockwise direction from the x-axis. When the y direction is up (right-handed coordinate system), the angle is measured in a clockwise direction from the x-axis.
lfOrientation	This element is the orientation of the characters. This value of this element is not used.
lfWeight	This element is the weight of the font. The **lfWeight** element can be assigned one of the following constant values (not all fonts support all of the weights listed below): FW_DONTCARE (Use font's default weight)

FW_THIN	
FW_EXTRALIGHT	(Same as FW_ULTRALIGHT)
FW_ULTRALIGHT	(Same as FW_ EXTRALIGHT)
FW_LIGHT	
FW_NORMAL	(Same as FW_ REGULAR)
FW_REGULAR	(Same as FW_ NORMAL)
FW_MEDIUM	
FW_SEMIBOLD	(Same as FW_ DEMIBOLD)
FW_DEMIBOLD	(Same as FW_ SEMIBOLD)

FW_BOLD	
FW_EXTRABOLD	(Same as FW_ ULTRABOLD)
FW_ULTRABOLD	(Same as FW_ EXTRABOLD)
FW_BLACK	(Same as FW_ HEAVY)
FW_HEAVY	(Same as FW_ BLACK)

lfItalic The value of **lfItalic** determines whether the font is italic. Its value is TRUE if the font is italic and FALSE if the font is not italic.

lfUnderline The value of **lfUnderline** determines whether the font is underlined. It is TRUE if the font is underlined and FALSE if the font is not underlined.

lfStrikeOut The value of **lfStrikeOut** determines whether the font is struck out. It is TRUE if the font is struck out and FALSE if the font is not struck out.

lfCharSet The **lfCharSet** element determines the font's character set, and can be assigned one of the following constant values:

ANSI_CHARSET

DEFAULT_CHARSET

SYMBOL_CHARSET

SHIFTJIS_CHARSET

OEM_CHARSET

The OEM character set is system-dependent.

The system's font mapper does not use the DEFAULT_CHARSET value. For this reason, the DEFAULT_CHARSET value should be used with the understanding that unexpected font mapping results may occur. If an application uses the DEFAULT_CHARSET value and the font name does not exist, a font from any character set can be substituted for the requested font.

If an application uses a font that has an unknown character set, the application should not attempt to translate or interpret strings that are to be rendered with that font.

IfOutPrecision How closely the output must match the requested font's character orientation, escapement, height, pitch, and width. The **lfOutPrecision** element can be assigned one of the following constant values:

OUT_CHARACTER_PRECIS	OUT_STRING_PRECIS
OUT_DEFAULT_PRECIS	OUT_STROKE_PRECIS
OUT_DEVICE_PRECIS	OUT_TT_PRECIS
OUT_RASTER_PRECIS	OUT_TT_ONLY_PRECIS

The values OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS, and OUT_TT_PRECIS can be used to control how the system's font mapper chooses a font when the system contains more than one font with a given name. For example, specifying the OUT_TT_PRECIS value forces the system's font mapper to choose a TrueType version of a font or to choose a TrueType font whenever the specified font name matches a device or raster font, even when there is no TrueType font with the same name.

The value OUT_TT_ONLY_PRECIS can be used to signify the exclusive use of only TrueType fonts. The system's font mapper chooses a TrueType font even when the font's face name matches a raster or vector font.

lfClipPrecision This element determines how to clip characters that are partially outside the clipping region. The **lfClipPrecision** element can be assigned one or more of the following constant values OR'ed together:

CLIP_CHARACTER_PRECIS	CLIP_MASK
CLIP_DEFAULT_PRECIS	CLIP_STROKE_PRECIS
CLIP_EMBEDDED	CLIP_TT_ALWAYS
CLIP_LH_ANGLES	

An application that wishes to use an embedded read-only font must use the CLIP_EMBEDDED value.

An application that wishes to have consistent rotation of device, TrueType, and vector fonts should use the CLIP_LH_ANGLES value. When CLIP_LH_ANGLES is not used, device fonts are always rotated counter-clockwise and the rotation of other fonts is dependent on the orientation of the coordinate system. When CLIP_LH_ANGLES is used, the rotation of all fonts is dependent on the orientation of the coordinate system.

lfQuality This element determines how carefully the graphics device interface (GDI) must attempt to match the attributes of the logical-font to the physical font. The **lfQuality** element can be assigned one of the following constant values:

DEFAULT_QUALITY	This value means that the font's appearance does not matter.
DRAFT_QUALITY	This value means that the font's appearance is less important than when the PROOF_QUALITY value is used. For a GDI raster font, scaling is enabled. If necessary, bold, italic, underline, and strikeout fonts are synthesized.
PROOF_QUALITY	This value means that the font's character quality is more important than the exact matching of the logical-font attributes. For a GDI raster font, scaling is disabled and the font closest in size is chosen. If necessary, bold, italic, underline, and strikeout fonts are synthesized.

lfPitchAndFamily

This element determines the font's family and pitch. The two low-order bits of the **lfPitchAndFamily** value contain the font's pitch and can be one of the following constant values:

DEFAULT_PITCH

FIXED_PITCH

VARIABLE_PITCH

A font family describes how a font looks in a general way. It is intended as a way in which to specify a font when the exact desired typeface is not available. The four high-order bits of the **lfPitchAndFamily** value contain the font's family and can be one of the following constant values:

FF_DECORATIVE	This value specifies a novelty font family, such as Old English.
FF_DONTCARE	This value means that a font's family is unimportant or unknown.
FF_MODERN	This value specifies a font with a constant stroke width, with or without serifs (for example, Pica, Elite, or Courier New).
FF_ROMAN	This value specifies a font with a variable stroke width and with serifs (for example, Times New Roman and New Century Schoolbook).
FF_SCRIPT	This value specifies a font that looks like handwriting (for example., Script and Cursive).
FF_SWISS	This value specifies a font with a variable stroke width and without serifs (for example, MS Sans Serif).

lfFaceName	This element specifies the font's typeface name. The length of name must not exceed LF_FACESIZE - 1. If the value of lfFaceName is NULL, GDI will use a device-dependent typeface.

C.17.3 Cross-References

CreateFontIndirect(), EnumFontFamilies()

C.18 LOGPALETTE

C.18 1 Synopsis

```
typedef struct tagLOGPALETTE {
    WORD palVersion;
    WORD palNumEntries;
    PALETTEENTRY palPalEntry[1];
} LOGPALETTE;
```

C.18.2 Description

The **LOGPALETTE** structure contains a logical color palette's attributes.

Element	Description
palVersion	This element specifies the version of the **LOGPALETTE** structure.
palNumEntries	This element specifies the number of **PALETTEENTRY** structures in the **palPalEntry** array.
palPalEntry	This element specifies the colors of the logical palette and their usage. The array entries are in order of their importance.

C.18.3 Cross-References

CreatePalette(), **PALETTEENTRY**

C.19 LOGPEN

C.19.1 Synopsis

```
typedef struct tagLOGPEN {
    UINT lopnStyle;
    POINT lopnWidth;
    COLORREF lopnColor;
} LOGPEN;
```

C.19.2 Description

The **LOGPEN** structure contains a logical pen's attributes.

Element	Description
lopnStyle	This element is the pen's style type. This **lopnStyle** element can be one of the following values:

PS_SOLID	This value specifies a solid pen.
PS_DASH	This value specifies a dashed pen. The value of **lopnWidth** element must be 1.
PS_DOT	This value specifies a dotted pen. The value of **lopnWidth** element must be 1.
PS_DASHDOT	This value specifies a pen with dashes and dots. The value of **lopnWidth** element must be 1.

PS_DASHDOTDOT	This value specifies a pen with dashes and double dots. The value of **lopnWidth** element must be 1.
PS_NULL	This value specifies a null pen.
PS_INSIDEFRAME	This value specifies that the pen will only be allowed to draw inside of a closed shape that was created by a GDI function that supports a bounding rectangle (for example, *Rectangle()*). If the shape was created by a GDI function that does not support a bounding rectangle, the pen's drawing area will not be limited by a frame. When the pen's width is less than or equal to 1, the PS_INSIDEFRAME style is the same as the PS_SOLID style. If the *Ellipse(), Rectangle(),* and *RoundRect()* functions were not used to created the object, a part of the line may not be completely inside the closed shape.

lopnWidth	This element is the pen's width in logical units. If the value of **lopnWidth** is zero, regardless of the mapping mode, the pen is one pixel wide on raster devices. The **POINT** structure's **y** element is not used.
lopnColor	This element is the pen's color. If the pen's style is PS_INSIDEFRAME, and **lopnColor** does not match a color in the logical color table, the pen is drawn with a dithered color. The PS_SOLID style cannot be used to create a pen with a dithered color.

C.19.3 Cross-References

CreatePenIndirect(), Ellipse(), LineTo(), MoveTo(), **POINT**, *Rectangle(), RoundRect()*

C.20 MDICREATESTRUCT

C.20.1 Synopsis

```
typedef struct tagMDICREATESTRUCT {
    LPCSTR szClass;
    LPCSTR szTitle;
    HINSTANCE hOwner;
    int x;
    int y;
    int cx;
    int cy;
    DWORD style;
    LPARAM lParam;
} MDICREATESTRUCT;
```

C.20.2 Description

The **MDICREATESTRUCT** structure contains multiple document interface (MDI) child window's information.

Element	**Description**
szClass	This element is the pointer to the child window's class name.
szTitle	This element is the pointer to the child window's title.
hOwner	This element is the instance handle of the application that is creating the MDI child window.
x	This element is the initial x-coordinate position of the MDI child window's upper left-hand corner. If the value of the x element is the constant value CW_USEDEFAULT, the system will use a default value.

y	This element is the initial y-coordinate position of the MDI child window's upper left-hand corner. If the value of the y element is the constant value CW_USEDEFAULT, the system will use a default value.
cx	This element is the MDI child window's initial width. If the value of the cx element is the constant value CW_USEDEFAULT, the system will use a default value.
cy	This element is the MDI child window's initial height. If the value of the cy element is the constant value CW_USEDEFAULT, the system will use a default value.
style	This element is the MDI child window's additional styles. If the MDI client window was created using the MDIS_ALLCHILDSTYLES window style, it can use any of the window styles that can be passed to the CreateWindow() function. If the MDI client window was not created using the MDIS_ALLCHILDSTYLES window style, the value of the style element can be one or more of the following constant values OR'ed together:
WS_MINIMIZE	This value minimizes the window when it is created.
WS_MAXIMIZE	This value maximizes the window when it is created.
WS_HSCROLL	This value creates a horizontal scroll bar for the window.
WS_VSCROLL	This value creates a vertical scroll bar for the window.
lParam	Application-specific value.

C.20.3 Cross-References

CREATESTRUCT, *CreateWindow()*

C.21 MEASUREITEMSTRUCT

C.21.1 Synopsis

```
typedef struct tagMEASUREITEMSTRUCT {
    UINT CtlType;
    UINT CtlID;
    UINT itemID;
    UINT itemWidth;
    UINT itemHeight;
    DWORD itemData;
} MEASUREITEMSTRUCT;
```

C.21.2 Description

The **MEASUREITEMSTRUCT** structure contains the dimensions of an owner-drawn control.

Element	**Description**	
CtlType	This element is the type of control. It can contain one of the following values:	
	ODT_BUTTON	This value specifies an owner-drawn button.
	ODT_COMBOBOX	This value specifies an owner-drawn combo box.
	ODT_LISTBOX	This value specifies an owner-drawn list box .
	ODT_MENU	This value specifies an owner-drawn menu.
CtlID	This element is the control's identifier. It is not used for menu controls.	
itemID	This element is the identifier of the list-box item in a variable-height combo box or list box, or the menu-item identifier for a menu control. The **itemID** element is not used for a fixed-height combo box or list box or for a button.	

itemWidth	This element is the menu item's width. Before returning from the WM_MEASUREITEM message, the owner of the owner-drawn menu item must assign a value to this element.
itemHeight	This element is the height of an item in a list box or a menu. Before returning from the WM_MEASUREITEM message, the owner of the owner-drawn combo box, list box, or menu item must assign a value to this element. The value of **itemHeight** cannot be greater than 255.
itemData	This element is the application-defined data that was passed to the combo box or list box in the *lParam* parameter of CB_ADDSTRING, CB_INSERTSTRING, LB_ADDSTRING, or LB_INSERTSTRING.

C.21.3 Cross-References

CB_ADDSTRING, CB_INSERTSTRING, LB_ADDSTRING, LB_INSERTSTRING, WM_MEASUREITEM

C.22 MENUITEMTEMPLATE

C.22.1 Synopsis

typedef struct {

UINT mtOption;

UINT mtID;

char mtString[1];

} MENUITEMTEMPLATE;

C.22.2 Description

The **MENUITEMTEMPLATE** structure contains information about a menu item.

Element	**Description**	
mtOption	This element is the menu item's appearance. The element can contain one or more of the following values OR'ed together:	
	MF_CHECKED	The menu item has a check mark next to it.
	MF_GRAYED	The menu item is inactive and drawn with the gray selection.
	MF_HELP	The menu item has a vertical separator to its left.
	MF_MENUBARBREAK	The menu item is placed in a new column. The old and new columns are separated by a bar.
	MF_MENUBREAK	The menu item is placed in a new column.
	MF_OWNERDRAW	The menu's owner draws all visual parts of the menu item (for example, highlighted, checked and inactive states). This value is not valid for a top-level menu item.
	MF_POPUP	The menu item is a pop-up that displays a sublist of menu items when selected.
	mtID	This element is the menu item's identifier. Not used if the structure's **mtOption** element contains the MF_POPUP value.
mtString	This element is the null-terminated string containing the menu item's name.	

C.22.3 Cross-References

LoadMenuIndirect(), MENUITEMTEMPLATEHEADER

C.23 MENUITEMTEMPLATEHEADER

C.23.1 Synopsis

typedef struct {

```
    UINT versionNumber;
    UINT offset;
} MENUITEMTEMPLATEHEADER;
```

C.23.2 Description

The **MENUITEMTEMPLATEHEADER** structure contains the header information for a menu-item list.

Element	Description
versionNumber	This element is the **MENUITEMTEMPLATEHEADER** structure's version number.
offset	This element is the number of bytes from the end of this structure to where the menu-item list begins.

C.23.3 Cross-References

MENUITEMTEMPLATE

C.24 MINMAXINFO

C.24.1 Synopsis

```
typedef struct tagMINMAXINFO {
    POINT ptReserved;
    POINT ptMaxSize;
    POINT ptMaxPosition;
    POINT ptMinTrackSize;
    POINT ptMaxTrackSize;
} MINMAXINFO;
```

C.24.2 Description

The **MINMAXINFO** structure contains a window's maximized size and position and tracking size.

Element	Description
ptReserved	This element is reserved by the system.
ptMaxSize	This element is the window's maximized width and height. The **POINT** structure's **x** element contains the window's maximized width. The **POINT** structure's **y** element contains the window's maximized height.
ptMaxPosition	This element is the window's maximized position. The **POINT** structure's **x** element contains the x-coordinate of the window's top-left corner. The **POINT** structure's **y** element contains the y-coordinate of the window's top-left corner.
PtMinTrackSize	This element is the window's minimum tracking width and height. The **POINT** structure's **x** element contains the window's minimum tracking width. The **POINT** structure's **y** element contains the window's minimum tracking height.
PtMaxTrackSize	This element is the window's maximum tracking width and height. The **POINT** structure's **x** element contains the window's maximum tracking width. The **POINT** structure's **y** element contains the window's maximum tracking height.

C.24.3 Cross-References

POINT, WM_GETMINMAXINFO

C.25 MSG

C.25.1 Synopsis

```
typedef struct tagMSG {
      HWND hwnd;
      UINT message;
      WPARAM wParam;
      LPARAM lParam;
      DWORD time;
      POINT pt;
} MSG;
```

C.25.2 Description

The **MSG** structure contains a message's information.

Element	Description
hwnd	This element indicates a window that receives the message.
message	This element is a message number.
wParam	This element is additional information specific to the message.
lParam	This element is additional information specific to the message.
time	This element is the time at which the message was posted.
pt	This element is the cursor's position, in screen coordinates, at the time that the message was posted.

C.25.3 Cross-References

GetMessage(), TranslateMessage(), DispatchMessage(), TranslateAccelerator()

C.26 NEWTEXTMETRIC

C.26.1 Synopsis

```
typedef struct tagNEWTEXTMETRIC {
      int tmHeight;
      int tmAscent;
      int tmDescent;
      int tmInternalLeading;
      int tmExternalLeading;
      int tmAveCharWidth;
      int tmMaxCharWidth;
      int tmWeight;
      BYTE tmItalic;
      BYTE tmUnderlined;
      BYTE tmStruckOut;
      BYTE tmFirstChar;
      BYTE tmLastChar;
      BYTE tmDefaultChar;
```

```
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    int tmOverhang;
    int tmDigitizedAspectX;
    int tmDigitizedAspectY;
    DWORD ntmFlags;
    UINT ntmSizeEM;
    UINT ntmCellHeight;
    UINT ntmAvgWidth;
} NEWTEXTMETRIC;
```

C.26.2 Description

The **NEWTEXTMETRIC** structure contains information about a physical font. The structure is an extension of the **TEXTMETRIC** structure.

Element	**Description**
tmHeight	This element is the character cell's height. It is the sum the values in the structure's **tmAscent** and **tmDescent** elements.
tmAscent	This element is the character cell's ascent. It is the space between the base line and the top of the character cell.
tmDescent	This element is the character cell's descent. It is the space between the bottom of the character cell and the base line.
tmInternalLeading	This element is the difference between the font's physical size and the font's point size. If the font is a TrueType font, the value of the **tmInternalLeading** element is equal to the value of **tmHeight** - (ScaleFactor * ntmSizeEM), where **ScaleFactor** is the font's scaling factor. If the font is a bitmap font, the value of the **tmInternalLeading** element is used to specify the font's point size. During a request for a logical font, if the **LOGFONT** structure's **lfHeight** element contains a negative value, the height of the font being requested equals the value of the **tmHeight** element minus the **tmInternalLeading** element.
TmExternalLeading	This element is the amount of extra leading space that the application adds between rows. This area is outside of the character cell and will therefore contains no marks and is not altered by text output calls using either the opaque or transparent modes. A font designer sometimes sets the value of this element to zero.
TmAveCharWidth	This element is the average width of the font's characters. If a font uses the ANSI character set (ANSI_CHARSET), the value of **tmAveCharWidth** is a weighted average width of the characters 'a' -'z' and the space character. For fonts that use other character sets, the value of **tmAveCharWidth** is an unweighted average of all characters in the font.
tmMaxCharWidth	This element is the "B" spacing of the font's widest character.
tmWeight	This element is the weight of the font. The **tmWeight** element can be assigned one of the following constant values:

FW_DONTCARE	(Use font's default weight)
FW_THIN	
FW_EXTRALIGHT	(Same as FW_ULTRALIGHT)

FW_ULTRALIGHT	(Same as FW_ EXTRALIGHT)
FW_LIGHT	
FW_NORMAL	(Same as FW_ REGULAR)
FW_REGULAR	(Same as FW_ NORMAL)
FW_MEDIUM	
FW_SEMIBOLD	(Same as FW_ DEMIBOLD)
FW_DEMIBOLD	(Same as FW_ SEMIBOLD)
FW_BOLD	
FW_EXTRABOLD	(Same as FW_ ULTRABOLD)
FW_ULTRABOLD	(Same as FW_ EXTRABOLD)
FW_BLACK	(Same as FW_ HEAVY)
FW_HEAVY	(Same as FW_ BLACK)

tmItalic This element means the font is italic. The value of **tmItalic** is TRUE if the font is italic and FALSE if the font is not italic.

tmUnderlined This element means the font is underlined. The value of **tmUnderlined** is TRUE if the font is underlined and FALSE if the font is not underlined.

tmStruckOut This element means the font is struck out. The value of **tmStruckOut** is TRUE if the font is struck out and FALSE if the font is not struck out.

tmFirstChar This element is the value of the font's first character.

tmLastChar This element is the value of the font's last character.

tmDefaultChar This element is the value of the character that is substituted for characters not found in the font.

tmBreakChar This element is the value of the character that is used to define word breaks for text justification.

tmPitchAndFamily

This element is the font's pitch and family.

The value of the four low-order bits of the **tmPitchAndFamily** element specifies the type of font and can be one or more of the following constant values OR'ed together:

TMPF_FIXED_PITCH	This value specifies a fixed-pitch font.
TMPF_VECTOR	This value specifies a vector or TrueType font.
TMPF_TRUETYPE	This value specifies a TrueType font. that can be used on a printer and display.
TMPF_DEVICE	This value specifies a device font. Set for downloaded and device-resident fonts.

For example, the TrueType font Courier New® uses the TMPF_FIXED_PITCH, TMPF_VECTOR, and TMPF_TRUETYPE constants.

The value of the four high-order bits of the **tmPitchAndFamily** element specifies the font family and can be one of the following constant values:

FF_DECORATIVE	This value specifies a novelty font family, such as Old English.
FF_DONTCARE	This value means that the font's family is unimportant or unknown.
FF_MODERN	This value specifies a font with a constant stroke width and with or without serifs (for example, Pica, Elite, or Courier New).

FF_ROMAN	Font with a variable stroke width and with serifs (for example, Times New Roman and New Century Schoolbook).
FF_SCRIPT	Font that looks like handwriting (for example, Script and Cursive).
FF_SWISS	Font with a variable stroke width and without serifs (for example, MS Sans Serif).

tmCharSet This element is the font's character set. The **tmCharSet** element can be assigned one of the following constant values:

ANSI_CHARSET	0
DEFAULT_CHARSET	1
SYMBOL_CHARSET	2
SHIFTJIS_CHARSET	128
OEM_CHARSET	255

tmOverhang This element is extra width that is added to some synthesized fonts. The GDI or a device will add width to a string on a per-character and per-string basis when synthesizing such as bold or italic.

The value of the **tmOverhang** element is zero for many italic and bold TrueType fonts because many TrueType fonts include non-synthesized italic and bold faces.

The value of a Raster font's overhang can be used to determine the amount of spacing between words that have different attributes.

tmDigitizedAspectX
This element is the horizontal aspect of the device for which the font was designed.

TmDigitizedAspectY
This element is vertical aspect of the device for which the font was designed.

ntmFlags This element provides more information about the font's style. The **ntmFlags** element can contain one or more of the following constant values OR'ed together:

NTM_REGULAR

NTM_BOLD

NTM_ITALIC

ntmSizeEM This element is the size of font's em square in notional units.

ntmCellHeight This element is the font's height in notional units. This value of the **ntmCellHeight** element should be compared with the value of the **ntmSizeEM** element **sntmAvgWidth**. This element is the average width of the font's characters in notional units. The value of the **ntmAvgWidth** element should be compared with the value of the **ntmSizeEM** element.

C.26.3 Cross-References

EnumFontFamilies(), EnumFonts(), GetDeviceCaps(), GetTextMetrics(), **TEXTMETRIC**

C.27 OFSTRUCT

C.27.1 Synopsis

typedef struct tagOFSTRUCT {

```
    BYTE cBytes;
    BYTE fFixedDisk;
    UINT nErrCode;
    BYTE reserved[4];
    BYTE szPathName[128];
} OFSTRUCT;
```

C.27.2 Description

The **OFSTRUCT** structure contains information about an open file.

Element	Description
cBytes	This element is the size of the **OFSTRUCT** structure in bytes.
fFixedDisk	This element specifies whether the file is on a fixed disk. The value of the **fFixedDisk** element is TRUE if the file is on a fixed disk and FALSE if the file is not on a fixed disk.
nErrCode	If the *OpenFile()* function returns the value -1, the value of the **nErrCode** element is set to one of the following MS-DOS error values:

0x0001	Invalid function
0x0002	File not found
0x0003	Path not found
0x0004	Too many open files
0x0005	Access denied
0x0006	Invalid handle
0x0007	Arena trashed
0x0008	Not enough memory
0x0009	Invalid block
0x000A	Bad environment
0x000B	Bad format
0x000C	Invalid access
0x000D	Invalid data
0x000F	Invalid drive
0x0010	Current directory
0x0011	Not same device
0x0012	No more files
0x0013	Write protect error
0x0014	Bad unit
0x0015	Not ready
0x0016	Bad command
0x0017	CRC error
0x0018	Bad length
0x0019	Seek error
0x001A	Not MS-DOS disk
0x001B	Sector not found
0x001C	Out of paper
0x001D	Write fault
0x001E	Read fault

0x001F	General failure
0x0020	Sharing violation
0x0021	Lock violation
0x0022	Wrong disk
0x0023	File control block unavailable
0x0024	Sharing buffer exceeded
0x0032	Not supported
0x0033	Remote not listed
0x0034	Duplicate name
0x0035	Bad netpath
0x0036	Network busy
0x0037	Device does not exist
0x0038	Too many commands
0x0039	Adaptor hardware error
0x003A	Bad network response
0x003B	Unexpected network error
0x003C	Bad remote adaptor
0x003D	Print queue full
0x003E	No spool space
0x003F	Print canceled
0x0040	Netname deleted
0x0041	Network access denied
0x0042	Bad device type
0x0043	Bad network name
0x0044	Too many names
0x0045	Too many sessions
0x0046	Sharing paused
0x0047	Request not accepted
0x0048	Redirection paused
0x0050	File exists
0x0051	Duplicate file control block
0x0052	Cannot make
0x0053	Interrupt 24 failure
0x0054	Out of structures
0x0055	Already assigned
0x0056	Invalid password
0x0057	Invalid parameter
0x0058	Net write fault

reserved	This element is reserved for future use by the system.
szPathName	This element is a buffer containing the file's path. The characters in the buffer are from the OEM character set.

C.27.3 Cross-References

OpenFile()

C.28 OPENFILENAME

C.28.1 Synopsis

```
typedef struct tagOPENFILENAME {
    DWORD lStructSize;
    HWND hwndOwner;
    HINSTANCE hInstance;
    LPCSTR lpstrFilter;
    LPSTR lpstrCustomFilter;
    DWORD nMaxCustFilter;
    DWORD nFilterIndex;
    LPSTR lpstrFile;
    DWORD nMaxFile;
    LPSTR lpstrFileTitle;
    DWORD nMaxFileTitle;
    LPCSTR lpstrInitialDir;
    LPCSTR lpstrTitle;
    DWORD Flags;
    UINT nFileOffset;
    UINT nFileExtension;
    LPCSTR lpstrDefExt;
    LPARAM lCustData;
    UINT (CALLBACK *lpfnHook) (HWND, UINT, WPARAM, LPARAM);
    LPCSTR lpTemplateName;
} OPENFILENAME;
```

C.28.2 Description

The **OPENFILENAME** structure contains information that is used by the system to initialize the Open and Save common dialog boxes and to return the user's dialog box selections.

Element	Description
lStructSize	This element is the size of the **OPENFILENAME** structure in bytes. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions.
hwndOwner	This element is the handle of the window that owns the common dialog box. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions. If there is no owner, the element's value should be NULL. If the OFN_SHOWHELP flag is set in the **Flags** element, a valid window handle must be assigned to the **hwndOwner** element. If the user selects the dialog box's Help button, the window is sent a notification message. The message's ID is registered at runtime and can be retrieved by calling the *RegisterWindowMessage()* function with the constant HELPMSGSTRING.
hInstance	The element should be assigned the handle of the data block containing the dialog box template given in the **lpTemplateName** element. The value of the **hInstance** element is used only when the OFN_ENABLETEMPLATEor OFN_ENABLETEMPLATEHANDLE constants are used in the **Flags** element. When the

CC_ENABLETEMPLATE constant is used, **hInstance** is an instance handle; when the CC_ENABLETEMPLATEHANDLE constant is used, **hInstance** is a handle to a dialog resource. If either of these two constants are used, a value must be assigned to the **hInstance** element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions.

lpstrFilter This element is a pointer to a buffer that contains one or more pairs of null-terminated strings representing file name filters. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions.

The first string in the pair of string is a description of the file filter (for example, "Help Files"). The second string in the pair of string is the actual file filter pattern (for example, "*.hlp"). Multiple file filter patterns can be associated with a single file filter description by separating each pattern with a semicolon (;) character (for example, "*.txt;*.doc;*.hlp"). Two NULL characters must appear after the last file filter pattern string to denote the end of the entire string in the buffer.

If the value of the lpstrFilter element is NULL, no filters are shown in the dialog box.

lpstrCustomFilter

This element is a pointer to a buffer that contains one or more pairs of null-terminated, custom strings representing file name filters. The strings are formatted in the same manner as the **lpstrFilter** element's file filter strings. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions.

If the value of the **lpstrFilter** element is not NULL, after the user closes the dialog box with the OK button, the system will always copy the file filter pattern from the "File Name" edit control to the second string location within the buffer.

When the value of the **nFilterIndex** element is zero, the string in the **lpstrCustomFilter** buffer is used as the dialog box's initial filter description and filter pattern. In this case, if the first string in the first pair of strings is a NULL string (for example, "", "*.hlp"), only the string in the **lpstrFilter** buffer is displayed in the dialog box's "List Files of Type" list box.

The **lpstrCustomFilter** buffer should be at least 40 bytes in size.

nMaxCustFilter The size of the **lpstrCustomFilter** buffer in bytes. Not used if the value of the **lpstrCustomFilter** element is NULL.

nFilterIndex The index number of the file filter to use when the common dialog box is first shown. An index value of 1, for example, will cause the first file filter string pair in the **lpstrFilter** buffer to be initially shown. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions.

If the value of the **nFilterIndex** member is zero and the value of the **lpstrCustomFilter** element is not NULL, the first filter in the **lpstrCustomFilter** buffer is used.

If the value of the **nFilterIndex** member is zero, the value of the **lpstrCustomFilter** element is NULL, or the value of the **lpstrCustomFilter** element is not NULL but the first string in the **lpstrCustomFilter** buffer is a NULL string, the first filter in the **lpstrFilter** buffer is used.

If the buffer pointed to by the **lpstrFilter** element should be used, but the value of the element is NULL, no file filter is used and no files is shown in the "File Name" list box.

After the user closes the dialog box with the OK button, the system will assign the index of the last selected file filter to the **nFilterIndex** element.

lpstrFile This element is a pointer to a buffer that contains a filename string to copy to the "File Name" edit control when the common dialog box is initialized. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions. If the initialization operation is not desired, the first character in the string should be NULL.

After the user closes the dialog box with the OK button, the selected file's complete path is copied into the **lpstrFile** buffer.

If the file path string is too large to fit in the buffer, the required size, in bytes, of the string is placed in the buffer instead of the string and the common dialog box's procedure will return zero. In this case, the application should cast the **lpstrFile** element to type LPWORD.

The size of the buffer pointed to by the **lpstrFile** element must be at least three bytes in order to receive the path size. If the lpstrFile buffer is too small, the *CommDlgExtendedError()* function will return the FNERR_BUFFERTOOSMALL value.

nMaxFile This element is the size of the **lpstrFile** buffer in bytes. Not used if the value of the **lpstrFile** element is NULL.

lpstrFileTitle This element is a pointer to a buffer in to which the common dialog box's procedure will copy the filename and extension of the file selected by the user. If the value of the **lpstrFileTitle** element is NULL, the copy operation will not be performed by the common dialog box's procedure.

nMaxFileTitle This element is the maximum size of a string, in bytes, that can be copied into the **lpstrFile** buffer. If the value of the **lpstrFileTitle** element is NULL, the **nMaxFileTitle** element is not used.

l**pstrInitialDir** This element is a pointer to a buffer that contains a string representing the initial file directory to use when the common dialog is displayed. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions.

If the value of the **lpstrInitialDir** element is NULL, the current directory is used as the initial directory.

If the **lpstrFile** buffer contains a valid path to a file, that file directory is initially used instead of the file directory given in **lpstrInitialDir**.

lpstrTitle This element is a pointer to a null-terminated string that is a custom title for the common dialog box. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions.

If the value of the **lpstrTitle** element is NULL, the dialog box's default title is shown.

Flags This element determines how the common dialog box is initialized. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions. The value of this element may the one or more of the following constant values OR'ed together:

OFN_ALLOWMULTISELECT
This value allows the "File Name" list box to have multiple selections. The **lpstrFile** buffer will contain a single path followed by all of the selected filenames. The path and each filename are separated from one another by a single one space character.

A filename may be preceded by a relative path. The buffer could, for example, look something like this:

c:\files file1.txt file2.txt ..\bin\file3.txt

OFN_CREATEPROMPT This value allows the user to create a new file. Automatically sets the OFN_PATHMUSTEXIST and OFN_FILEMUSTEXIST constants.

OFN_ENABLEHOOK This value uses the hook function given in the structure's **lpfnHook** element.

OFN_ENABLETEMPLATE
This value uses the dialog box template given in the **hInstance** and **lpTemplateName** elements.

OFN_ENABLETEMPLATEHANDLE
The **hInstance** element is a data block that has a pre-loaded dialog box template. The **lpTemplateName** element should be ignored.

OFN_EXTENSIONDIFFERENT	This value sets the common dialog box's procedure to indicate that the returned filename's extension is different from the extension given in the **lpstrDefExt** element. The constant will not be set if the value of the **lpstrDefExt** element is NULL, if the file extensions match, or if the returned filename has no extension
OFN_FILEMUSTEXIST	This value warns the user when they type an invalid name into the "File Name" edit control; only allows valid filenames. Automatically sets the OFN_PATHMUSTEXIST constant.
OFN_HIDEREADONLY	This value hides the dialog box's "Read Only" check box.
OFN_NOCHANGEDIR	This value resets the current directory to what it was when the dialog box was created.
OFN_NOREADONLYRETURN	The returned file will not have the Read Only attribute and will not be in a write-protected directory.
OFN_NOTESTFILECREATE	The file will not be created before the dialog box is closed and the system will not check against write protection, a full disk, an open drive door, or network protection. This constant is usually used when an application saves the file on a create-no-modify network share point.
OFN_NOVALIDATE	This value allows the returned filename to contain invalid characters. In order to check the filename, an application can use a hook function that responds to the FILEOKSTRING registered message. If the edit control's text is empty or if it contains only spaces, the lists of files and directories are updated. If the edit control's text is not empty and does contain only spaces, the structure's **nFileOffset** and **nFileExtension** elements are updated. A default extension will not be added to the text and text will not be copied to the **lpstrFileTitle** buffer. If the value of the **nFileOffset** element is negative, the returned filename is invalid. If the value of the **nFileOffset** element is not negative, the filename is valid and the values of the **nFileOffset** and **nFileExtension** elements can be used as if the OFN_NOVALIDATE constant had not been set at all.
OFN_OVERWRITEPROMPT	If the selected file already exists, the Save As common dialog box will display a message box and the user must confirm the file overwriting action.
OFN_PATHMUSTEXIST	This value warns the user when they type in an invalid path into the "File Name" edit control; only allows valid paths.
OFN_READONLY	When the dialog box is created, this value checks the "Read Only" check box. After the user closes the dialog box with the OK button, this constant is set if the "Read Only" check box is checked.

	OFN_SHAREAWARE	If a network sharing violation occurs when the *OpenFile()* function is called, this value ignores the error and returns the given filename.

If this constant is not used, the registered message for SHAREVISTRING is sent to the hook function. The message's *lParam* value will contain a pointer to a null-terminated string for the path name. The hook function can return one of the following constant values:

OFN_SHAREFALLTHROUGH - This value returns the filename from the dialog box.

OFN_SHARENOWARN - This value performs no further action.

OFN_SHAREWARN - This value displays the standard warning message for the error. This is the default action when there is not a hook function.

OFN_SHOWHELP This value displays the Help button in the dialog box.

nFileOffset After the user closes the dialog box with the OK button, this **nFileOffset** element will contain a zero-based offset value from the beginning of the **lpstrFile** buffer to the selected file's filename.

nFileExtension After the user closes the dialog box with the OK button, this **nFileOffset** element will contain a zero-based offset value from the beginning of the **lpstrFile** buffer to the selected file's extension.

lpstrDefExt This element is a pointer to a null-terminated string that is a default file extension. If the user does not enter a file extension into the "File Name" edit control, the common dialog box procedure internally appends the default extension to the file's name and looks for the file. If the search fails, the common dialog box procedure searches for the file using the exact filename information that the user entered. A value must be assigned to this element before the structure is passed to the *GetOpenFileName()* or *GetSaveFileName()* functions.

Only the first three characters of the string are used. The string should not contain a period character.

If the value of the **lpstrDefExt** element is NULL and the user does not type an extension into the "File Name" edit control, no extension is appended to the user's entry.

lCustData This element is application-defined data that the system passes to the hook function specified in the structure's **lpfnHook** element when the dialog box is initialized.

lpfnHook This element is a pointer to a hook function that processes messages for the common dialog box. The hook function is used only when the OFN_ENABLEHOOK constant is specified in the structure's **Flags** element.

The hook function is sent all of the messages that the dialog box receives. When the dialog box is created, the hook function is sent a WM_INITDIALOG message whose **lParam** contains a pointer to the **FINDREPLACE** structure. This is the only time that the hook function will have access to the application-defined data specified in the **lCustData** element and to the rest of the values stored in the **FINDREPLACE** structure.

The hook function must return TRUE when it processes a message that is sent to it, or FALSE when it does not process a message that is sent to it.

lpTemplateName

This element is a null-terminated string containing the name of the resource file that has an application-defined dialog box template that is to be substituted for the standard common dialog box's template. This element is used only when the OFN_ENABLETEMPLATE constant is specified in the structure's **Flags** element The MAKEINTRESOURCE macro can be used if the dialog box resource is numbered.

C.28.3 Cross-References

GetOpenFileName(), *GetSaveFileName()*, MAKEINTRESOURCE

C.29 PAINTSTRUCT

C.29.1 Synopsis

```
typedef struct tagPAINTSTRUCT {
    HDC hdc;
    BOOL fErase;
    RECT rcPaint;
    BOOL fRestore;
    BOOL fIncUpdate;
    BYTE rgbReserved[16];
} PAINTSTRUCT;
```

C.29.2 Description

The **PAINTSTRUCT** structure contains information that an application can use to paint the client area of a window that it owns.

Element	Description
hdc	This element is the device context to be used when painting.
fErase	This element determines whether the background of the client area needs to be redrawn. If the value of the **fErase** element is TRUE, the background needs to be redrawn. If the value of the **fErase** element is FALSE, the background does not need to be redrawn.
rcPaint	This element is the position information for the area that needs to be painted.
fRestore	This element is reserved for use by the system.
fIncUpdate	This element is reserved for use by the system.
rgbReserved	This element is reserved for use by the system.

C.29.3 Cross-References

BeginPaint(), *EndPaint()*, **WNDCLASS**

C.30 PALETTEENTRY

C.30.1 Synopsis

```
typedef struct tagPALETTEENTRY {
    BYTE peRed;
    BYTE peGreen;
    BYTE peBlue;
    BYTE peFlags;
} PALETTEENTRY;
```

C.30.2 Description

The **PALETTEENTRY** structure contains the color and usage information for an entry in a logical color palette.

Element	Description
peRed	This element is the palette entry's intensity of red.
peGreen	This element is the palette entry's intensity of green.

peBlue	This element is the palette entry's intensity of blue.
peFlags	This element determines how the palette entry is to be used. The value of the **peFlags** element can be one of the following values:

NULL	The system assumes that the the palette entry contains an RGB value that is mapped normally.
PC_EXPLICIT	The low-order word of the logical palette entry is a hardware palette index. An application can use this constant value to show the contents of the display device's palette.
PC_NOCOLLAPSE	The color is placed in an unused system-palette entry instead of being matched to an existing system-palette color. Colors in other logical palettes can be matched to this color. If there are no unused system-palette entries, the color is matched normally.
PC_RESERVED	The logical palette entry is used for palette animation. Because the palette entry's color will change frequently, the use of this constant prevents other windows from matching colors to this palette entry. If there is an unused, available system-palette entry, the color is placed in that entry. If there is not an unused, available system-palette entry, the color will not be available for animation.

C.30.3 Cross-References

AnimatePalette(), **LOGPALETTE**

C.31 POINT

C.31.1 Synopsis

```
typedef struct tagPOINT {
    int x;
    int y;
} POINT;
```

C.31.2 Description

The **POINT** structure contains the x- and y-coordinates of a point.

Element	Description
x	This element is a point's x-coordinate.
y	This element is a point's y-coordinate.

C.31.3 Cross-References

ChildWindowFromPoint(), PtInRect(), WindowFromPoint()

C.32 PRINTDLG

C.32.1 Synopsis

```
typedef struct tagPD {
    DWORD lStructSize;
    HWND hwndOwner;
    HGLOBAL hDevMode;
    HGLOBAL hDevNames;
    HDC hdc;
    DWORD Flags;
    UINT nFromPage;
    UINT nToPage;
    UINT nMinPage;
    UINT nMaxPage;
    UINT nCopies;
    HINSTANCE hInstance;
    LPARAM lCustData;
    UINT (CALLBACK* lpfnPrintHook)(HWND, UINT, WPARAM, LPARAM);
    UINT (CALLBACK* lpfnSetupHook)(HWND, UINT, WPARAM, LPARAM);
    LPCSTR lpPrintTemplateName;
    LPCSTR lpSetupTemplateName;
    HGLOBAL hPrintTemplate;
    HGLOBAL hSetupTemplate;
} PRINTDLG;
```

C.32.2 Description

The **PRINTDLG** structure contains information that is used by the system to initialize the Print common dialog box and to return the user's dialog box selections.

Element	Description
lStructSize	This element is the size of the **PRINTDLG** structure in bytes. A value must be assigned to this element before the structure is passed to the *PrintDlg()* function.
hwndOwner	This element is the handle of the window that owns the common dialog box. A value must be assigned to this element before the structure is passed to the *PrintDlg()* function. If there is no owner, the element's value should be NULL. If the **PD_SHOWHELP** flag is set in the **Flags** element, a valid window handle must be assigned to the **hwndOwner** element. If the user selects the dialog box's Help button, the window is sent a notification message. The message's ID is registered at runtime and can be retrieved by calling the *RegisterWindowMessage()* function with the constant HELPMSGSTRING.
hDevMode	This element identifies a movable global memory object that contains a **DEVMODE** structure. If an application wishes to set the initial state of the Print dialog box's controls, it can allocate the **DEVMODE** structure and assign initial values to the structure's elements. If an application does not wish to set the initial state of the Print dialog box's controls, the value of the **hDevMode** element should be set to NULL. If the value of the **hDevMode** element is NULL, the *PrintDlg()* function will allocate the memory for the **DEVMODE** structure, set the value of its elements, and return a handle that identifies it. If the specified printer's device driver does not support extended device modes, the *PrintDlg()* function will set the value of **hDevMode** to NULL when the *PrintDlg()* function returns. If the device name specified in the **dmDeviceName** element of the **DEVMODE** structure is not in the WIN.INI file's (devices) section, the *PrintDlg()* function will return an error. The value of the **hDevMode** element may be changed by the *PrintDlg()* function. If the value of the **hDevMode** element is changed by the *PrintDlg()* function, it can be assumed that the original handle was freed by the *PrintDlg()* function and that the new handle should be freed by the application.

When the *PrintDlg()* function returns, an application can use the **DEVMODE** structure to determine the last state of the dialog box controls that were associated with the elements in the structure.

hDevNames	This element identifies a movable global memory object that contains a **DEVNAMES** structure. If an application wishes to set the initial state of the Print dialog box's controls, it can allocate the **DEVNAMES** structure and assign initial values to the structure's elements. If an application does not wish to set the initial state of the Print dialog box's controls, the value of the **hDevNames** element should be set to NULL. If the value of the **hDevNames** element is NULL, the *PrintDlg()* function will allocate the memory for the **DEVNAMES** structure, set the value of its elements, and return a handle that identifies it. When the *PrintDlg()* function initially sets the values of the **DEVNAMES** structure's elements, it uses the first port name that appears in the (devices) section of WIN.INI. When the *PrintDlg()* function returns, an application can use the **DEVNAMES** structure to determine the last state of the dialog box controls (for example, the strings in the controls) that were associated with the elements in the structure. An application can, for example, use the information to create a device context or an information context. If the value of the **PRINTDLG** structure's **hDevMode** and **hDevNames** elements are NULL, the *PrintDlg()* function specifies the default printer for **hDevNames**. The value of the **hDevNames** element can be changed by the *PrintDlg()* function. If the value of the **hDevNames** element is changed by the *PrintDlg()* function, it can be assumed that the original handle was freed by the *PrintDlg()* function and that the new handle should be freed by the application.
hdc	When the *PrintDlg()* function is finished, it returns a value in the **hdc** element. The type of value stored in the **hdc** element is dependent on which constant value is set in the **Flags** element, **PD_RETURNDC** or **PC_RETURNIC**. If the **PD_RETURNDC** constant value is used, the value stored in the **hdc** element is a device context matching the selections that the user made in the dialog box. If the **PD_RETURNIC** constant value is used, the value stored in the **hdc** element is an information context matching the selections that the user made in the dialog box. If both constant values are used, the value stored in the **hdc** element is a device context. If neither constant value is used, the value stored in the **hdc** element is undefined.
Flags	These flags determine how the common dialog box is initialized. A value must be assigned to this element before the structure is passed to the *PrintDlg()* function. The value of this element may be one or more of the following constant values OR'ed together:

PD_ALLPAGES	This value selects the "All" Page Range radio button.
PD_COLLATE	This value checks the "Collate Copies" check box.
PD_DISABLEPRINTTOFILE	This value disables the "Print to File" check box.
PD_ENABLEPRINTHOOK	This value uses the hook function given in the structure's **lpfnPrintHook** element.
PD_ENABLEPRINTTEMPLATE	This value uses the dialog box template given in the **hInstance** and **lpPrintTemplateName** elements.
PD_ENABLEPRINTTEMPLATEHANDLE	The **hPrintTemplate** element is a data block that has a pre-loaded dialog box template. The **lpPrintTemplateName** element should be ignored.
PD_ENABLESETUPHOOK	This value uses the hook function given in the structure's **lpfnSetupHook** element.

PD_ENABLESETUPTEMPLATE	This value uses the dialog box template given in the **hInstance** and **lpSetupTemplateName** elements.
PD_ENABLESETUPTEMPLATEHANDLE	The **hSetupTemplate** element is a data block that has a preloaded dialog box template. The **lpSetupTemplateName** element should be ignored.
PD_HIDEPRINTTOFILE	This value hides and disables the "Print to File" check box.
PD_NOPAGENUMS	This value disables the "Pages" radio button and its associated edit controls.
PD_SHOWHELP	This value displays the Help button in the dialog box.
PD_NOSELECTION	This value disables the "Selection" radio button.
PD_NOWARNING	This value does not show a warning message when there is no default printer.
PD_PAGENUMS	This value selects the "Pages" radio button.
PD_PRINTSETUP	This value displays the Print Setup dialog box instead of the Print dialog box.
PD_PRINTTOFILE	This value checks the "Print to File" check box.
PD_RETURNDC	This value returns a device context matching the selections that the user made in the dialog box. Assigns the value of the handle to the device context to the PRINTDLG structure's **hdc** element.
PD_RETURNDEFAULT	This value does not display a dialog box. It returns **DEVMODE** and **DEVNAMES** structures that are initialized for the system default printer. When this constant is used, the value of the **PRINTDLG** structure's **hDevNames** and **hDevMode** elements should be NULL. If the system default printer is supported by an old printer driver, the value of the **hDevMode** element is NULL.
PD_RETURNIC	This value returns an information context matching the selections that the user made in the dialog box. It assigns the value of the handle to the device context to the **PRINTDLG** structure's **hdc** element.
PD_SELECTION	This value selects the "Selection" radio button.
PD _SHOWHELP	This value displays the Help button in the dialog box.
PD_USEDEVMODECOPIES	If a printer driver does not support multiple copies and this constant value is used, this value disables the Copies edit control. If a printer driver does support multiple copies and this constant value is used, the *PrintDlg()* function stores the requested number of copies in the **DEVMODE** structure's **dmCopies** element and the value 1 in the **PRINTDLG** structure's **nCopies** member. If this constant value is not used, the *PrintDlg()* function will store the value 1 in the **DEVMODE** structure's **dmCopies** element and the requested number of copies in the **PRINTDLG** structure's **nCopies** member.

nFromPage This element is the initial value for the dialog box's "From" edit control. When the *PrintDlg()* function returns, the **nFromPage** element contains the page at which to begin printing. The value of the **nFromPage** element should only be used when the PD_PAGENUMS constant is set in the **flag** element. The maximum value that can be stored in the *nFromPage* element is 0xFFFE. If the initial value for the dialog box's "From" edit control is set to 0xFFFF, the edit control is blank.

nToPage This element is the initial value for the dialog box's "To" edit control. When the *PrintDlg()* function returns, the **nToPage** element contains the page at which to stop printing. The value of the **nToPage** element should only be used when the PD_PAGENUMS constant is set in the **Flags** element. The maximum value that can be stored in the **nToPage** element is 0xFFFE. If the initial value for the dialog box's "To" edit control is set to 0xFFFF, the edit control is blank.

nMinPage This element is the minimum number that can be specified in the "From" and "To" edit controls.

nMaxPage This element is the minimum number that can be specified in the "From" and "To" edit controls.

nCopies This element is the initial value for the dialog box's "Copies" edit control when the value of the **hDevMode** elements is NULL.

Before the *PrintDlg()* function returns, it stores a value in the **nCopies** element. The value stored in the **nCopies** element is dependent on the age of the printer driver. For older printer drivers, the **nCopies** element is assigned the number of copies requested by the user in the dialog box's "Copies" edit control. For newer printer drivers, when the PD_USEDEVMODECOPIES constant is not set in the **Flags** element, the **nCopies** element is assigned copies requested by the user. For newer printer drivers, when the PD_USEDEVMODECOPIES constant is set in the **Flags** element, the **nCopies** element is assigned the value 1 and the actual number of copies requested by the user is assigned the **DEVMODE** structure's **dmCopies** element.

hInstance This element should be assigned the handle to the data block containing the dialog box templates given in the **lpPrintTemplateName** and **lpSetupTemplateName** elements.

The value of the **hInstance** element is used only when the PD_ENABLEPRINTTEMPLATE or PD_ENABLESETUPTEMPLATE constants are used in the **Flags** element. When the CC_ENABLETEMPLATE constant is used, **hInstance** is an instance handle; when the CC_ENABLETEMPLATEHANDLE constant is used, **hInstance** is a handle to a dialog resource. If either of these constants are used, a value must be assigned to the **hInstance** element before the structure is passed to the *PrintDlg()* function.

lCustData This element is application-defined data that the system passes to the hook functions specified in the structure's **lpfnPrintHook** and **lpfnSetupHook** elements when the dialog box is initialized.

lpfnPrintHook This element is a pointer to a hook function that processes messages for the Print common dialog box. The hook function is used only when the PD_ENABLEPRINTHOOK constant is specified in the structure's **Flags** element.

The hook function is sent all of the messages that the dialog box receives. When the dialog box is created, the hook function is sent a WM_INITDIALOG message whose **lParam** contains a pointer to the **PRINTDLG** structure. This is the only time that the hook function has access to the application-defined data specified in the **lCustData** element and to the rest of the values stored in the **PRINTDLG** structure.

The hook function must return TRUE when it processes a message that is sent to it, or FALSE when it does not process a message that is sent to it.

lpfnSetupHook This element is a pointer to a hook function that processes messages for the Print Setup common dialog box. The hook function is used only when the PD_ENABLESETUPHOOK constant is specified in the structure's **Flags** element.

The hook function is sent all of the messages that the dialog box receives. When the dialog box is created, the hook function is sent a WM_INITDIALOG message whose *lParam* contains a pointer to the **PRINTDLG** structure. This is the only time that the hook function will have access to the application-defined data specified in the **lCustData** element and to the rest of the values stored in the **PRINTDLG** structure.

The hook function must return TRUE when it processes a message that is sent to it, or FALSE when it does not process a message that is sent to it.

LpPrintTemplateName
This element is a null-terminated string containing the name of the resource file that has an application-defined dialog box template that is to be substituted for the Print common dialog box's template. This element is used only when the PD_ENABLEPRINTTEMPLATE constant is specified in the structure's **Flags** element. The MAKEINTRESOURCE macro can be used if the dialog box resource is numbered.

lpSetupTemplateName
This element is a null-terminated string containing the name of the resource file that has an application-defined dialog box template that is to be substituted for the Print Setup common dialog box's template. This element is used only when the PD_ENABLESETUPTEMPLATE constant is specified in the structure's **Flags** element. The MAKEINTRESOURCE macro can be used if the dialog box resource is numbered.

hPrintTemplate
This element is a handle to a global memory object containing a pre-loaded dialog box template to be used instead of the default Print dialog box template. The value of the **hPrintTemplate element** is used only when the PD_ENABLEPRINTTEMPLATEHANDLE constant is found in the **Flags** element.

hSetupTemplate
This element is a handle to a global memory object containing a pre-loaded dialog box template to be used instead of the default Print Setup dialog box template. The value of the **hSetupTemplate** element is used only when the PD_ENABLESETUPTEMPLATEHANDLE constant is found in the **Flags** element.

C.32.3 Cross-References

CreateDC(), CreateIC(), PrintDlg()

C.33 RECT

C.33.1 Synopsis

typedef struct tagRECT {

int left;

int top;

int right;

int bottom;

} RECT;

C.33.2 Description

The **RECT** structure contains the coordinates of a rectangle's upper-left and lower-right corners.

Element	**Description**
left	This element is the X-coordinate of the rectangle's upper-left corner.
top	This element is the Y-coordinate of the rectangle's upper-left corner.
right	This element is the X-coordinate of the rectangle's lower-right corner.
bottom	This element is the Y-coordinate of the rectangle's lower-right corner.

A rectangle defined by a **RECT** structure cannot have a width that exceeds 32,767 units.

C.33.3 Cross-References

CopyRect(), SetRect(), FillRect(), FrameRect(), InvertRect(), PtInRect()

C.34 RGBQUAD

C.34.1 Synopsis

```
typedef struct tagRGBQUAD {
      BYTE rgbBlue;
      BYTE rgbGreen;
      BYTE rgbRed;
      BYTE rgbReserved;
} RGBQUAD;
```

C.34.2 Description

The **RGBQUAD** structure contains information that describes a color.

Element	**Description**
rgbBlue	This element is the intensity of blue in the color.
rgbGreen	This element is the intensity of green in the color.
rgbRed	This element is the intensity of red in the color.
rgbReserved	This element is unused. It must be assigned the value zero.

C.34.3 Cross-References

BITMAPINFO

C.35 RGBTRIPLE

C.35.1 Synopsis

```
typedef struct tagRGBTRIPLE {
      BYTE rgbtBlue;
      BYTE rgbtGreen;
      BYTE rgbtRed;
} RGBTRIPLE;
```

C.35.2 Description

The **RGBTRIPLE** structure contains information that describes a color.

Element	**Description**
rgbtBlue	This element is the intensity of blue in the color.
rgbtGreen	This element is the intensity of green in the color.
rgbtRed	This element is the intensity of red in the color.

C.35.3 Cross-References

BITMAPCOREINFO, BITMAPINFO, RGBQUAD

C.36 SIZE

C.36.1 Synopsis

```
typedef struct tagSIZE {
      int cx;
      int cy;
} SIZE;
```

C.36.2 Description

The **SIZE** structure contains some function-specific types of size information (for example, viewport extents, window extents, text extents, bitmap dimensions, and aspect-ratio filters).

Element	Description
cx	This element's meaning is specific to the function being used.
cy	This element's meaning is specific to the function being used.

C.36.3 Cross-References

GetAspectRatioFilterEx(), GetBitmapDimensionEx(), GetTextExtentPoint(), GetViewportExtEx(), GetWindowExtEx(), ScaleViewportExtEx(), ScaleWindowExtEx(), SetBitmapDimensionEx(), SetViewportExtEx(), SetWindowExtEx()

C.37 TEXTMETRIC

C.37.1 Synopsis

```
typedef struct tagTEXTMETRIC {
    int tmHeight;
    int tmAscent;
    int tmDescent;
    int tmInternalLeading;
    int tmExternalLeading;
    int tmAveCharWidth;
    int tmMaxCharWidth;
    int tmWeight;
    BYTE tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    int tmOverhang;
    int tmDigitizedAspectX;
    int tmDigitizedAspectY;
} TEXTMETRIC;
```

C.37.2 Description

The **TEXTMETRIC** structure contains information about a physical font.

Element	Description
tmHeight	This element is the character cell's height; the sum the values in the structure's **tmAscent** and **tmDescent** elements.

tmAscent This element is the character cell's ascent; the space between the base line and the top of the character cell.

tmDescent This element is the character cell's descent; the space between the bottom of the character cell and the base line.

tmInternalLeading
This element is the difference between the font's physical size and the font's point size.

If the font is a TrueType font, the value of the **tmInternalLeading** element is equal to the value of **tmHeight** - (**ScaleFactor** * **ntmSizeEM**), where **ScaleFactor** is the font's scaling factor.

If the font is a bitmap font, the value of the **tmInternalLeading** element is used to specify the font's point size. During a request for a logical font, if the LOGFONT structure's **lfHeight** element contains a negative value, the height of the font being requested equals the value of the **tmHeight** element minus the **tmInternalLeading** element.

tmExternalLeading
This element is the amount of extra leading space that the application adds between rows. This area is outside of the character cell. It will therefore contain no marks, and will not be altered by text output calls using either the opaque or transparent modes. A font designer will sometimes set the value of this element to zero.

tmAveCharWidth
This element is the average width of the font's characters. If a font uses the ANSI character set (ANSI_CHARSET), the value of **tmAveCharWidth** is a weighted average width of the characters "a" - "z" and the space character. For fonts that use other character sets, the value of **tmAveCharWidth** is an unweighted average of all characters in the font.

tmMaxCharWidth
This element is the "B" spacing of the font's widest character.

tmWeight This element is the weight of the font. The **tmWeight** element can be assigned one of the following constant values:

FW_DONTCARE	(Use font's default weight)
FW_THIN	
FW_EXTRALIGHT	(Same as FW_ULTRALIGHT)
FW_ULTRALIGHT	(Same as FW_ EXTRALIGHT)
FW_LIGHT	
FW_NORMAL	(Same as FW_ REGULAR)
FW_REGULAR	(Same as FW_ NORMAL)
FW_MEDIUM	
FW_SEMIBOLD	(Same as FW_ DEMIBOLD)
FW_DEMIBOLD	(Same as FW_ SEMIBOLD)
FW_BOLD	
FW_EXTRABOLD	(Same as FW_ ULTRABOLD)
FW_ULTRABOLD	(Same as FW_ EXTRABOLD)
FW_BLACK	(Same as FW_ HEAVY)
FW_HEAVY	(Same as FW_ BLACK)

tmItalic The font is italic. The value of **tmItalic** is TRUE if the font is italic and FALSE if the font is not italic.

tmUnderlined The font is underlined. The value of **tmUnderlined** is TRUE if the font is underlined and FALSE if the font is not underlined.

tmStruckOut The font is struck out. The value of **tmStruckOut** is TRUE if the font is struck out and FALSE if the font is not struck out.

tmFirstChar This element is the value of the font's first character.

tmLastChar This element is the value of the font's last character.

tmDefaultChar This element is the value of the character that is substituted for characters not found in the font.

tmBreakChar This element is the value of the character that is used to define word breaks for text justification.

TmPitchAndFamily

This element is the font's pitch and family.

The value of the four low-order bits of the **tmPitchAndFamily** element specifies the type of font and can be one or more of the following constant values OR'ed together:

TMPF_FIXED_PITCH	This value is the fixed-pitch font.
TMPF_VECTOR	This value is the vector or TrueType font.
TMPF_TRUETYPE	This value is the TrueType font that can be used on a printer and display.
TMPF_DEVICE	This value is the device font, which is set for downloaded and device-resident fonts.
	For example, the TrueType font Courier New® uses the TMPF_FIXED_PITCH, TMPF_VECTOR, and TMPF_TRUETYPE constants.
	The value of the four high-order bits of the tmPitchAndFamily element specifies the font family and can be one of the following constant values:
FF_DECORATIVE	This value specifies a novelty font family, such as Old English.
FF_DONTCARE	This value means the font's family is unimportant or unknown.
FF_MODERN	This value specifies a font with a constant stroke width and with or without serifs (for example, Pica, Elite, or Courier New).
FF_ROMAN	This value specifies a font with a variable stroke width and with serifs (for example, Times New Roman and New Century Schoolbook).
FF_SCRIPT	This value specifies a font that looks like handwriting (for example, Script and Cursive).
FF_SWISS	This value specifies a font with a variable stroke width and without serifs (for example, MS Sans Serif).

tmCharSet This element is the font's character set. The **tmCharSet** element can be assigned one of the following constant values:

ANSI_CHARSET

DEFAULT_CHARSET

SYMBOL_CHARSET

SHIFTJIS_CHARSET

OEM_CHARSET

tmOverhang This element is extra width that is added to some synthesized fonts. The GDI or a device will add width to a string on a per-character and per-string basis when synthesizing such items as bold or italic.

The value of the **tmOverhang** element is zero for many italic and bold TrueType fonts because many TrueType fonts include non-synthesized italic and bold faces.

The value of a Raster font's overhang can be used to determine the amount of spacing between words that have different attributes.

tmDigitizedAspectX
This element is the horizontal aspect of the device for which the font was designed.

tmDigitizedAspectY
This element is the vertical aspect of the device for which the font was designed.

C.37.3 Cross-References

EnumFontFamilies(), EnumFonts(), GetDeviceCaps(), GetTextMetrics(), **NEWTEXTMETRIC**

C.38 WINDOWPLACEMENT

C.38.1 Synopsis

```
typedef struct tagWINDOWPLACEMENT {
    UINT length;
    UINT flags;
    UINT showCmd;
    POINT ptMinPosition;
    POINT ptMaxPosition;
    RECT rcNormalPosition;
} WINDOWPLACEMENT;
```

C.38.2 Description

The **WINDOWPLACEMENT** structure contains information about a window's placement on the screen.

Element	Description
length	This element is the size of the **WINDOWPLACEMENT** structure in bytes.
flags	This element controls the position of the window when minimized and the method by which the window is restored. The value of the **Flags** element can be one or more of the following constant values OR'ed together:

WPF_SETMINPOSITION	Uses the minimized window position specified in the **ptMinPosition** element.
WPF_RESTORETOMAXIMIZED	Maximizes the window the next time that the window is restored. This setting has no impact after the window is restored one time. This constant value can only be used when the SW_SHOWMINIMIZED constant value is set in the **showCmd** element.

showCmd	This element is the current show state of the window. The value of the **showCmd** element can be one of the following constant values:

SW_HIDE	This value hides the window and activates another window.
SW_MINIMIZE	This value minimizes the window and activates the top-level window in the system's list.
SW_RESTORE	This value activates and displays a window. If the window is minimized or maximized, restores it to its original size and position. Same as SW_SHOWNORMAL.
SW_SHOW	This value activates a window and displays it in its current size and position.

SW_SHOWMAXIMIZED	This value activates a window and displays it as a maximized window.
SW_SHOWMINIMIZED	This value activates a window and displays it as an icon.
SW_SHOWMINNOACTIVE	This value displays a window as an icon. The window that is currently active will remain active.
SW_SHOWNA	This value displays a window in its current state. The window that is currently active will remain active.
SW_SHOWNOACTIVATE	This value displays a window in its most recent size and position. The window that is currently active will remain active.
SW_SHOWNORMAL	This value activates and displays a window. If the window is minimized or maximized, restores it to its original size and position. Same as SW_ RESTORE.

ptMinPosition	This element is the position of the window's top-left corner when minimized.
ptMaxPosition	This element is the position of the window's top-left corner when maximized.
rcNormalPosition	This element is the window's coordinates when restored.

C.38.3 Cross-References

POINT, RECT, *ShowWindow(), GetWindowPlacement(), SetWindowPlacement()*

C.39 WINDOWPOS

C.39.1 Synopsis

typedef struct tagWINDOWPOS {

HWND hwnd;

HWND hwndInsertAfter;

int x;

int y;

int cx;

int cy;

UINT flags;

} WINDOWPOS;

C.39.2 Description

The **WINDOWPOS** structure contains information about a window's size and position.

Element	**Description**
hwnd	This element is the handle of the window.
hwndInsertAfter	This element is the handle of the window behind which the window is placed.
x	This element is the X-coordinate for the upper-left hand corner of the window.
y	This element is the Y-coordinate for the upper-left hand corner of the window.
cx	This element is the width of the window.
cy	This element is the height of the window.

flags	This element defines other window attributes. The value of the **flags** element can be one or more of the following constant values OR'ed together:

SWP_DRAWFRAME	This value draws a frame around the window. The window is sent a WM_NCCALCSIZE message. The frame is specified in the window's class description.
SWP_HIDEWINDOW	This value hides the window.
SWP_NOACTIVATE	This value does not activate the window.
SWP_NOMOVE	This value does not move the window. The **x** and **y** elements will not be used.
SWP_NOOWNERZORDER	This value does not change the owner window's Z order position. Same as the SWP_NOREPOSITION constant value.
SWP_NOSIZE	This value does not resize the window. The **cx** and **cy** elements will not be used.
SWP_NOREDRAW	This value does not redraw the window.
SWP_NOREPOSITION	This value does not change the owner window's Z order position. Same as the SWP_NOOWNERZORDER constant value.
SWP_NOZORDER	This value uses current ordering. The **hwndInsertAfter** element will not be used.
SWP_SHOWWINDOW	This value shows the window.

C.39.3 Cross-References

EndDeferWindowPos(), WM_NCCALCSIZE, WM_WINDOWPOSCHANGED, WM_WINDOWPOSCHANGING

C.40 WNDCLASS

C.40.1 Synopsis

typedef struct tagWNDCLASS {

UINT style;

WNDPROC lpfnWndProc;

int cbClsExtra;

int cbWndExtra;

HINSTANCE hInstance;

HICON hIcon;

HCURSOR hCursor;

HBRUSH hbrBackground;

LPCSTR lpszMenuName;

LPCSTR lpszClassName;

} WNDCLASS;

C.40.2 Description

The **WNDCLASS** structure contains information about a window class.

Element	Description
style	This element is the class's styles. The value of the style element can be one or more of the following constant values OR'ed together:

CS_BYTEALIGNCLIENT	A window's client area is aligned on the byte boundary in the x-direction.
CS_BYTEALIGNWINDOW	The window is aligned on the byte boundary in the x-direction. Used when an application uses the *BitBlt()* function.
CS_CLASSDC	The window class has its own display context that is shared among instances.
CS_DBLCLKS	The window receives mouse double-click messages.
CS_GLOBALCLASS	The window class is created by an application or library and is available to all applications. The class is destroyed when the application or library that created it exits. Any windows that use the class should be closed before the class is destroyed.
CS_HREDRAW	The entire window is redrawn when its horizontal size changes.
CS_NOCLOSE	This value disables the System menu's Close menu item.
CS_OWNDC	Each window instance has its own display context.
CS_PARENTDC	This value uses the parent window's display context.
CS_SAVEBITS	The system creates a bitmap of the screen image that is covered by the window. When the window is closed, the system quickly restores the screen image using the bitmap. A window that uses this option will take longer to display. This option is useful when displaying windows that are displayed only briefly and then are removed before other screen operations can take place (for example, menus and dialog boxes).
CS_VREDRAW	The entire window is redrawn when its vertical size changes.

lpfnWndProc This element is a pointer to the window's message handling function.

cbClsExtra This element is the size of a buffer, in bytes, that is allocated and associated with the class. Each window that is created from this class has access to the class buffer. The buffer is initialized with zero when it is allocated. Refer to the *SetClassWord()* and *SetClassLong()* functions.

cbWndExtra This element is the size of a buffer, in bytes, that is allocated each time that a window of the class is created. The buffer is initialized with zero when it is allocated. Refer to the *SetWindowWord()* and *SetWindowLong()* functions. If the **WNDCLASS** structure is used to register a dialog box created with the CLASS resource file keyword, the value of the cbWndExtra element must be set to DLGWINDOWEXTRA.

hInstance This element is the class module. The value of the **hInstance** element must be a valid instance handle and cannot be the value NULL.

hIcon This element is the handle of the window class's icon. The icon is drawn when a window of the class is minimized. If the value of the hIcon element is NULL, the application is responsible for drawing the icon when the window is minimized.

hCursor This element is the handle of the window class's cursor. This cursor shape is shown whenever the mouse is moved into a window of this class. If the value of the **hCursor** element is NULL, the application is responsible for setting the cursor shape whenever the mouse is moved into the window.

hbrBackground This element is the window's background painting. The value of the **hbrBackground** element can either be a handle to a physical brush or a color value that is used to paint the window's background.

If the value of the **hbrBackground** element is a color value, it must be one of the standard system colors listed below with the value 1 added to it (for example, COLOR_MENU + 1):

COLOR_ACTIVEBORDER	COLOR_HIGHLIGHTTEXT

COLOR_ACTIVECAPTION	COLOR_INACTIVEBORDER
COLOR_APPWORKSPACE	COLOR_INACTIVECAPTION
COLOR_BACKGROUND	COLOR_INACTIVECAPTIONTEXT
COLOR_BTNFACE	COLOR_MENU
COLOR_BTNSHADOW	COLOR_MENUTEXT
COLOR_BTNTEXT	COLOR_SCROLLBAR
COLOR_CAPTIONTEXT	COLOR_WINDOW
COLOR_GRAYTEXT	COLOR_WINDOWFRAME
COLOR_HIGHLIGHT	COLOR_WINDOWTEXT

When the class is freed, the brush associated with the **hbrBackground** element will automatically be deleted.

If the value of the **hbrBackground** element is NULL, the application is responsible for painting the window's background. In this case, an application should respond to the WM_ERASEBKGND message and also test the value of the **PAINTSTRUCT** structure's **fErase** element when calling the *BeginPaint()* function.

lpszMenuName This element is a pointer to a null-terminated string that contains the name of the class's menu resource. If an integer is used to identify the menu resource, use the MAKEINTRESOURCE macro.

If the value of the **lpszMenuName** element is NULL, windows of this class will have no default menu.

lpszClassName This element is a pointer to a null-terminated string that contains the name of the window class.

C.40.3 Cross-References

PAINTSTRUCT, MAKEINTRESOURCE, *RegisterClass(), WindowProc(), BitBlt(), SetClassWord(), SetClassWord(), SetClassLong(), SetWindowWord(), SetWindowLong(), BeginPaint()*

Annex D

Window Messages

Description

This Annex describes window messages.

D.1 BM_GETCHECK

D.1.1 Description

The BM_GETCHECK message is sent to a button to get its check state.

The button must be created with the BS_AUTOCHECKBOX, BS_AUTORADIOBUTTON, BS_AUTO3STATE, BS_CHECKBOX, BS_RADIOBUTTON, or BS_3STATE style**.**

Parameter	Description
wParam	Not used.
lParam	Not used.

D.1.2 Returns

The return value specifies the check state of the button and is one of the following values:

0	Unchecked button state.
1	Checked button state.
2	Indeterminate button state (only for a button with the BS_3STATE or BS_AUTO3STATE style).

D.1.3 Cross-References

BM_GETSTATE, BM_SETCHECK, *SendDlgItemMessage()*

D.2 BM_GETSTATE

D.2.1 Description

The BM_GETSTATE message is sent to a button to get information about its current state.

Parameter	Description
wParam	Not used.
lParam	Not used.

D.2.2 Returns

The return value contains all of the state information for the button. To obtain a specific type of state information for the button, use one of the following mask values and the return value:

Mask	Description	
0x0003	The check state (used for radio buttons and check boxes only).	
	Value	**Meaning**
	0	Button is unchecked.
	1	Button is checked.
	2	Indeterminate check state (when a 3-state check box is grayed).
0x0004	The highlight state. When the user presses a button control and holds the left mouse button down, the button control is highlighted. The highlighting is removed when the user releases the left mouse button.	

Value	Meaning
0	Button is not highlighted.
1	Button is highlighted.

0x0008 The focus state. A non-zero value indicates that the button has the focus.

Value	Meaning
0	Button does not have the focus.
1	Button has the focus.

D.2.3 Cross-References

BM_GETCHECK, BM_SETSTATE, *SendDlgItemMessage()*

D.3 BM_SETCHECK

D.3.1 Description

The BM_SETCHECK message is sent to a button to set its current check state.

Parameter	Description
wParam	The button's new check state. The *wParam* parameter can be one of the following values:

Value	Meaning
0	Button should be unchecked.
1	Button should be checked.
2	The button state should be indeterminate (only for the BS_3STATE or BS_AUTO3STATE styles).

lParam	Not used.

D.3.2 Returns

The message's return value is always zero.

D.3.3 Cross-References

BM_GETCHECK, BM_SETSTATE, *SendDlgItemMessage()*

D.4 BM_SETSTATE

D.4.1 Description

The BM_SETSTATE message is sent to a button to set its current highlight state.

Parameter	Description
wParam	The button's new highlight state. The *wParam* parameter can be one of the following values:

Value	Meaning
0	Button should be not highlighted.
1	Button should be highlighted.

lParam	Not used.

D.4.2 Returns

The message's return value is always zero.

D.4.3 Cross-References

BM_GETSTATE, BM_SETCHECK, *SendDlgItemMessage()*

D.5 BM_SETSTYLE

D.5.1 Description

The BM_SETSTYLE message is sent to a button to set its style.

Parameter	Description
wParam	The value of the button's new style. For a list of supported button styles, see “Button Styles” in Annex F.
lParam	The value of the low-order word of the *lParam* parameter specifies if the button should be redrawn. It can be one of the following values:

Value	Meaning
FALSE	Button should not be redrawn.
TRUE	Button should be redrawn.

To retrieve the button's complete button style, call the *GetWindowLong()* function with the GWL_STYLE offset value. The low-word of the complete button style is the button's button-specific style.

D.5.2 Returns

The message's return value is always zero.

D.5.3 Cross-References

GetWindowLong()

D.6 CB_ADDSTRING

D.6.1 Description

The CB_ADDSTRING message is sent to a combo box and used to add a string to the combo box's list box. If the control does not have the CBS_SORT style set, the specified string is added to the end of the list. If the control has the CBS_SORT style set, the specified string is added to the list and the list is then sorted.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	A pointer to the null-terminated string to add to the control. If the control was created with an owner-drawn style and does not have the CBS_HASSTRINGS style set, the value of *lParam* is stored instead of the string to which *lParam* points.

If the control was created with an owner-drawn style and has the CBS_SORT style set but not the CBS_HASSTRINGS style set, a WM_COMPAREITEM message is sent one or more times to the combo box's owner so that the new string can be placed in the correct position in the list.

The CB_INSERTSTRING message can be used to insert a string into a specific location within the combo box list.

D.6.2 Returns

If the insertion is successful, the message returns the string's zero-based position in the combo box's list box. If there is not enough space to store the string, the return value is CB_ERRSPACE. If any other error occurs, the return value is CB_ERR.

D.6.3 Cross-References

CB_INSERTSTRING, WM_COMPAREITEM, CB_DIR

D.7 CB_CURSEL

D.7.1 Description

The CB_CURSEL message is sent to a combo box and used to retrieve the zero-based index position of the currently selected item in the combo box's list box**.**

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.7.2 Returns

If there is an item selected in the combo box's list box, the item's zero-based position in the list box is returned. If there is not an item selected in the combo box's list box, the return value is CB_ERR.

D.7.3 Cross-References

CB_SELECTSTRING, CB_SETCURSEL

D.8 CB_DELETESTRING

D.8.1 Description

The CB_DELETESTRING message is sent to a combo box and is used to delete a string from the combo box's list box.

Parameter	Description
wParam	Specifies the zero-based index of the string to delete.
lParam	Not used. Must be set to zero.

If the control is created with an owner-drawn style but does not have the CBS_HASSTRINGS style set, a WM_DELETEITEM message is sent to the control's owner to inform it that any additional data associated with the item can be freed.

D.8.2 Returns

The number of strings left in the combo box after the deletion. If the deletion fails, the return value is CB_ERR.

D.8.3 Cross-References

WM_DELETEITEM, CB_RESETCONTENT

D.9 CB_DIR

D.9.1 Description

The CB_DIR message is sent to a combo box and used to add a list of filenames to the combo box's list box.

Parameter	Description
wParam	The attributes of the files to be added to the list box. The value can be one or more of the following constant values OR'ed together:

Value	Meaning
DDL_READWRITE	Reading and writing allowed.
DDL_READONLY	Read only file.
DDL_HIDDEN	Hidden file.
DDL_SYSTEM	System file.
DDL_DIRECTORY	Is a directory.
DDL_ARCHIVE	Archived file.
DDL_DRIVES	Includes all drives that match the name specified in the buffer pointed to by the lParam parameter. When this value is used, the DDL_EXCLUSIVE value is automatically used as well.
DDL_EXCLUSIVE	Only lists files of the type specified. If the DDL_EXCLUSIVE value is not used, files of the specified type are listed in addition to files that do not match the specified type.

	To create a directory listing that shows files and drives, the application should send the CB_DIR message to the combo box two times. The first message should use the DDL_DRIVES value to show only the drives. The second message should use the values that are needed for the files.
lParam	A pointer to a null-terminated string that contains the filename to add to the list. If the string contains any wildcards (for example, *.txt), any file that matches the wildcard specification and has the desired file attributes is added to the list.

D.9.3 Returns

If the insertion of the entries was successful, the message returns the zero-based position of the last filename that was inserted into the combo box's list box. If there is not enough space in which to store the strings, the return value is CB_ERRSPACE. If any other error occurs, the return value is CB_ERR.

D.9.4 Cross-References

CB_ADDSTRING, CB_INSERTSTRING, DlgDirList(), DlgDirListComboBox()

D.10 CB_FINDSTRING

D.10.1 Description

The CB_ FINDSTRING message is sent to a combo box and used to search the combo box's list box for an item that begins with the characters in the search string. The search string and a list box string is compared up to the first number of characters in the search string only. Therefore, the target string can contain more characters than the search string. The string comparison is not case-sensitive.

Parameter	**Description**
wParam	The zero-based list box position of the list box item that is before the first list box item to be searched. For example, if the value -1 is specified, the list box is searched from the first item in the list box since that is the zero position in the list box. If the search fails to find a match after processing the last list box item, the search is continued from the top of the list box back to the specified position.
lParam	A pointer to the null-terminated string to search for in the list box.

If the control was created with an owner-drawn style and does not have the CBS_HASSTRINGS style set, the way in which comparisons are made during the search is dependent on whether the CBS_SORT style is used. If the CBS_SORT style is used, a WM_COMPAREITEM message is sent one or more times to the combo box's owner when making string comparisons. Otherwise, the doubleword value of the list box item is compared to the value of the search string.

D.10.2 Returns

If the search is successful, the return value is the zero-based position of a list box. If the search is not successful, the return value is CB_ERR.

D.10.3 Cross-References

CB_FINDSTRINGEXACT, CB_SELECTSTRING, WM_COMPAREITEM

D.11 CB_FINDSTRINGEXACT

D.11.1 Description

The CB_FINDSTRINGEXACT message is sent to a combo box and used to search for a string in the combo box's list box. The target string must contain the same number of characters as the search string for it to be considered a match. The string comparison is not case-sensitive.

Parameter	**Description**
wParam	The zero-based list box position of the list box item that is before the first list box item to be searched. For example, if the value -1 is specified, the list box is searched from the first item in the list box since that is the zero position in the list box. If the search fails to find a match after processing the last list box item, the search is continued from the top of the list box back to the specified position.

lParam	A pointer to the null-terminated string to search for in the list box.

If the control was created with an owner-drawn style and does not have the CBS_HASSTRINGS style set, the way in which comparisons are made during the search is dependent on whether or not the CBS_SORT style is used. If the CBS_SORT style is used, a WM_COMPAREITEM message is sent one or more times to the combo box's owner when making string comparisons. Otherwise, the doubleword value of the list box item is compared to the value of the search string.

D.11.2 Returns

If the search is successful, the return value is the zero-based position of a list box. If the search is not successful, the return value is CB_ERR.

D.11.3 Cross-References

CB_FINDSTRING, CB_SELECTSTRING, WM_COMPAREITEM

D.12 CB_GETCOUNT

D.12.1 Description

The CB_GETCOUNT message is sent to a combo box and retrieves the number of items in the combo box's list box.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.12.2 Returns

Returns the number of items in the list.

D.12.3 Cross-References

None.

D.13 CB_GETDROPPEDCONTROLRECT

D.13.1 Description

The CB_GETDROPPEDCONTROLRECT message is sent to a combo box and used to retrieve the screen coordinates of the combo box's visible (dropped-down) list box.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	A pointer to a RECT structure in which the screen coordinates of the combo box's visible (dropped-down) list box are stored.

D.13.2 Returns

The value CB_OKAY is always returned.

D.13.3 Cross-References

RECT

D.14 CB_GETDROPPEDSTATE

D.14.1 Description

The CB_GETDROPPEDSTATE message is sent to a combo box and is used to determine if the combo box's list box is visible (dropped down) or not.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.14.2 Returns

If the combo box's list box is visible, the value TRUE is returned. If the combo box's list box is not visible, the value FALSE is returned.

D.14.3 Cross-References

CB_SHOWDROPDOWN

D.15 CB_GETEDITSEL

D.15.1 Description

The CB_GETEDITSEL message is sent to a combo box and used to retrieve the starting and ending character positions of the characters selected in the combo box's edit control.

Parameter	Description
wParam	Not used. Must be set to zero
lParam	Not used. Must be set to zero.

D.15.2 Returns

A doubleword value is returned, which contains the starting position in the low-order word and the position of the first non-selected character, after the end of the selection, in the high-order word.

D.15.3 Cross-References

CB_SETEDITSEL

D.16 CB_GETEXTENDEDUI

D.16.1 Description

The CB_GETEXTENDEDUI message is sent to a combo box and used to find out if the combo box has the default user interface or the extended user interface.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.16.2 Returns

If the combo box has the extended user interface, a non-zero value is returned. If the combo box does not have the extended user interface, the zero value is returned.

D.16.3 Cross-References

CB_SETEXTENDEDUI

D.17 CB_GETITEMDATA

D.17 1 Description

The CB_GETITEMDATA message is sent to a combo box and used to retrieve the doubleword value that an application has associated with a combo box's list box item, using the CB_SETITEMDATA message.

Parameter	Description
wParam	The zero-based list box position of the list box item.
lParam	Not used. Must be set to zero.

D.17.2 Returns

If the message is successful, it returns the doubleword value that is associated with the specified list box item. If the message is not successful, it returns the value CB_ERR.

D.17.3 Cross-References

CB_SETITEMDATA

D.18 CB_GETITEMHEIGHT

D.18.1 Description

The CB_GETITEMHEIGHT message is sent to a combo box and used to retrieve the height of a component of the combo box.

Parameter	Description
wParam	The value of *wParam* specifies the desired combo box component whose height is desired.
	If the combo box has the CBS_OWNERDRAWVARIABLE style set, the value of *wParam* can be a list box item's zero-based position.
	If the value of *wParam* is -1, the height of the combo box's edit control is returned.
lParam	Not used. Must be set to zero.

D.18.2 Returns

If the value of *wParam* is -1, the height of the combo box's edit control is returned.

If the value of *wParam* was not -1 and the combo box has the CBS_OWNERDRAWVARIABLE style set, the height of the specified list box item is returned.

If an error occurs, the return value is CB_ERR.

D.18.3 Cross-References

CB_SETITEMHEIGHT, WM_MEASUREITEM

D.19 CB_GETLBTEXT

D.19.1 Description

The CB_GETLBTEXT message is sent to a combo box and used to retrieve one of its list box item's strings.

Parameter	Description
wParam	The zero-based position of the list box item whose string is being retrieved.
lParam	A pointer to a text buffer that is large enough to store the list box item's string and a terminating null character. To determine how large the buffer must be, an application can send the combo box the CB_GETLBTEXTLEN message.

If the combo box is an owner-drawn combo box and uses the CBS_HASSTRINGS style, the doubleword value associated with the list box item is stored in the buffer pointed to by the lParam parameter.

D.19.2 Returns

If the message is processed successfully, the number of bytes that were used in the buffer to store the string (not including the terminating null character) is returned.

If an error occurs, the return value is CB_ERR.

D.19.3 Cross-References

CB_GETLBTEXTLEN

D.20 CB_GETLBTEXTLEN

D.20.1 Description

The CB_GETLBTEXTLEN message is sent to a combo box and used to retrieve the size, in bytes, of one of the strings in the combo box's list box.

Parameter	Description
wParam	The zero-based position of the list box item.
lParam	Not used. Must be set to zero.

D.20.2 Returns

If the message is processed successfully, the size, in bytes, of the list box item's string (not including the terminating null character) is returned.

If an error occurs, the return value is CB_ERR.

D.20.3 Cross-References

CB_GETLBTEXT

D.21 CB_INSERTSTRING

D.21.1 Description

The CB_INSERTSTRING message is sent to a combo box and used to add a string to the combo box's list box. The items in the list box will not be sorted after the insertion; even if the CBS_SORT style is set.

Parameter	**Description**
wParam	The zero-based list box position at which to insert the string. If the value of *wParam* is -1, the string is inserted at the end of the list.
lParam	A pointer to the null-terminated string to add to the control. If the control was created with an owner-drawn style and does not have the CBS_HASSTRINGS style set, the value of *lParam* is stored instead of the string to which it *lParam* points.

The CB_ADDSTRING message can be used to insert a string into the combo box's list box and sort the list after the insertion.

D.21.2 Returns

If the insertion was successful, the message returns the string's zero-based position in the combo box's list box. If there is not enough space in which to store the string, the return value is CB_ERRSPACE. If any other error occurs, the return value is CB_ERR.

D.21.3 Cross-References

CB_ADDSTRING, CB_DIR

D.22 CB_LIMITTEXT

D.22.1 Description

The CB_LIMITTEXT message is sent to a combo box and used to limit the size of the text that can be typed into the combo box's edit control.

Parameter	**Description**
wParam	The number of bytes of text that can be typed into the combo box's edit control. If the value of *wParam* is zero, the size defaults to 65,535 bytes.
lParam	Not used. Must be set to zero.

If the CBS_AUTOHSCROLL style is not set in the combo box, setting the text limit to be larger than the size of the edit control has no effect.

This message only limits the amount of text that can be entered into the edit control by a user. It will have no impact on text that is already in the edit control when the message is processed. If one of the list box's strings is longer than the limit and is selected, the entire string is still shown in the edit control.

D.22.2 Returns

If the message is processed successfully, TRUE is returned.

If the message is sent to a combo box that has the style CBS_DROPDOWNLIST set, the return value is CB_ERR.

D.22.3 Cross-References

CBS_AUTOHSCROLL, CBS_DROPDOWNLIST

D.23 CB_RESETCONTENT

D.23.1 Description

The CB_RESETCONTENT message is sent to a combo box and is used to clear the contents of the combo box's list box and edit control.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

If the control was created with an owner-drawn style but does not have the CBS_HASSTRINGS style set, a WM_DELETEITEM message is sent to the combo box's owner each time that an item is deleted from the combo box's list box.

D.23.2 Returns

CB_OKAY is always returned.

D.23.3 Cross-References

CB_DELETESTRING, WM_DELETEITEM

D.24 CB_SELECTSTRING

D.24.1 Description

The CB_SELECTSTRING message is sent to a combo box and used to search the combo box's list box for an item that begins with the characters in a given search string. If a match is found during the search, the list box item is selected and its text is copied to the combo box's edit control.

The search string and a list box string is only compared up to the first number of characters in the search string. Therefore, the target string can contain more characters than the search string. The string comparison is not case-sensitive.

Parameter	Description
wParam	The zero-based list box position of the list box item that is before the first list box item to be searched. For example, if the value -1 is specified, the list box is searched from the first item in the list box since that is the zero position in the list box. If the search fails to find a match after processing the last list box item, the search is continued from the top of the list box back to the specified position.
lParam	A pointer to the null-terminated string to search for in the list box.

If the control was created with an owner-drawn style and does not have the CBS_HASSTRINGS style set, the way in which comparisons are made during the search is dependent on whether or not the CBS_SORT style is used. If the CBS_SORT style is used, a WM_COMPAREITEM message is sent one or more times to the combo box's owner when making string comparisons. Otherwise, the doubleword value of the list box item is compared to the value of the search string.

D.24.2 Returns

If the search string was found, the return value is the zero-based position of the selected list box item. If the search string was not found, the return value is CB_ERR and the current selection is not changed.

D.24.3 Cross-References

CB_FINDSTRING, CB_FINDSTRINGEXACT, CB_SETCURSEL, WM_COMPAREITEM

D.25 CB_SETCURSEL

D.25.1 Description

The CB_SETCURSEL message is sent to a combo box and used to select an item in the combo box's list box. If the specified list box item is not visible, the list box is scrolled until the item is visible. The selected item's string is copied into the edit control. Any previously selected item is unselected.

Parameter	**Description**
wParam	The zero-based list box position of the list box item to select. If the value of *wParam* is -1, any previous list box selections is cleared and no new selections are made.
lParam	Not used. Must be set to zero.

D.25.2 Returns

If successful, the selected list box item's position is returned. If an error occurs or if the value of *wParam* was -1, the return value is CB_ERR.

D.25.3 Cross-References

CB_GETCURSEL, CB_SELECTSTRING

D.26 CB_SETEDITSEL

D.26.1 Description

An application sends a CB_SETEDITSEL message to select characters in the edit control of a combo box.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	The low-order word of *lParam* specifies the starting position. If this parameter is set to -1, the selection, if any, is removed. The high-order word of *lParam* specifies the ending position. If this parameter is set to -1, all text from the starting position to the last character in the edit control is selected.

The positions in the edit control are zero-based, meaning that to select the first character of the edit control one would specify a starting position of zero, the ending position is the character just after to select.

D.26.2 Returns

The return value is non-zero if the message is successful. It is CB_ERR if the message is sent to a combo box with the CBS_DROPDOWNLIST style.

D.26.3 Cross-References

CB_GETEDITSEL

D.27 CB_SETEXTENDEDUI

D.27.1 Description

An application sends a CB_SETEXTENDEDUI message to select either the default user interface or the extended user interface for a combo box that has the CBS_DROPDOWN or CBS_DROPDOWNLIST style.

Parameter	**Description**
wParam	Specifies whether the combo box should use the extended user interface or the default user interface. A value of TRUE selects the extended user interface, a value of FALSE selects the standard user interface.
lParam	Not used. Must be set to zero.

The extended user interface is different from the default interface in the following ways:

Clicking the static control displays the list box (CBS_DROPDOWNLIST style only). Pressing the DOWN ARROW key displays the list box (F4 is disabled).

Scrolling in the static control is disabled when the item list is not visible (the arrow keys are disabled).

D.27.2 Returns

The return value is CB_OKAY if the operation is successful, or it is CB_ERR if an error occurred.

D.27.3 Cross-References

CB_GETEXTENDEDUI

D.28 CB_SETITEMDATA

D.28.1 Description

An application sends a CB_SETITEMDATA message to set the doubleword value associated with the specified item in a combo box.

Parameter	**Description**
wParam	Specifies the zero-based index to the item.
lParam	Specifies the new value to be associated with the item.

D.28.2 Returns

The return value is CB_ERR if an error occurs.

D.28.3 Cross-References

CB_GETITEMDATA

D.29 CB_SETITEMHEIGHT

D.29.1 Description

An application sends a CB_SETITEMHEIGHT message to set the height of list items in a combo box or the height of the edit-control (or static-text) portion of a combo box.

Parameter	**Description**
wParam	Specifies whether the height of list items or the height of the edit control (or static-text) portion of the combo box is set.
lParam	The low-order word of *lParam* specifies the height, in pixels, of the combo box component identified by *wParam*.

If the combo box has the CBS_OWNERDRAWVARIABLE style, the *wParam* parameter specifies the zero-based index of the list item whose height is to be set. Otherwise, the *wParam* parameter must be zero and the height of all list items is set. If *wParam* is -1, the height of the edit control or static-text portion of the combo box is to be set.

The height of the edit control (or static-text) portion of the combo box is set independently of the height of the list items. Therefore an application must ensure that the height of the edit control (or static-text) portion is not smaller than the height of a particular list box item.

D.29.2 Returns

The return value is CB_ERR if the index or height is invalid.

D.29.3 Cross-References

CB_GETITEMHEIGHT, WM_MEASUREITEM

D.30 CB_SHOWDROPDOWN

D.30.1 Description

An application sends a CB_SHOWDROPDOWN message to show or hide the list box of a combo box that has the CBS_DROPDOWN or CBS_DROPDOWNLIST style.

Parameter	**Description**
wParam	Specifies whether the drop-down list box is to be shown or hidden. A value of TRUE shows the list box, a value of FALSE hides it.
lParam	Not used. Must be set to zero.

This message has no effect on a combo box created with the CBS_SIMPLE style.

D.30.2 Returns

The return value is always non-zero.

D.30.3 Cross-References

CB_GETDROPPEDSTATE

D.31 DM_GETDEFID

D.31.1 Description

An application sends a DM_GETDEFID message to get the identifier of the default push button for a dialog box.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.31.2 Returns

The return value is a doubleword value. If the default push button has an identifier value, the high-order word contains DC_HASDEFID and the low-order word contains the identifier value. The return value is zero if the default push button does not have an identifier value.

D.31.3 Cross-References

DM_SETDEFID

D.32 DM_SETDEFID

D.32.1 Description

An application sends a DM_SETDEFID message to change the identifier of the default push button for a dialog box.

Parameter	**Description**
wParam	Specifies the identifier of the push button that will become the new default.
lParam	Not used. Must be set to zero.

D.32.2 Returns

The return value is always non-zero.

D.32.3 Cross-References

DM_GETDEFID

D.33 EM_CANUNDO

D.33.1 Description

An application sends an EM_CANUNDO message to determine whether an edit-control operation can be undone.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.33.2 Returns

The return value is non-zero if the last edit operation can be undone, or it is zero if the last edit operation cannot be undone.

D.33.3 Cross-References

EM_EMPTYUNDOBUFFER, EM_UNDO

D.34 EM_EMPTYUNDOBUFFER

D.34.1 Description

An application sends an EM_EMPTYUNDOBUFFER message to reset (clear) the undo flag of an edit control. The undo flag is set whenever an operation within the edit control can be undone.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

The undo flag is automatically cleared whenever the edit control receives a WM_SETTEXT or EM_SETHANDLE message.

D.34.2 Returns

This message does not return a value.

D.34.3 Cross-References

EM_CANUNDO, EM_UNDO

D.35 EM_FMTLINES

D.35.1 Description

An application sends an EM_FMTLINES message to set the inclusion of soft line break characters on or off within a multiline edit control. A soft line break consists of two carriage returns and a linefeed inserted at the end of a line that is broken because of wordwrapping. This message is processed only by multiline edit controls.

Parameter	**Description**
wParam	Specifies whether soft line break characters are to be inserted. A value of TRUE inserts the characters. A value of FALSE removes them.
lParam	Not used. Must be set to zero.

This message affects only the buffer returned by the EM_GETHANDLE message and the text returned by the WM_GETTEXT message. It has no effect on the display of the text within the edit control. A line that ends with a hard line break is not affected by the EM_FMTLINES message. A hard line break consists of one carriage return and a linefeed.

D.35.2 Returns

The return value is identical to the wParam parameter.

D.35.3 Cross-References

EM_GETWORDBREAKPROC, EM_SETWORDBREAKPROC

D.36 EM_GETFIRSTVISIBLELINE

D.36.1 Description

An application sends an EM_GETFIRSTVISIBLELINE message to determine the topmost visible line in an edit control.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.36.2 Returns

The return value is the zero-based index of the topmost visible line. For single-line edit controls, the return value is zero.

D.36.3 Cross-References

None.

D.37 EM_GETHANDLE

D.37.1 Description

An application sends an EM_GETHANDLE message to retrieve a handle to the memory currently allocated for a multiline edit control. The handle is a local memory handle and can be used by any of the functions that take a local memory handle as a parameter. This message is processed only by multiline edit controls.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

An application can send this message to a multiline edit control in a dialog box only if it created the dialog box with the DS_LOCALEDIT style flag set. If the DS_LOCALEDIT style is not set, the return value is still non-zero, but the return value will not be meaningful.

D.37.2 Returns

The return value is a local memory handle identifying the buffer that holds the contents of the edit control. If an error occurs, such as sending the message to a single-line edit control, the return value is zero.

D.37.3 Cross-References

EM_SETHANDLE

D.38 EM_GETLINE

D.38.1 Description

An application sends an EM_GETLINE message to retrieve a line of text from an edit control.

Parameter	Description
wParam	Specifies the line number of the line to retrieve from a multiline edit control. Line numbers are zero-based; a value of zero specifies the first line. This parameter is ignored by a single-line edit control.
lParam	Points to the buffer that receives a copy of the line. The first word of the buffer specifies the maximum number of bytes that can be copied to the buffer.

The copied line does not contain a terminating null character.

D.38.2 Returns

The return value is the number of bytes actually copied. The return value is zero if the line number specified by the line parameter is greater than the number of lines in the edit control.

D.38.3 Cross-References

EM_LINELENGTH, WM_GETTEXT

D.39 EM_GETLINECOUNT

D.39.1 Description

An application sends an EM_GETLINECOUNT message to retrieve the number of lines in a multiline edit control. This message is processed only by multiline edit controls.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.39.2 Returns

The return value is an integer containing the number of lines in the multiline edit control. If no text is in the edit control, the return value is 1.

D.39.3 Cross-References

None.

D.40 EM_GETMODIFY

D.40.1 Description

An application sends an EM_GETMODIFY message to determine whether the contents of an edit control have been modified.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

Windows maintains an internal flag indicating whether the contents of the edit control have been changed. This flag is cleared when the edit control is first created, or an EM_SETMODIFY message can be sent to clear the flag.

D.40.2 Returns

The return value is non-zero if the edit control contents have been modified, or the value is zero if the contents remain unchanged.

D.40.3 Cross-References

EM_SETMODIFY

D.41 EM_GETPASSWORDCHAR

D.41.1 Description

An application sends an EM_GETPASSWORDCHAR message to retrieve the password character displayed in an edit control when the user enters text.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

If the edit control is created with the ES_PASSWORD style, the default password character is set to an asterisk (*).

D.41.2 Returns

The return value specifies the character to be displayed in place of the character typed by the user. The return value is NULL if no password character exists.

D.41.3 Cross-References

EM_SETPASSWORDCHAR

D.42 EM_GETRECT

D.42.1 Description

An application sends an EM_GETRECT message to retrieve the formatting rectangle of an edit control. The formatting rectangle is the limiting rectangle of the text. The limiting rectangle is independent of the size of the edit control window.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Points to the **RECT** structure that receives the formatting rectangle.

The formatting rectangle of a multiline edit control can be modified by the EM_SETRECT and EM_SETRECTNP messages.

D.42.2 Returns

The return value is not a meaningful value.

D.42.3 Cross-References

EM_SETRECT, EM_SETRECTNP, RECT

D.43 EM_GETSEL

D.43.1 Description

An application sends an EM_GETSEL message to get the starting and ending character positions of the current selection in an edit control.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.43.2 Returns

The return value is a doubleword value that contains the starting position in the low-order word and the position of the first nonselected character after the end of the selection in the high-order word.

D.43.3 Cross-References

EM_REPLACESEL, EM_SETSEL

D.44 EM_GETWORDBREAKPROC

D.44.1 Description

An application sends the EM_GETWORDBREAKPROC message to an edit control to retrieve the current wordwrap function.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

A wordwrap function scans a text buffer (which contains text to be sent to the display), looking for the first word that does not fit on the current display line. The wordwrap function places this word at the beginning of the next line on the display. A wordwrap function defines the point at which Windows should break a line of text for multiline edit controls, usually at a space character that separates two words.

D.44.2 Returns

The return value specifies the procedure-instance address of the application-defined wordwrap function. The return value is NULL if no wordwrap function exists.

D.44.3 Cross-References

EM_FMTLINES, EM_SETWORDBREAKPROC, MakeProcInstance(), WordBreakProc()

D.45 EM_LIMITTEXT

D.45.1 Description

An application sends an EM_LIMITTEXT message to limit the length of the text the user can enter into an edit control.

Parameter	Description
wParam	Specifies the length, in bytes, of the text the user can enter. If this parameter is zero, the text length is set to 65,535 bytes.
lParam	Not used. Must be set to zero.

The EM_LIMITTEXT message limits the text the user can enter. It has no effect on any text already in the edit control when the message is sent, nor does it affect the length of text copied to the edit control by the WM_SETTEXT message. If an application uses the WM_SETTEXT message to place more text into an edit control than is specified in the EM_LIMITTEXT message, the user can edit the entire contents of the edit control.

D.45.2 Returns

None.

D.45.3 Cross-References

None.

D.46 EM_LINEFROMCHAR

D.46.1 Description

An application sends an EM_LINEFROMCHAR message to retrieve the line number of the line that contains the specified character index. A character index is the number of characters from the beginning of the edit control. This message is processed only by multiline edit controls.

Parameter	**Description**
wParam	Specifies the character index of the character contained in the line whose number is to be retrieved. If the value of the *wParam* parameter is -1, either the line number of the current line (the line containing the caret) is retrieved or, if there is a selection, the line number of the line containing the beginning of the selection is retrieved.
lParam	Not used. Must be set to zero.

D.46.2 Returns

The return value is the zero-based line number of the line containing the character index specified by the *wParam* parameter.

D.46.3 Cross-References

EM_LINEINDEX

D.47 EM_LINEINDEX

D.47.1 Description

An application sends an EM_LINEINDEX message to retrieve the character index of a line within a multiline edit control. The character index is the number of characters from the beginning of the edit control to the specified line. This message is processed only by multiline edit controls.

Parameter	**Description**
wParam	Specifies the zero-based line number. A value of -1 specifies the current line number (the line that contains the caret).
lParam	Not used. Must be set to zero.

D.47.2 Returns

The return value is the character index of the line specified in the line parameter, or it is -1 if the specified line number is greater than the number of lines in the edit control.

D.47.3 Cross-References

EM_LINEFROMCHAR

D.48 EM_LINELENGTH

D.48.1 Description

An application sends an EM_LINEINDEX message to retrieve the character index of a line within a multiline edit control. The character index is the number of characters from the beginning of the edit control to the specified line. This message is processed only by multiline edit controls.

Parameter	**Description**
wParam	Specifies the character index of a character in the line whose length is to be retrieved when EM_LINELENGTH is sent to a multiline edit control. If this parameter is -1, the message returns the number of unselected characters on lines containing selected characters. For

	example, if the selection extended from the fourth character of one line through the eighth character from the end of the next line, the return value would be 10 (three characters on the first line and seven on the next).
lParam	Not used. Must be set to zero.

When EM_LINELENGTH is sent to a single-line edit control, the *wParam* parameter is ignored.

Use the EM_LINEINDEX message to retrieve a character index for a given line number within a multiline edit control.

D.48.2 Returns

The return value is the length, in bytes, of the line specified by the *wParam* parameter when an EM_LINELENGTH message is sent to a multiline edit control. The return value is the length, in bytes, of the text in the edit control when an EM_LINELENGTH message is sent to a single-line edit control.

D.48.3 Cross-References

EM_GETLINE

D.49 EM_LINESCROLL

D.49.1 Description

An application sends an EM_LINESCROLL message to scroll the text of a multiline edit control. This message is processed only by multiline edit controls.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	The low-order word of *lParam*. Specifies the number of lines to scroll vertically. The high-order word of *lParam* specifies the number of character positions to scroll horizontally. This value is ignored if the edit control has either the ES_RIGHT or ES_CENTER style.

The edit control does not scroll vertically past the last line of text in the edit control. If the current line plus the number of lines specified by the low-order word of *lParam* parameter exceeds the total number of lines in the edit control, the value is adjusted so that the last line of the edit control is scrolled to the top of the edit-control window.

The EM_LINESCROLL message can be used to scroll horizontally past the last character of any line.

D.49.2 Returns

The return value is non-zero if the message is sent to a multiline edit control, or it is zero if the message is sent to a single-line edit control.

D.49.3 Cross-References

None.

D.50 EM_REPLACESEL

D.50.1 Description

An application sends an EM_REPLACESEL message to replace the current selection in an edit control with the text specified by the value of the lParam parameter.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Points to a null-terminated string containing the replacement text.

Use the EM_REPLACESEL message when you want to replace only a portion of the text in an edit control. If you want to replace all of the text, use the WM_SETTEXT message. If there is no current selection, the replacement text is inserted at the current cursor location.

D.50.2 Returns

This message does not return a value.

D.50.3 Cross-References

EM_GETSEL, EM_SETSEL

D.51 EM_SETHANDLE

D.51.1 Description

The EM_SETHANDLE message is processed by multiline edit controls only, and is used to set a handle to local memory that is used by the control.

Parameter	**Description**
wParam	Handle to local memory to be used by the multiline edit control.
lParam	Not used. Must be 0L.

The handle to be used must be created using *LocalAlloc()* with the LMEM_MOVEABLE flag set. The memory allocated must then contain a null-terminated string.

Because there may be a previous handle used by the multiline edit control before setting the new handle the old handle should be freed. This can be done by sending an EM_GETHANDLE message to the control and then freeing the returned handle via *LocalFree()*.

Sending the EM_SETHANDLE message to a multiline edit control clears the undo buffer and clears the EM_CANUNDO and EM_GETMODIFY flags. The control is also redrawn.

Note that multiline edit controls in dialog boxes only respond to this message if the dialog was created with the DS_LOCALEDIT flag set.

D.51.1 Returns

None.

D.51.2 Cross-References

EM_GETHANDLE, *LocalAlloc(), LocalFree()*

D.52 EM_SETMODIFY

D.52.1 Description

The EM_SETMODIFY message sets the modification status of an edit control. This status indicates whether the control has been modified.

Parameter	**Description**
wParam	The new modification status of either TRUE or FALSE.
lParam	Not used. Must be 0L.

This flag is automatically set whenever the user of the application makes a change.

D.52.2 Returns

None.

D.52.3 Cross-References

EM_GETMODIFY

D.53 EM_SETPASSWORDCHAR

D.53.1 Description

The EM_SETPASSWORDCHAR message sets the character to be used for display instead of the actual characters typed by the user.

Parameter	**Description**
wParam	New character.
lParam	Not used. Must be 0L.

Upon processing this message the edit control, which cannot be multiline, redisplays the contents using the new character. If the character is null, the actual characters are displayed.

D.53.2 Returns

If the message was sent to an edit control, the return value is non-zero.

D.53.3 Cross-References

EM_GETPASSWORDCHAR

D.54 EM_SETREADONLY

D.54.1 Description

The EM_SETREADONLY messages set a flag that indicates whether the user may modify an edit control.

Parameter	**Description**
wParam	New status of either TRUE or FALSE.
lParam	Not used. Must be 0L.

D.54.2 Returns

A non-zero value indicates that the process was successful and zero indicates that an error occurred.

D.54.3 Cross-References

None.

D.55 EM_SETRECT

D.55.1 Description

The EM_SETRECT and EM_SETRECTNP messages modify the formatting rectangle of a multiline edit control.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Pointer to a **RECT** structure.

This message is processed by multiline edit controls only. The difference between EM_SETRECT and EM_SETRECTNP is that upon processing the message, the edit control will redraw itself if the message was EM_SETRECT. The EM_SETRECTNP message will not cause a redraw. The original formatting rectangle is the client area of the control. This message can change the formatting rectangle to be smaller or larger. If the formatting rectangle is larger and the control does not have scroll bars, the excess is clipped instead of wrapped.

If the edit control has borders, the formatting rectangle is reduced by the size of the border.

D.55.2 Returns

None.

D.55.3 Cross-References

EM_GETRECT, EM_SETRECTNP, RECT

D.56 EM_SETRECTNP

D.56.1 Description

The EM_SETRECT and EM_SETRECTNP messages modify the formatting rectangle of a multiline edit control.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Pointer to a **RECT** structure.

This message is processed by multiline edit controls only. The difference between EM_SETRECT and EM_SETRECTNP is that when processing the message, the edit control redraws itself if the message was EM_SETRECT. If the message is EM_SETRECTNP, no redraw occurs. The original formatting rectangle is the client area of the control.

This message can change the formatting rectangle so it is either smaller or larger. If the formatting rectangle is larger and the control has scroll bars, the excess is clipped instead of wrapped. If the edit control has borders, the formatting rectangle is reduced by the size of the border.

D.56.2 Returns

None.

D.56.3 Cross-References

EM_GETRECT, EM_SETRECT, **RECT**

D.57 EM_SETSEL

D.57.1 Description

The EM_SETSEL message sets the selected text within an edit control.

Parameter	**Description**
wParam	If set to zero, the caret is scrolled into view. If set to one**,** the caret is not scrolled into view.
lParam	The low-order word indicates the position of the first character and the high-order word indicates the position of the last character.

If the starting position is zero and the ending position is -1, all of the text in the edit control is selected. If the starting position is -1, the current selection is removed. The caret is placed at the end of the selection, which is indicated by the greater positional value of the starting and ending positions.

D.57.2 Returns

Returns are non-zero if the message is sent to an edit control.

D.57.3 Cross-References

EM_GETSEL, EM_REPLACESEL

D.58 EM_SETTABSTOPS

D.58.1 Description

The EM_SETTABSTOPS message resets the tabs for a multiline edit control.

Parameter	**Description**
wParam	The number of tab stops.
lParam	Pointer to an array of integers containing tab stop values.

If the number of tab stops is zero, a default tab value of every 32 dialog units is used. If the number of tab stops is 1, tab stops are set to every n dialog units, where n is an integer pointed to by the *lParam* pointer. If the number of tabstops is greater than or equal to 2, the tab stops are set, in dialog units, according to the integer array pointed to by the *lParam* pointer. Note that this message does not alter the display of the control. In order to update the display to the new tab stops, *InvalidateRect()* should be called.

D.58.2 Returns

If tabs are set, the return value is non-zero. Otherwise it is zero.

D.58.3 Cross-References

InvalidateRect()

D.59 EM_SETWORDBREAKPROC

D.59.1 Description

The EM_SETWORDBREAKPROC message allows the application to replace the default word break function.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Pointer to a user-defined callback function of the type EDITWORDBREAKPROC.

This is a form of subclassing in which the word break processes are subclassed.

D.59.2 Returns

None

D.59.3 Cross-References

EM_GETWORDBREAKPROC, EM_FMTLINES, GETWORDBREAKPROC, MAKEPROCINSTANCE, *WordBreakProc()*

D.60 EM_UNDO

D.60.1 Description

The EM_UNDO message allows the application to undo the last change made to the control.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be 0L.

D.60.2 Returns

A multiline edit control returns a non-zero value if the operation was successful. Otherwise it returns zero. A single line edit control always returns a non-zero value.

D.60.3 Cross-References

EM_CANUNDO

D.61 LB_ADDSTRING

D.61.1 Description

The LB_ADDSTRING message adds a string to the list box control.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Points to the string to be added.

If the list box was created with CBS_SORT, the string is added to the appropriate place in the list. Otherwise, it is added to the end of the list. If the list box was created without LBS_HASSTRINGS, lParam is assumed to be a value rather than a pointer. If the list box is owner-drawn and is created with LBS_SORT and without LBS_HASTRINGS, the application has to be able to process one or more WM_COMPAREITEM messages.

D.61.2 Returns

A value greater than or equal to zero is the index position where the string was inserted. LB_ERRSPACE is returned if there was not enough memory available to add the string. LB_ERR is returned if any other error occurred.

D.61.3 Cross-References

LB_DELETESTRING, LB_INSERTSTRING, WM_COMPAREITEM

D.62 LB_DELETESTRING

D.62.1 Description

The LB_DELETESTRING message deletes a string to the list box control.

Parameter	Description
wParam	Index of the string to be deleted.
lParam	Not used. Must be 0L.

If the list box is owner-drawn but was not created with LBS_HASSTRINGS, a WM_DELETEITEM is sent to the owner so any data associated with the item can also be deleted at the same time.

D.62.2 Returns

The number of strings left in the list box is returned, or LB_ERR is returned, if an error occurred.

D.62.3 Cross-References

LB_INSERTSTRING, WM_DELETEITEM

D.63 LB_DIR

D.63.1 Description

The LB_DIR message adds a directory listing to a list box according to the parameters that are passed.

Parameter	Description
wParam	File attributes.

Value	Description
0x0000	File is read/write.
0x0001	File is read only.
0x0002	File is hidden.
0x0004	File is a system file.
0x0010	The lParam points to a directory name.
0x0020	The file is archived.
0x4000	All drives that match the name specified by the lParam are included.
0x8000	If the files with the specified attribute are exclusively displayed.

lParam	Pointer to a null-terminated string that specifies a file filter including wild card values as required.

D.63.2 Returns

A value greater than or equal to zero is the number of items in the list. LB_ERRSPACE is returned if there was not enough memory available to add the string. LB_ERR is returned if any other error occurred.

D.63.3 Cross-References

DlgDirList(), LB_ADDSTRING, LB_INSERTSTRING

D.64 LB_FINDSTRING

D.64.1 Description

The LB_FINDSTRING message searches the list for a matching entry and returns its index.

Parameter	Description
wParam	Index to begin the search.
lParam	Pointer to a null-terminated string that is to be located.

The search performed is non-case-sensitive and begins with the index entry specified in the *wParam*. If the search is unsuccessful by the time the end of the list is reached, the search is continued from the beginning. If the index to begin searching is -1, the entire list is searched starting at index zero. If the list box is owner-drawn and is created without LBS_HASTRINGS, the action taken depends upon whether the list box was created with LBS_SORT. If the list box is sorted, the owner is sent a WM_COMPAREITEM message. Otherwise, *lParam* is taken as a value and is directly compared to the values associated with each list box entry.

Note: If the string being searched for was "Abc," an entry of "ABCDEF" would be considered a match.

D.64.2 Returns

The index of the matching string is returned, or LB_ERR if the search failed or an error occurred.

D.64.3 Cross-References

LB_ADDSTRING, LB_INSERTSTRING, LB_FINDSTRINGEXACT

D.65 LB_FINDSTRINGEXACT

D.65.1 Description

The LB_FINDSTRINGEXACT message searches the list for a matching entry and returns its index.

Parameter	Description
wParam	Index to begin the search.
lParam	Pointer to a null-terminated string that is to be located.

The search performed is non-case-sensitive and begins with the index entry specified in wParam. If the search is unsuccessful by the time the end of the list is reached, the search is continued from the beginning. If the index to begin searching is -1, the entire list is searched starting at index zero. If the list box is owner-drawn and was created without LBS_HASTRINGS, the action taken depends on whether the list box is created with LBS_SORT. If the list box is sorted, the owner is sent a WM_COMPAREITEM message. Otherwise, the *lParam* is taken as a value and is directly compared to the values associated with each list box entry.

This message differs from LB_FINDSTRING in that the lengths of the strings must be similar.

D.65.2 Returns

The index of the matching string is returned or LB_ERR if the search failed or an error occurred.

D.65.3 Cross-References

LB_ADDSTRING, LB_INSERTSTRING, LB_FINDSTRING

D.66 LB_GETCARETINDEX

D.66.1 Description

The LB_GETCARETINDEX message finds the list item that currently has the focus, regardless of whether it is selected.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

This message is used for multiselection list boxes.

D.66.2 Returns

The index of the item with focus is returned or LB_ERR if an error occurred.

D.66.3 Cross-References

LB_SETCARETINDEX

D.67 LB_GETCOUNT

D.67.1 Description

The LB_GETCOUNT message finds the number of items currently in the list.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.67.2 Returns

The number of items in the list or LB_ERR if an error occurred.

D.67.3 Cross-References

None.

D.68 LB_GETCURSEL

D.68.1 Description

The LB_GETCURSEL message finds the index of the currently selected item in the list.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

This message is used for single selection list boxes.

D.68.2 Returns

The index of the selected item is returned, or LB_ERR if no item is selected or if an error occurred.

D.68.3 Cross-References

LB_SETCURSEL

D.69 LB_GETHORIZONTALEXTENT

D.69.1 Description

The LB_GETHORIZONTALEXTENT message finds the number of pixels that can be horizontally scrolled within the list box, if the list box has a horizontal scroll bar.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be 0L.

This message is used for list boxes created with the WS_HSCROLL style.

D.69.2 Returns

The width in pixels, is returned, or LB_ERR, if an error occurred.

D.69.3 Cross-References

LB_SETHORIZONTALEXTENT

D.70 LB_GETITEMDATA

D.70.1 Description

The LB_GETITEMDATA message retrieves data that the application has associated with a given item in the list.

Parameter	Description
wParam	The index of the item of the data to be retrieved.
lParam	Not used. Must be 0L.

The data retrieved is the *lParam* value that was passed when sending an LB_SETITEMDATA message to the list box.

D.70.2 Returns

The data retrieved is the *lParam* value that was passed when sending an LB_SETITEMDATA message to the list box, or LB_ERR if an error occurred.

D.70.3 Cross-References

LB_SETITEMDATA

D.71 LB_GETITEMHEIGHT

D.71.1 Description

The LB_GETITEMHEIGHT message retrieves the height of an item in a list box.

Parameter	Description
wParam	The index of the item for which the height is to be retrieved.
lParam	Not used. Must be 0L.

If the list box has the style LBS_OWNERDRAW, only then should an index be passed in the *wParam* parameter. Otherwise, *wParam* should be zero.

D.71.2 Returns

The height in pixels of the list item is returned, or LB_ERR, if an error occurred.

D.71.3 Cross-References

LB_SETITEMHEIGHT

D.72 LB_GETITEMRECT

D.72.1 Description

The LB_GETITEMRECT message retrieves the display rectangle of an item within the list box.

Parameter	Description
wParam	The index of the item for which the display rectangle is to be retrieved.
lParam	Pointer to a **RECT** structure.

The rectangle is in client coordinates.

D.72.2 Returns

LB_ERR if an error occurred.

D.72.3 Cross-References

LB_GETITEMHEIGHT, LB_SETITEMHEIGHT, WM_MEASUREITEM, RECT

D.73 LB_GETSEL

D.73.1 Description

The LB_GETSEL message retrieves the selection status of an item in a list box.

Parameter	**Description**
wParam	The index of the item for which the selection status is to be retrieved.
lParam	Not used. Must be 0L.

D.73.2 Returns

TRUE if the item is selected, FALSE if it is not selected, or LB_ERR if an error occurred.

D.73.3 Cross-References

LB_SETSEL, LB_GETCURSEL, LB_SELECTSTRING, LB_SETITEMRANGE

D.74 LB_GETSELCOUNT

D.74.1 Description

The LB_GETSELCOUNT message retrieves the number of items selected in a list box.

Parameter	**Description**
wParam	Not used, and must be 0.
lParam	Not used. Must be 0L.

This message is for use with a multiselection list box.

D.74.2 Returns

The number of selected items, or LB_ERR if an error occurred.

D.74.3 Cross-References

LB_SETSEL, LB_GETSELITEMS

D.75 LB_GETSELITEMS

D.75.1 Description

The LB_GETSELITEMS message gets the index values for all selected items.

Parameter	**Description**
wParam	The maximum number of items that can be retrieved.
lParam	Pointer to an integer array to hold item index values for selected items.

This message should be used with a multiselection list box.

D.75.2 Returns

The actual number of items that was placed in the array, or LB_ERR if an error occurred.

D.75.3 Cross-References

LB_SETSEL, LB_GETSELCOUNT

D.76 LB_GETTEXT

D.76.1 Description

An application sends an LB_GETTEXT message to retrieve a string from a list box.

Parameter	**Description**
wParam	Specifies the index of the string to retrieve in the list box.
lParam	A pointer (LPCSTR) to the buffer to receive the string.

The buffer *lParam* must be large enough for the entire string and its terminating character. Use the LB_GETTEXTLEN message prior to the LB_GETTEXT message to retrieve the length of the string.

D.76.2 Returns

The message returns the length of the string in bytes if successful. If an invalid index was specified, LB_ERR is returned.

D.76.3 Cross-References

LB_GETTEXTLEN

D.77 LB_GETTEXTLEN

D.77.1 Description

An application sends an LB_GETTEXTLEN message to retrieve the length of a string from a list box.

Parameter	**Description**
wParam	Specifies the index of the string in the list box whose length is to be retrieved.
lParam	Not used. Must be set to zero.

D.77.2 Returns

The message returns the length of the string in bytes, if successful. If an invalid index was specified, LB_ERR is returned.

D.77.3 Cross-References

LB_GETTEXT

D.78 LB_GETTOPINDEX

D.78.1 Description

An application sends an LB_GETTOPINDEX message to retrieve the index of the first visible item in a list box.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

The first item in a list box is initially zero, but if the list box is scrolled, another item can be at the top of the list box.

D.78.2 Returns

The message returns the index of the first visible item in the list box.

D.78.3 Cross-References

LB_SETOPINDEX

D.79 LB_INSERTSTRING

D.79.1 Description

An application sends an LB_INSERTSTRING message to insert a string into a list box.

Parameter	**Description**
wParam	Specifies the index where the string will inserted in the list box.
lParam	A pointer (LPCSTR) to the string that is to be inserted.

If the list is an owner-drawn style without the LBS_HASSTRINGS style, the string pointer, rather than the string itself, is stored.

The LB_INSERTSTRING message does not cause a list with a LBS_SORT style to be sorted. Use the LB_ADDSTRING function for this capability.

D.79.2 Returns

The message returns the index where the string was actually inserted. If an error occurs, LB_ERR is returned. If insufficient space is available to store the string, LB_ERRSPACE is returned.

D.79.3 Cross-References

LB_ADDSTRING

D.80 LB_RESETCONTENT

D.80.1 Description

An application sends an LB_RESETCONTENT message to all items in a list box.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

If the list box style is owner-drawn without the LBS_HASSTRINGS style, the owner receives a WM_DELETEITEM message for each item in the list box.

D.80.2 Returns

None.

D.80.3 Cross-References

WM_DELETEITEM, LB_DELETESTRING

D.81 LB_SELECTSTRING

D.81.1 Description

An application sends an LB_SELECTSTRING message to search for an item in the list box that matches the specified string, and if a match is found, selects the item.

Parameter	Description
wParam	The index of the item before the first item searched.
lParam	A pointer (LPCSTR) to the string to be searched.

The search begins at the item after the one specified by *wParam*. When the end of the list is reached, the search continues from the top of the list until the specified item is reached. To search from the beginning of the list, pass -1 as the start index in *wParam*.

The search is not case sensitive.

D.81.2 Returns

The message returns the index of the selected item, if a match is found. If a match is not found, LB_ERR is returned.

D.81.3 Cross-Reference

LB_FINDSTRING, LB_ADDSTRING

D.82 LB_SELITEMRANGE

D.82.1 Description

An application sends an LB_SELITEMRANGE message to one or more items in a list box consecutively.

Parameter	Description
wParam	The selection flag.
lParam	The low-order word specifies the first item and the high-order word the last item to select.

If the selection flag is TRUE, the string is selected and highlighted. If it is FALSE, the selection is unselected and the highlighting is removed.

D.82.2 Returns

The message returns LB_ERR, if an error occurs.

D.82.3 Cross-References

LB_SELECTSTRING

D.83 LB_SETCARETINDEX

D.83.1 Description

An application sends an LB_SETCARETINDEX message to set the focus on an item in a multiple selection list box.

Parameter	Description
wParam	Specifies the item to receive focus in the list box.
lParam	The selection flag.

If the selection flag, *lParam*, is zero, the item is scrolled until it is fully visible. If it is non-zero, the selection is scrolled until it is at least partially visible.

D.83.2 Returns

The message returns LB_ERR, if an error occurs.

D.83.3 Cross-References

LB_GETCARETINDEX

D.84 LB_SETCOLUMNWIDTH

D.84.1 Description

An application sends an LB_SETCOLUMNWIDTH message to set the column width in a multiple column list box.

Parameter	Description
wParam	Specifies the width, in pixels, of all the columns.
lParam	Not used. Must be set to zero.

D.84.2 Returns

None.

D.84.3 Cross-References

LB_SETHORIZONTALEXTENT

D.85 LB_SETCURSEL

D.85.1 Description

An application sends an LB_SETCURSEL message to select an item in a single selection list box.

Parameter	Description
wParam	Specifies the index of the item to be selected and scrolled into view.
lParam	Not used. Must be set to zero.

The previously selected item in the list box is deselected.

If *wParam* is -1, the list box will have no current selection.

D.85.2 Returns

The message returns LB_ERR if an error occurs. If *wParam* is -1, LB_ERR is also returned, even though this is a valid operation.

D.85.3 Cross-References

LB_GETCURSEL, LB_SELECTSTRING, LB_SETSEL

D.86 LB_SETHORIZONTALEXTENT

D.86.1 Description

An application sends an LB_SETHORIZONTALEXTENT message to set the width that a list box can be scrolled horizontally.

Parameter	Description
wParam	Horizontal scroll width in pixels.
lParam	Not used. Must be set to zero.

If the size of the list box is greater than the specified width, the horizontal scroll bar is enabled to horizontally scroll items. If the size is smaller than list box, the horizontal scroll bar is hidden. The default size is set to zero, so that a scroll bar is not drawn.

The list box must have the WS_HSCROLL style set for the message to be handled.

D.86.2 Retu rns

None.

D.86.3 Cross-References

LB_GETHORIZONTALEXTENT, LB_SETCOLUMNWIDTH

D.87 LB_SETITEMDATA

D.87.1 Description

An application sends an LB_SETITEMDATA message to set a value that is associated with a specific item in the list box.

Parameter	Description
wParam	The index of the item associated with the data.
lParam	The doubleword value to associate to a list box item.

D.87.2 Returns

The message returns LB_ERR, if an error occurs.

D.87.3 Cross-References

LB_ADDSTRING, LB_GETITEMDATA

D.88 LB_SETITEMHEIGHT

D.88.1 Description

An application sends an LB_SETITEMHEIGHT message to set the height of items in a list box.

Parameter	Description
wParam	The index of the item for which the height is being set.
lParam	The low-order word specifies the height of the item in pixels.

If the list box has the LBS_OWNERDRAWVARIABLE style, only the height of the item specified is set. Otherwise, all items in the list are set to the specified height and wParam is ignored.

D.88.2 Returns

If an invalid index or height was specified, LB_ERR is returned.

D.88.3 Cross-References

LB_GETITEMHEIGHT

D.89 LB_SETSEL

D.89.1 Description

An application sends an LB_SETSEL message to select a string in a multiple selection list box.

Parameter	Description
wParam	The selection flag.
lParam	The low-order word specifies the index of the item to select.

If the selection flag is TRUE, the string is selected and highlighted. If it is FALSE, the selection is unselected and the highlighting is removed.

D.89.2 Returns

The message returns LB_ERR if an error occurs.

D.89.3 Cross-References

LB_GETSEL

D.90 LB_SETTABSTOPS

D.90.1 Description

An application sends an LB_SETTABSTOPS message to set the tab stops in a list box.

Parameter	Description
wParam	The number of tab stops.
lParam	A pointer to the tab stops array.

The tab stops array is an array of integers containing the tab stops in dialog box units. The tab stops are sorted in increasing order. If the number of tab stops in *wParam* is zero, the default tab stop of two dialog units is used.

If *wParam* is 1, the list box will have tab stops separated by the distance specified by *lParam*. If *wParam* is greater than one, a tab stop is set for each value in the tab stops array.

D.90.2 Returns

The message returns a non-zero value if all the tabs were set. Otherwise, zero is returned.

D.90.3 Cross-References

None.

D.91 LB_SETTOPINDEX

D.91.1 Description

An application sends an LB_SETTOPINDEX message to make sure an item in the list box is visible.

Parameter	Description
wParam	The index of the item in the list box.
lParam	Not used. Must be set to zero.

If the specified item is not in the list box, the list is scrolled until it is in view.

D.91.2 Returns

The message returns LB_ERR if an error occurs.

D.91.3 Cross-References

LB_GETTOPINDEX

D.92 STM_GETICON

D.92.1 Description

An application sends an STM_GETICON message to get the handle of an icon associated with the icon resource.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.92.2 Returns

The message returns the handle to the icon, if successful. If an error occurred or the icon has no associated icon resource, zero is returned.

D.92.3 Cross-References

STM_SETICON

D.93 STM_SETICON

D.93.1 Description

An application sends an STM_SETICON message to associate an icon with an icon resource.

Parameter	Description
wParam	The icon to associate with the icon resource.
lParam	Not used. Must be set to zero.

D.93.2 Returns

The message returns the handle of the previously associated icon, if successful. If an error occurred, zero is returned.

D.93.3 Cross-References

STM_GETICON

D.94 WM_ACTIVATE

D.94.1 Description

A WM_ACTIVATE message is sent whenever a window is being activated or deactivated. The window being deactivated is sent the message first. Then the message is sent to the window being activated.

Parameter	Description
wParam	Specifies whether the window is being activated or deactivated. If the window was activated by a mouse click, the parameter is WA_CLICKACTIVE. If it was activated my a means other than a mouse click, it is WA_ACTIVE. If the window is being deactivated, the parameter is WA_INACTIVE.
lParam	The high-order word specifies the minimized state of the window. A non-zero state means the window is minimized. The low-order word is the HWND handle of the window, which can be NULL.

D.94.2 Returns

The application should return zero if it processes the message.

D.94.3 Cross-References

WM_MOUSEACTIVATE, WM_NCACTIVATE

D.95 WM_ACTIVATEAPP

D.95.1 Description

A WM_ACTIVATEAPP message is sent to all top-level windows of the task that is being activated and the task being deactivated.

Parameter	Description
wParam	Specifies whether the window is being activated or deactivated. The value is TRUE for windows being activated, and FALSE for windows being deactivated.
lParam	The low-order word is the task handle (HTASK) of the window.

D.95.2 Returns

The application should return zero if it processes the message.

D.95.3 Cross-References

WM_ACTIVATE

D.96 WM_ASKCBFORMATNAME

D.96.1 Description

A WM_ASKCBFORMATNAME message is used to query the clipboard owner for the format name of the data in the clipboard. A clipboard viewer application sends the WM_ASKCBFORMATNAME message to a clipboard owner when the clipboard contains data that the clipboard owner should display. This is specified when the clipboard data handle is of the CF_OWNERDISPLAY format.

Parameter	Description
wParam	Specifies the maximum number of bytes to copy.
lParam	A pointer to the buffer where the copy of the format name is to be stored.

The clipboard owner copies the name of the CF_OWNERDISPLAY format into the buffer pointed to by *lParam*.

D.96.2 Returns

The application should return zero if it processes the message.

D.96.3 Cross-References

WM_PAINTCLIPBOARD

D.97 WM_CANCELMODE

D.97.1 Description

A WM_CANCELMODE message is sent to a window to cancel any internal modes, such as mouse capture, it may be running.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.97.2 Returns

The application should return zero if it processes the message.

D.97.3 Cross-References

None.

D.98 WM_CHANGECBCHAIN

D.98.1 Description

A WM_CHANGECBCHAIN message is sent to the first window in the clipboard viewer chain, notifying it that a window is being removed from the chain.

Parameter	**Description**
wParam	Specifies the window (HWND) being removed from the chain.
lParam	The low-order word specifies the window that follows the one being removed from the chain.

Each window that receives the WM_CHANGECBCHAIN message should pass the message on to the next window in the chain.

D.98.2 Returns

The application should return zero if it processes the message.

D.98.2 Cross-References

None.

D.99 WM_CHAR

D.99.1 Description

A WM_CHAR message is sent when a WM_KEYUP and a WM_KEYDOWN message are translated. The WM_CHAR message contains the value of the key being pressed or released.

Parameter	**Description**	
wParam	The virtual key code of the key.	
lParam	Provides the following additional information about the key:	
	Bits 0-15	Repeat count, indicating the number of times the keystroke is repeated as a result of holding down the key.
	Bits 16-23	Scan code, which is OEM dependent.
	Bit 24	If 1, it is extended key. Otherwise, the value is zero.
	Bits 25-26	Not used.
	Bits 27-29	Reserved.
	Bit 29	If 1, the ALT key is held down while the key is pressed; otherwise the value is zero.
	Bit 30	If 1, the key is down before the message is sent. Otherwise, the value is zero.
	Bit 31	If 1, the key is being released. Otherwise, the value is zero.

D.99.2 Returns

The application should return zero if it processes the message.

D.99.3 Cross-References

WM_KEYDOWN, WM_KEYUP

D.100 WM_CHARTOITEM

D.100.1 Description

A WM_CHARTOITEM message is sent by a list box with the LBS_WANTKEYBOARDINPUT style to its owner after receiving a WM_CHAR message.

Parameter	**Description**
wParam	The value of the key that was pressed.

lParam	The low-order word specifies the list box. The high-order word specifies the current list box caret position.

The list box must be an owner-drawn style and must not have the LBS_HASSTRINGS style set to receive this message.

D.100.2 Returns

The application returns a -2 if it handled all aspects of the selecting item and no further action is required of the list box. A -1 is returned to indicate that the list box should perform the default action in response to the keystroke. Returning a zero or greater indicates that the list box should perform the default action for the keystroke on the specified item.

D.100.3 Cross-References

WM_CHAR, WM_VKEYTOITEM

D.101 WM_CHILDACTIVATE

D.101.1 Description

An application sends a WM_CHILDACTIVATE message to a multiple document interface child window when the user clicks on the window's title bar or when the window is activated, moved or resized. The WM_CHILDACTIVATE message has no parameters.

D.101.2 Returns

The application must return zero if it processes this message.

D.101.3 Cross-References

MoveWindow(), SetWindowPos()

D.102 WM_CHOOSEFONT_GETLOGFONT

D.102.1 Description

An application sends a WM_CHOOSEFONT_GETLOGFONT message to the Choose Font dialog (created by a previous call to the *ChooseFont()* function) to get the current **LOGFONT** structure. The program uses this message to get LOGFONT data when the Choose Font dialog is open.

Parameter	Description
wParam	Must be zero.
lParam	Points to the **LOGFONT** structure, which receives information about the selected logical font.

D.102.2 Returns

None.

D.102.3 Cross-References

WM_GETFONT, **LOGFONT**, *ChooseFont()*

D.103 WM_CLEAR

D.103.1 Description

An application sends a WM_CLEAR message to either standalone edit control or edit control in a combo box, which alerts edit control to clear the current text selection, if any. To undo text deletion, the application can send an EM_UNDO message. To delete the current selection and put it into the clipboard, the application should use a WM_CUT message. A WM_CLEAR message has no parameters.

D.103.2 Returns

The return value is non-zero if this message is sent to edit control or to a combo box.

D.103.3 Cross-References

WM_UNDO, WM_COPY, WM_CUT, WM_PASTE

D.104 WM_CLOSE

D.104.1 Description

When an application receives a WM_CLOSE message, it terminates. The application usually destroys the window by calling *DestroyWindow()* when processing this message. A WM_CLOSE message has no parameters.

D.104.2 Returns

The application must return zero if it processes this message.

D.104.3 Cross-References

DestroyWindow(), PostQuitMessage(), WM_DESTROY, WM_QUERYENDSESSION, WM_QUIT

D.105 WM_COMMAND

D.105.1 Description

A WM_COMMAND message is sent to a window when an accelerator keystroke is translated, when a child control sends a notification to its parent window, or when the user selects a menu item. If an accelerator keystroke occurs that matches any menu item when the owning window is minimized, no WM_COMMAND message is sent. If the accelerator keystroke does not match a menu item, a WM_COMMAND is sent to the window regardless of its state.

Parameter	**Description**
wParam	Specifies the control or menu item identifier.
lParam	The low-order word of lParam specifies control's handle. The high-order word of lParam contains a notification message.

D.105.2 Returns

The application must return zero if it processes this message.

D.105.3 Cross-References

WM_SYSCOMMAND

D.106 WM_COMMNOTIFY

D.106.1 Description

A WM_COMMNOTIFY message is posted to a window by the communication device driver whenever a communication port event occurs. The message contains information about the status of the window's input and output queues. The application must clear each event to receive the next notification message.

Parameter	**Description**
wParam	Specifies the identifier of the communication device that is posting the message.
lParam	The low-order word of lParam specifies notification status and may be one or more of the following flags:

CN_EVENT	Indicates that an event has occurred that was enabled previously by a call to the *SetComEventMask()* function. The application should call *GetCommEventMask()* to determine the specific event and clear it.
CN_RECEIVE	Indicates that at least *cbInQueue* bytes are in the input queue. The *cbInQueue value is a parameter of the EnableCommNotification()* function.

CN_TRANSMIT	Indicates that at least *cbOutQueue* bytes are in the output queue. The *cbOutQueue* value is a parameter of the *EnableCommNotification()* function.

D.106.2 Returns

The application must return zero if it processes this message.

D.106.3 Cross-References

EnableCommNotification()

D.107 WM_COMPAREITEM

D.107.1 Description

A WM_COMPAREITEM message is sent to the owner of an owner-drawn combo box or list box to determine the relative position of a new item in a sorted list. The owner-drawn combo box should be created with the CBS_SORT style and the owner-drawn list box must contain the LBS_SORT style. When the owning window receives a WM_COMPAREITEM message, it returns a value that indicates the position of the new item.

Parameter	**Description**
wParam	Specifies the identifier of the control that sent the message.
lParam	Points to the **COMPAREITEMSTRUCT** structure.

D.107.2 Returns

If item 1 precedes item 2, this message returns -1.

If item 1 and 2 are equivalent, this messsage returns 0.

If item 1 follows item 2, this message returns 1.

D.107.3 Cross-References

COMPAREITEMSTRUCT

D.108 WM_COPY

D.108.1 Description

An application sends a WM_COPY message to standalone edit control or edit control in a combo box, which notifies edit control to copy the current selection to the clipboard in CF_TEXT format. This message has no parameters.

D.108.2 Returns

The return value is non-zero if this message is sent to edit control or to a combo box.

D.108.3 Cross-References

WM_CLEAR, WM_CUT, WM_PASTE

D.109 WM_CREATE

D.109.1 Description

A WM_CREATE message is sent when an application requests window creation by calling either *CreateWindow()* or *CreateWindowEx()* functions. The message is sent to the window procedure before *CreateWindow()* or *CreateWindowEx()* exits, and before the created window becomes visible.

Parameter	**Description**
wParam	Must be zero.
lParam	Points to the **CREATESTRUCT** structure, which contains information about the window being created.

D.109.2 Returns

The application must return zero if it processes this message. If it returns -1, the window is destroyed and the *CreateWindow()* or *CreateWindowEx()* function returns NULL.

D.109.3 Cross-References

CreateWindow(), CreateWindowEx(), WM_NCCREATE, **CREATESTRUCT**

D.110 WM_CTLCOLOR

D.110.1 Description

A WM_CTLCOLOR message is sent to the parent of a predefined control class or a message box when the control class or message box is ready to be drawn. The predefined control classes are combo boxes, edit controls, list boxes, static controls, buttons or scroll bars. For dialog boxes, the WM_CTLCOLOR message is sent to the dialog box procedure.

Parameter	**Description**
wParam	Identifies the display context for the child window.
lParam	The low-order word of *lParam* specifies the child window. The high-order word specifies the type of control and can be one of the following values:

CTLCOLOR_BTN	button
CTLCOLOR_DLG	dialog box
CTLCOLOR_EDIT	edit control
CTLCOLOR_LISTBOX	list box
CTLCOLOR_MSGBOX	message box
CTLCOLOR_SCROLLBAR	scroll bar
CTLCOLOR_STATIC	static control

D.110.2 Returns

If an application processes a WM_CTLCOLOR message, it must return a handle to the brush that is to be used to paint the control's background. Otherwise an application should return NULL.

D.110.3 Cross-References

SetBkColor()

D.111 WM_CUT

D.111.1 Description

An application sends a WM_CUT message to either stand-alone edit control or edit control in a combo box, which notifies edit control to delete the current selection and put the deleted text to the clipboard in CF_TEXT format. This message has no parameters.

D.111.2 Returns

The return value is non-zero if this message is sent to edit control or a combo box.

D.111.3 Cross-References

WM_CLEAR, WM_COPY, WM_PASTE

D.112 WM_DEADCHAR

D.112.1 Description

A WM_DEADCHAR message is sent to a window when WM_KEYUP or WM_KEYDOWN messages indicate a dead key character value. A dead key is a key that is combined with another key to create a composite character, such as an umlaut.

Parameter	Description	
wParam	Specifies the value of a dead key.	
lParam	Provides the following additional information about the key:	
	Bits 0-15	Repeat count. The value is the number of times the keystroke is repeated as a result of holding down the key.
	Bits 16-23	Scan code, which is OEM dependent.
	Bit 24	If 1, it is extended key. Otherwise, the value is zero.
	Bits 25-26	Not used.
	Bits 27-28	Reserved.
	Bit 29	If 1, the ALT key is held down while the key is pressed. Otherwise, the value is zero.
	Bit 30	If 1, the key is down before the message is sent. Otherwise, the value is zero.
	Bit 31	If 1, the key is being released. Otherwise, the value is zero.

D.112.2 Returns

The application must return zero if it processes this message.

D.112.3 Cross-References

WM_KEYDOWN

D.113 WM_DELETEITEM

D.113.1 Description

A WM_DELETEITEM message is sent to the owner of an owner-drawn list box or combo box when it is about to be destroyed or when items are removed as a result of a LB_DELETESTRING, LB_RESETCONTENT, CB_DELETESTRING or CB_RESETCONTENT message.

Parameter	Description
wParam	Specifies the control that sent the WM_DELETEITEM message.
lParam	Points to the **DELETEITEM** structure that contains information about the item being deleted.

D.113.2 Returns

The application must return TRUE if it processes this message.

D.113 3 Cross-References

LB_DELETESTRING, LB_RESETCONTENT, CB_DELETESTRING, CB_RESETCONTENT

D.114 WM_DESTROY

D.114 1 Description

A WM_DESTROY message is sent to a window when it is destroyed. The message is sent after the window is removed from the screen. The parent window receives the WM_DESTROY message before the child windows, so it can assume that all child windows still exist. This message has no parameters.

D.114.2 Returns

The application must return zero if it processes this message.

D.114.3 Cross-References

DestroyWindow(), PostQuitMessage(), WM_CLOSE

D.115 WM_DESTROYCLIPBOARD

D.115.1 Description

A WM_DESTROYCLIPBOARD message is sent to the clipboard owner when the contents of the clipboard are emptied by the *EmptyClipboard()* function call. This message has no parameters.

D.115.2 Returns

The application must return zero if it processes this message.

D.115.3 Cross-References

EmptyClipboard()

D.116 WM_DEVMODECHANGE

D.116.1 Description

A WM_DEVMODECHANGE message is sent to all top-level windows when the default device mode settings have changed.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Points to the device name specified in Windows initialization file.

D.116.2 Returns

The application must return zero if it processes this message.

D.116.3 Cross-References

ExtDeviceMode(), WM_WININICHANGE

D.117 WM_DRAWCLIPBOARD

D.117.1 Description

A WM_DRAWCLIPBOARD message is sent to the first window in the clipboard viewing chain when the contents of the clipboard change. An application can join the clipboard viewing chain by calling the *SetClipboardViewer()* function. Each window that receives the WM_DRAWCLIPBOARD message should pass the message on to the next window in the clipboard viewing chain. The handle of the next window is returned by the *SetClipboardViewer()* function and can be modified in response to a WM_CHANGECBCHAIN message. The WM_DRAWCLIPBOARD message has no parameters.

D.117.2 Returns

The application must return zero if it processes this message.

D.117.3 Cross-References

SendMessage(), SetClipboardViewer(), WM_CHANGECBCHAIN

D.118 WM_DRAWITEM

D.118.1 Description

A WM_DRAWITEM message is sent to the owner of an owner-drawn button, combo box, list box, or menu when the visual appearance of the button, combo box, list box, or menu has changed. Before returning from processing this message, an application should put the device context identified by the **hdc** member of the **DRAWITEMSTRUCT** structure back in the default state.

Parameter	**Description**
wParam	Specifies the control that sent the message, or zero if the message was sent by a menu.
lParam	Points to a **DRAWITEMSTRUCT** structure that contains information about the item being drawn.

D.118.2 Returns

The application must return TRUE if it processes this message.

D.118.3 Cross-References

WM_COMPAREITEM, WM_DELETEITEM, WM_INITDIALOG, WM_MEASUREITEM, **DRAWITEMSTRUCT**

D.119 WM_DROPFILES

D.119.1 Description

A WM_DROPFILES message is sent to the window when the user releases the left mouse button while in the window of an application that has registered itself as a recipient of dropped files. The WM_DROPFILES message is posted rather than sent.

Parameter	**Description**
wParam	Specifies the internal data structure, which represents dropped files. The value is valid only during the processing of a WM_DROPFILES message.
lparam	Not used. Must be set to zero.

D.119.2 Returns

The application must return zero if it processes this message.

D.119.3 Cross-References

DragAcceptFiles(), DragFinish(), DragQueryFile(), DragQueryPoint()

D.120 WM_ENABLE

D.120.1 Description

A WM_ENABLE message is sent when an application enables or disables a window. This message is sent before the *EnableWindow()* function returns, but after the WS_DISABLE style bit of the window has changed.

Parameter	**Description**
wParam	The value is TRUE if the window has been enabled, and FALSE if window has been disabled.
lparam	Not used. Must be set to zero.

D.120.2 Returns

The application must return zero if it processes this message.

D.120.3 Cross-References

EnableWindow()

D.121 WM_ENDSESSION

D.121.1 Description

A WM_ENDSESSION message is sent to an application that has returned a non-zero value in response to a WM_QUERYENDSESSION message. The WM_ENDSESSION message notifies the application whether the session is actually ending. The application does not need to call *DestroyWindow()* or *PostQuitMessage()* when processing this message.

Parameter	**Description**
wParam	The value is TRUE if the session is being ended, and FALSE otherwise.
lparam	Not used. Must be set to zero.

D.121.2 Returns

The application must return zero if it processes this message.

D.121.3 Cross-References

DestroyWindow(), ExitWindows(), PostQuitMessage(), WM_QUERYENDSESSION

D.122 WM_ENTERIDLE

D.122.1 Description

A WM_ENTERIDLE message is sent to an application's main window procedure when a modal dialog box or a menu is entering an idle state. A modal dialog box or menu enters an idle state when no messages are waiting in its message queue.

Parameter	Description
wParam	The value of this parameter can be MSGF_DIALOGBOX, which means the system is idle because a dialog box is being displayed, or MSGF_MENU, which has the same meaning for menu.
lParam	The low-order word is either dialog box handle (if *wParam* is MSGF_DIALOGBOX) or the handle of the window containing the displayed menu (if *wParam* is MSGF_MENU).

D.122.2 Returns

The application must return zero if it processes this message.

D.122.3 Cross-References

DefWindowProc()

D.123 WM_ERASEBKGND

D123.1 Description

A WM_ERASEBKGND message is sent when the window background needs to be erased. By default the *DefWindowProc()* function erases the background by using the class background brush specified by the **hbrbackground** member of the **WNDCLASS** structure. If the value of hbrbackground is NULL, the application should process the WM_ERASEBKGND message and erase the background color itself. When processing this message, the application must align the origin of the intended brush with the window coordinates by calling the *UnrealizeObject()* function for the brush.

Parameter	Description
wParam	Identifies the device context of the window.
lParam	Not used. Must be set to zero.

D.123.2 Returns

The application must return non-zero if it erases the background or zero otherwise.

D.123.3 Cross-References

UnrealizeObject(), WM_ICONERASEBKGND

D.124 WM_FONTCHANGE

D.124.1 Description

An application sends a WM_FONTCHANGE message to all top-level windows after changing the pool of available font resources. To do this, an application can call the *SendMessage()* function with the *hwnd* parameter set to HWND_BROADCAST. The WM_FONTCHANGE message has no parameters.

D.124.2 Returns

The application must return zero if it processes this message.

D.124.3 Cross-References

AddFontResource(), RemoveFontResource(), SendMessage()

D.125 WM_GETDLGCODE

D.125.1 Description

A WM_GETDLGCODE message is sent to the dialog box procedure associated with a control and contains information about the type of input the application is about to process. By responding to the WM_GETDLGCODE message, an application can trap a particular type of input and process the input itself.

D.125.2 Returns

The return value should be any combination of the following flags, indicating which type of input the application processes:

DLGC_BUTTON	Push button.
DLGC_DEFPUSHBUTTON	Default push button.
DLGC_HASSETSEL	Edit control's EM_SETSEL message.
DLGC_UNDEFPUSHBUTTON	There is no default push button processing.
DLGC_RADIOBUTTON	Radio button.
DLGC_STATIC	Static control.
DLGC_WANTALLKEYS	All keyboard input.
DLGC_WANTARROWKEYS	All arrow keys.
DLGC_WANTCHARS	WM_CHARS messages.
DLGC_WANTMESSAGE	All keyboard input (the application passes this message on to the control).
DLGC_WANTTAB	TAB key.

D.125.3 Cross-References

DefWindowProc()

D.126 WM_GETFONT

D.126.1 Description

An application sends a WM_GETFONT message to a control to get the current font associated with that control. This message has no parameters.

D.126.2 Returns

The return value is either the HFONT value of the font or NULL if the control uses the default system font.

D.126.3 Cross-References

WM_SETFONT

D.127 WM_GETMINMAXINFO

D.127.1 Description

A WM_GETMINMAXINFO message is sent to a window whenever the system needs the maximized position and dimensions of a window or a window's minimum/maximum tracking size. By default, the system fills in a **MINMAXINFO** data structure, specifying default values for the all positions and dimensions. The application can change these values when it processes this message.

Parameter	**Description**
wParam	Points to the **MINMAXINFO** data structure.
lParam	Not used. Must be set to zero.

D.127.2 Returns

The application must return zero if it processes this message.

D.127.3 Cross-References

MINMAXINFO

D.128 WM_GETTEXT

D.128.1 Description

An application sends a WM_GETTEXT message to copy the text associated with a window into a buffer provided by the caller. The window text depends on the type of window. For an edit control, the text to be copied is the contents of the edit control. For a combo box, the text is the contents of the edit control or static-text portion of the combo box. For a button, the text is the button name. For other windows, except list boxes, the text is the window title.

Parameter	Description
wParam	Specifies the length of the buffer into which the string is to be copied, including the terminating null character.
lParam	Points to the buffer.

D.128.2 Returns

The return value is the number of bytes copied. In the case of a combo box with no edit control, the return value is CB_ERR.

D.128.3 Cross-References

LB_GETTEXTLEN, WM_GETTEXT

D.129 WM_GETTEXTLENGTH

D.129.1 Description

An application sends a WM_GETTEXTLENGTH message to determine the length of text associated with a window. The length of the window text depends on the type of window. For an edit control, the text to be copied is the contents of the edit control. For a combo box, the text is the contents of the edit-control or static-text portion of the combo box. For a button, the text is the button name. For other windows, except list boxes, the text is the window title. The length is returned in bytes and the terminating null character is not included. The WM_GETTEXTLENGTH message has no parameters.

D.129.2 Returns

The return value specifies the length (in bytes) of the text.

D.129.3 Cross-References

LB_GETTEXTLEN, WM_GETTEXT

D.130 WM_HSCROLL

D.130.1 Description

A WM_HSCROLL message is sent to a window when a user clicks on its horizontal scroll bar.

Parameter	Description
wParam	Specifies a scroll bar code that indicates a scrolling request. Scrolling requests can be one of the following values:

SB_BOTTOM	Scroll to the bottom.
SB_ENDSCROLL	End scroll.
SB_LINEDOWN	Scroll one line down.
SB_LINEUP	Scroll one line up.
SB_PAGEDOWN	Scroll one page down.

	SB_PAGEUP	Scroll one page up.
	SB_THUMBPOSITION	Scroll to the position specified by the low-order word of *lParam*.
	SB_THUMBTRACK	Drag to the position specified by low-order word of *lParam*.
	SB_TOP	Scroll to the top.
lParam	The low-order word of *lParam*, which specifies the current position of the scroll box when *wParam* is either SB_THUMBPOSITION or SB_THUMBTRACK. Otherwise, the low-order word is not used. The high-order word identifies the control if a WM_HSCROLL message is sent by a scroll bar. Otherwise, the high-order word is not used.	

D.130.2 Returns

The application must return zero if it processes this message.

D.130.3 Cross-References

SetScrollPos(), WM_VSCROLL

D.131 WM_HSCROLLCLIPBOARD

D.131.1 Description

A WM_HSCROLLCLIPBOARD message is sent to the owner of the clipboard when its data has the CF_OWNERDISPLAY format and an event occurs in the clipboard viewer's horizontal scroll bar. The owner should scroll the clipboard image, invalidate it and update the scroll bar values.

Parameter	**Description**	
wParam	Specifies a window of the clipboard viewer.	
lParam	The low-order word of *lParam* specifies a scroll bar code that indicates a scrolling request. Scrolling requests can be one of the following values:	
	SB_BOTTOM	Scroll to the lower right.
	SB_ENDSCROLL	End scroll.
	SB_LINEDOWN	Scroll one line down.
	SB_LINEUP	Scroll one line up.
	SB_PAGEDOWN	Scroll one page down.
	SB_PAGEUP	Scroll one page up.
	SB_THUMBPOSITION	Scroll to the absolute position.
	SB_TOP	Scroll to the upper left.

The high-order word specifies the scroll position if the scroll bar code is SB_THUMBPOSITION. Otherwise, it is not used.

D.131.2 Returns

The application must return zero if it processes this message.

D.131.3 Cross-References

InvalidateRect(), WM_VSCROLLCLIPBOARD

D.132 WM_ICONERASEBKGND

D.132.1 Description

A WM_ICONERASEBKGND message is sent to a minimized window when its background must be filled before painting the icon. A window receives this message only if a class icon is defined for the window.

Otherwise, WM_ERASEBKGND is sent instead. By default, the *DefWindowProc()* function fills the icon background with the background brush of the parent window.

Parameter	**Description**
wParam	Specifies the device context of the icon.
lParam	Not used. Must be set to zero.

D.132.2 Returns

The application must return zero if it processes this message.

D.132.3 Cross-References

DefWindowProc(), WM_ERASEBKGND

D.133 WM_INITDIALOG

D.133.1 Description

A WM_INITDIALOG message is sent to a dialog box window procedure immediately before the dialog box is displayed.

Parameter	**Description**
wParam	Specifies the first child control that can be given the input focus.
lParam	Specifies the application specific data that can be passed by one of the following functions: *CreateDialogParam(), DialogBoxIndirectParam(),* or *DialogBoxParam().*

D.133.2 Returns

An application must return non-zero if it wants to set the default input focus to the control identified by the *wParam*.

If the dialog box procedure uses the *SetFocus()* function to set the input focus to a different child control, the application should return zero.

D.133.3 Cross-References

CreateDialogParam(), DialogBoxIndirectParam(), DialogBoxParam(), SetFocus()

D.134 WM_INITMENU

D.134.1 Description

A WM_INITMENU message is sent when a menu associated with a window is about to become active. This message occurs when a user clicks on the menu item or presses a menu hotkey. The WM_INITMENU message allows an application to modify the menu before it is displayed.

Parameter	**Description**
wParam	Specifies the menu.
lParam	Not used. Must be set to zero.

D.134.2 Returns

The application must return zero if it processes this message.

D.134.3 Cross-References

WM_INITMENUPOPUP

D.135 WM_INITMENUPOPUP

D.135.1 Description

A WM_INITMENUPOPUP message is sent when a pop-up menu associated with a window is about to become active. This allows an application to modify the pop-up menu before it is displayed.

Parameter	Description
wParam	Specifies the pop-up menu.
lParam	The low-order word specifies the index of the pop-up menu in the main menu. The high-order word is non-zero if the pop-up menu is the System menu. Otherwise, the high-order word is zero.

D.135.2 Returns

The application must return zero if it processes this message.

D.135.3 Cross-References

WM_INITMENU

D.136 WM_KEYDOWN

D.136.1 Description

The WM_KEYDOWN message is sent for non-system keys and keys pressed while the window has input focus.

Parameter	Description	
wParam	The virtual key code	
lParam	Key data:	
	Bits 0 through 15	Specify the repeat count.
	Bits 16 through 23	Specify the manufacturer's scan code.
	Bit 24	Specifies whether the key was an extended key.
	Bits 25 and 26	Not used.
	Bits 27 and 28	Used internally by the OS.
	Bit 29	Context code that indicates if the ALT key was pressed.
	Bit 30	Indicates the previous state of the key. It is set if the key was down before the message was sent, or clear if the key was up.
	Bit 31	Indicates the transition status. It is set if the key is being released, or clear if it is being pressed.

Note: For WM_KEYDOWN, bits 29 and 30 are clear, whereas bit 30 will indicate if this is the first WM_KEYDOWN. For 101 and 102 keyboards, the following keys are considered to be enhanced keys: right ALT, right CTRL, as well as the INSERT, DELETE, HOME, END, PAGE UP, PAGE DOWN, UP, DOWN, LEFT and RIGHT keys, which are not part of the numeric keypad, and the / and ENTER keys, which are a part of the numeric keypad.

D.136.2 Returns

If the application processes this message, it should return zero.

D.136.3 Cross-References

WM_CHAR, WM_KEYUP

D.137 WM_KEYUP

D.137.1 Description

The WM_KEYUP message is sent for non-system keys and keys released while the window has input focus.

Parameter	Description	
wParam	The virtual key code	
lParam	Key data:	
	Bits 0-15	Specify the repeat count.

Bits 16-23	Specify the manufacturer’s scan code.
Bit 24	Specifies whether the key was an extended key.
Bits 25-26	Not used.
Bits 27-28	Used internally by the OS.
Bit 29	Context code that indicates if the ALT key was pressed.
Bit 30	Indicates the previous state of the key. It is set if the keywas down before the message was sent, or clear if the keywas up.
Bit 31	Indicates the transition status. It is set if the key is being released, or clear if it is being pressed.

Note: For WM_KEYUP, bit 29 is clear, whereas bit 31 is set. For 101 and 102 keyboards, the following keys are considered to be enhanced keys: right ALT, right CTRL, as well as the INSERT, DELETE, HOME, END, PAGE UP, PAGE DOWN, UP, DOWN, LEFT and RIGHT keys, which are not part of the numeric keypad, and the / and ENTER keys, which are a part of the numeric keypad.

D.137.2 Returns

If the application processes this message, it should return zero.

D.137.3 Cross-References

WM_CHAR, WM_KEYDOWN

D.138 WM_KILLFOCUS

D.138.1 Description

The WM_KILLFOCUS message is sent when a window is about to lose input focus.

Parameter	**Description**
wParam	The window that will receive focus.
lParam	Not used. Must be set to zero.

D.138.2 Returns

If the application processes this message, it should return zero.

D.138.3 Cross-References

SetFocus(), WM_SETFOCUS

D.139 WM_LBUTTONDBLCLK

D.139.1 Description

The WM_LBUTTONDBLCLK message is sent when the user double-clicks the left mouse button.

Parameter	**Description**	
wParam	Key status, which can be one or more of the following values:	
	MK_CONTROL	The CTRL key is pressed.
	MK_LBUTTON	The left button is pressed.
	MK_MBUTTON	The middle button is pressed.
	MK_RBUTTON	The right button is pressed.
	MK_SHIFT	The SHIFT key is pressed.
lParam	LOWORD is the horizontal position and HIWORD is the vertical position.	

Note: Only windows whose window class has the CS_DBLCLKS style receive double-click messages. Double-clicks are generated when the user presses and releases the left mouse button twice within the system's time limit.

A double-click generates the following sequence of messages: WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDBLCLK, and WM_LBUTTONUP.

D.139.2 Returns

If the application processes this message, it should return zero.

D.139.3 Cross-References

WM_LBUTTONDOWN, WM_LBUTTONUP

D.140 WM_LBUTTONDOWN

D.140.1 Description

The WM_LBUTTONDOWN message is sent when the user presses the left mouse button.

Parameter	Description	
wParam	Key status, which can be one or more of the following values:	
	MK_CONTROL	The CTRL key is pressed.
	MK_MBUTTON	The middle button is pressed.
	MK_RBUTTON	The right button is pressed.
	MK_SHIFT	The SHIFT key is pressed.
lParam	LOWORD is the horizontal position and HIWORD is the vertical position.	

D.140.2 Returns

If the application processes this message, it should return zero.

D.140.3 Cross-References

WM_LBUTTONDBLCLK, WM_LBUTTONUP

D.141 WM_LBUTTONUP

D.141.1 Description

The WM_LBUTTONUP message is sent when the user releases the left mouse button.

Parameter	Description	
wParam	Key status, which can be one or more of the following values:	
	MK_CONTROL	The CTRL key is pressed.
	MK_MBUTTON	The middle button is pressed.
	MK_RBUTTON	The right button is pressed.
	MK_SHIFT	The SHIFT key is pressed.
lParam	LOWORD is the horizontal position and HIWORD is the vertical position.	

D.141.2 Returns

If the application processes this message, it should return zero.

D.141.3 Cross-References

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN

D.142 WM_MBUTTONDBLCLK

D.142.1 Description

The WM_MBUTTONDBLCLK message is sent when the user double-clicks the middle mouse button.

Parameter	Description	
wParam	Key status, which can be one or more of the following values:	
	MK_CONTROL	The CTRL key is pressed.
	MK_LBUTTON	The left button is pressed.
	MK_MBUTTON	The middle button is pressed.
	MK_RBUTTON	The right button is pressed.
	MK_SHIFT	The SHIFT key is pressed.
lParam	LOWORD is the horizontal position and HIWORD is the vertical position.	

Note: Only windows whose window class has the CS_DBLCLKS style receive double-click messages. Double-clicks are generated when the user presses and releases the mouse twice within the system's time limit. A double-click generates the following sequence of messages: WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MBUTTONDBLCLK, and WM_MBUTTONUP.

D.142.2 Returns

If the application processes this message, it should return zero.

D.142.3 Cross-References

WM_MBUTTONDOWN, WM_MBUTTONUP

D.143 WM_MBUTTONDOWN

D.143.1 Description

The WM_MBUTTONDOWN message is sent when the user presses the middle mouse button.

Parameter	Description	
wParam	Key status, which can be one or more of the following values:	
	MK_CONTROL	The CTRL key is pressed.
	MK_LBUTTON	The left button is pressed.
	MK_RBUTTON	The right button is pressed.
	MK_SHIFT	The SHIFT key is pressed.
lparam	LOWORD is the horizontal position and HIWORD is the vertical position.	

D.143.2 Returns

If the application processes this message, it should return zero.

D.143.3 Cross-References

WM_MBUTTONDBLCLK, WM_MBUTTONUP

D.144 WM_MBUTTONUP

D.144.1 Description

The WM_MBUTTONUP message is sent when the user releases the middle mouse button.

Parameter	Description	
wParam	Key status, which can be one or more of the following values:	
	MK_CONTROL	The CTRL key is pressed.
	MK_LBUTTON	The left button is pressed.
	MK_RBUTTON	The right button is pressed.
	MK_SHIFT	The SHIFT key is pressed.
lParam	LOWORD is the horizontal position and HIWORD is the vertical position.	

D.144.2 Returns

If the application processes this message, it should return zero.

D.144.3 Cross-References

WM_MBUTTONDBLCLK, WM_MBUTTONDOWN

D.145 WM_MDIACTIVATE

D.145.1 Description

The WM_MDIACTIVATE message is sent to MDI client windows to change the active MDI child window and the MDI child windows to either activate or deactivate them.

Parameter	**Description**
wParam	Specifies the child window to activate for MDI client windows or the activation flag for MDI child windows.
lParam	Not used. Must be zero for MDI client windows, or LOWORD if a child window is being activated and HIWORD if a child windows is being deactivated for MDI child windows.

If the frame window is being activated, the child window that was last active receives a WM_NACTIVATE message, but does not receive a WM_MDIACTIVATE message.

D.145.2 Returns

If the application processes this message, it should return zero.

D.145.3 Cross-References

WM_MDIGETACTIVE, WM_NCACTIVATE, WM_MDINEXT

D.146 WM_MDICASCADE

D.146.1 Description

A WM_MDICASCADE message is sent to a MDI client window to arrange its windows in a cascade format.

Parameter	**Description**
wParam	Specifies the cascade flag.
lParam	Not used. Must be set to zero.

The cascade flag MDITILE_SKIPDISABLED prevents disabled child windows from being cascaded.

D.146.2 Returns

The application should return zero if it processes the message.

D.146.3 Cross-References

WM_MDIICONARRANGE, WM_MDITILE

D.147 WM_MDICREATE

D.147.1 Description

An application sends a WM_MDICREATE message to a MDI client window to create a child window.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	A pointer to an **MDICREATESTRUCT** structure.

The child window will have the style bits WM_CHILD, WS_CLIPSIBLINGS, WS_CLIPCHILDREN, WS_SYSMENU, WS_CAPTION, WS_THICKFRAME, WS_MINIMIZEBOX and WS_MAXIMIZEBOX in addition to the style bits in the **MDICREATESTRUCT** structure.

If the MDIS_ALLCHLDSTYLES style is set when the client window was created, *CreateWindow()* will override the default style bits.

When the MDI child window is created, it receives a WM_CREATE message, where the **MDICREATESTRUCT** structure is referenced by the **lpCreateParams** pointer in the **CREATESTRUCT** structure. A second WM_MDICREATE message must not be sent while the WM_MDICREATE message is still being processed.

D.147.2 Returns

The low-order word contains the handle of the new child window.

D.147.3 Cross-References

WM_MDIDESTROY, MDICREATESTRUCT

D.148 WM_MDIDESTROY

D.148.1 Description

An application sends a WM_MDIDESTROY message to a MDI client window to destroy a child window.

Parameter	Description
wParam	Specifies the child window (HWND) to be destroyed.
lParam	Not used. Must be set to zero.

D.148.2 Returns

The application should return zero if it processes the message.

D.148.3 Cross-References

WM_MDIDESTROY

D.149 WM_MDIGETACTIVE

D.149.1 Description

The WM_MDIGETACTIVE message gets the MDI child window that is active.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.149.2 Returns

The low-order word contains the handle to the active MDI child window. The high-order word is 1 if the window is maximized. Otherwise, it is zero.

D.149.3 Cross-References

WM_MDIACTIVATE

D.150 WM_MDIICONARRANGE

D.150.1 Description

The WM_MDIICONARRANGE message instructs an MDI client window to arrange all of its minimized child window icons.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.150.2 Returns

The application should return zero if it processes the message.

D.150.3 Cross-References

WM_MDICASCADE, WM_MDITILE

D.151 WM_MDIMAXIMIZE

D.151.1 Description

The WM_MDIMAXIMIZE message instructs an MDI client window to maximize the specified child window.

Parameter	Description
wParam	Specifies the child window (HWND) to maximize.
lParam	Not used. Must be set to zero.

D.151.2 Returns

The application should return zero if it processes the message.

D.151.3 Cross-References

None.

D.152 WM_MDINEXT

D.152.1 Description

The WM_MDINEXT message instructs an MDI client window to activate the child window behind the currently active child window and send the currently active window behind all other child windows.

Parameter	Description
wParam	Specifies the child window (HWND).
l*Param*	If the value is zero, the next child window is activated. If the value is non-zero, the previous child window is activated.

D.152.2 Returns

The application should return zero if it processes the message.

D.152.3 Cross-References

None.

D.153 WM_MDIRESTORE

D.153.1 Description

The WM_MDIRESTORE message instructs an MDI client window to restore a child window from the minimized or maximized size.

Parameter	Description
wParam	Specifies the child window (HWND).
lParam	Not used. Must be set to zero.

D.153.2 Returns

The application should return zero if it processes the message.

D.153.3 Cross-References

WM_MDIMAXIMIZE

D.154 WM_MDISETMENU

D.154.1 Description

The WM_MDISETMENU message is sent to replace the menu of an MDI frame window, the window pop-up menu, or both.

Parameter	Description
wParam	Refresh flag. If TRUE, the menus are refreshed. If FALSE, the *lParam* specifies new menus for the window.

lParam	The low-order word specifies the new frame window menu. The high-order word specifies the new Window pop-up menu. If either parameter is zero, the respective menu is left untouched.

D.154.2 Returns

The handle of the frame window menu replaced with this message.

D.154.3 Cross-References

None.

D.155 WM_MDITILE

D155.1 Description

A WM_MDITILE message is sent to a MDI client window to arrange its windows in a tile format.

Parameter	**Description**
wParam	Specifies the tile flag. If the flag is MDITILE_HORIZONTAL, the child windows are tiled wide. If the flag is MDITILE_VERTICAL, the child windows are tiled tall. If the flag is MDITILE_SKIPDISABLED, disabled child windows are not tiled.
lParam	Not used. Must be set to zero.

D.155.2 Returns

The application should return zero if it processes the message.

D.155.3 Cross-References

WM_MDICASCADE

D.156 WM_MEASUREITEM

D.156.1 Description

A WM_MEASUREITEM message is sent to an owner-drawn control to obtain its dimensions. The control can be a button, combo box, list box, or menu item.

Parameter	**Description**
wParam	Specifies the control that sent the WM_MEASUREITEM message. If the parameter is zero, the request was sent by a menu. If it is -1, the system is requesting dimensions of an edit control in a owner-drawn combo box.
lParam	A pointer to a **MEASUREITEMSTRUCT** structure to be filled by the owner of the control.

D.156.2 Returns

The application should return TRUE if it processes the message.

D.156.3 Cross-References

WM_COMPAREITEM, WM_DELTEITEM, WM_DRAWITEM

D.157 WM_MENUCHAR

D.157.1 Description

A WM_MENUCHAR message is sent when a key is pressed corresponding to a menu mnemonic character that does not match any predefined mnemonics in the menu. The message is sent to the window that owns the menu.

Parameter	**Description**
wParam	Specifies the ASCII character of the key pressed.
lParam	The low-order word specifies the type of selected menu. The type MF_POPUP indicates a pop-up menu, and MF_SYSMENU indicates a system menu. The high-order word identifies the selected menu.

D.157.2 Returns

The application should return TRUE if it processes the message.

D.157.3 Cross-References

WM_COMPAREITEM, WM_DELTEITEM, WM_DRAWITEM

D.158 WM_MENUSELECT

D.158.1 Description

A WM_MENUSELECT message is sent to the window that owns the menu, when a menu item has been selected.

Parameter	Description	
wParam	Specifies the menu item identifier if it is a menu item. If the item is a pop-up menu, it specifies the pop-up menu's handle.	
lParam	The high-order word specifies the system menu handle if the MF_SYSMENU flag is set in the low-order word. The low-order word specifies one or more of the following flags.	
	MF_BITMAP	Menu item is a bitmap.
	MF_CHECKED	Menu item is checked.
	MF_DISABLED	Menu item is disabled.
	MF_GRAYED	Menu item is grayed out.
	MF_MOUESELECT	Menu item was selected using the mouse.
	MF_OWNERDRAW	Menu item is owner draw.
	MF_POPUP	Menu item contains a pop-up menu.
	MF_SEPARATOR	Menu item is a separator.
	MF_SYSMENU	Menu item is in the system menu.

D.158.2 Returns

The application should return zero if it processes the message.

D.158.3 Cross-References

None.

D.159 WM_MOUSEACTIVATE

D.159.1 Description

A WM_MOUSEACTIVATE message is sent when the mouse is pressed in an inactive window.

Parameter	Description
wParam	Specifies the top-level parent window (HWND) of the window being activated.
lParam	The low-order word specifies the hit test area code. The high-order word specifies the identifier of the message.

D.159.2 Returns

The application return value determines the systems handling of the mouse event. If MA_ACTIVATE is returned, the window is activated. If it is MA_NOACTIVATE, the window is not activated. If MA_ACTIVATEANDEAT, the window is activated and the mouse event discarded. If MA_NOACTIVATEANDEAT is specified, the window is not activated and the mouse event is discarded.

D.159.3 Cross-References

None.

D.160 WM_MOUSEMOVE

D.160.1 Description

A WM_MOUSEMOVE message is sent when the mouse is moved within a window.

Parameter	Description	
wParam	Specifies the status of several keys and can be any combination of these values:	
	MK_CONTROL	Control key is down.
	MK_LBUTTON	Left button is down.
	MK_MBUTTON	Middle button is down.
	MK_RBUTTON	Right button is down.
	MK_SHIFT	Shift key is down.
lParam	The low-order word specifies the *x* screen coordinate of the mouse. The high-order word specifies the *y* screen coordinate of the mouse.	

If the mouse is captured, the message goes to the window holding the capture. Otherwise, it will go to the window directly under the cursor.

D.160.2 Returns

The application should return zero if it processes the message.

D.160.3 Cross-References

WM_NCHITTEST

D.161 WM_MOVE

D.160.1 Description

The WM_MOVE message is sent after a window has been moved.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	The low-order word of *lParam* specifies the new x-coordinate of the upper-left corner of the window's client area. The high-order word of *lParam* specifies the new y-coordinate of the upper-left corner of the window's client area. The low-order and high-order words of *lParam* are given in screen coordinates for overlapped and pop-up windows and in parent-client coordinates for child windows. An application can use the MAKEPOINT macro to convert the *lParam* parameter to a **POINT** data structure.

D.160.2 Returns

The application should return zero if it processes this message.

D.160.3 Cross-References

MAKEPOINT, **POINT**

D.161 WM_NCACTIVATE

D.161.1 Description

The WM_NCACTIVATE message is sent to a window when its non-client area needs to be changed to indicate an active or inactive state.

Parameter	Description
wParam	Specifies when a title bar or icon needs to be changed to indicate an active or inactive state. The *wParam* parameter is TRUE if an active title bar or icon is to be drawn. It is FALSE for an inactive title bar or icon.
lParam	Not used. Must be set to zero.

The *DefWindowProc()* function draws the title bar and title bar text in their active colors when the *wParam* parameter is TRUE and in their inactive colors when *wParam* is FALSE.

D.161.2 Returns

When the *wParam* parameter is FALSE, an application should return TRUE to indicate that Windows should proceed with the default processing or FALSE to prevent the caption bar or icon from being deactivated. When *wParam* is TRUE, the return value is ignored.

D.161.3 Cross-References

DefWindowProc()

D.162 WM_NCCALCSIZE

D.162.1 Description

The WM_NCCALCSIZE message is sent when the size and position of a window's client area needs to be calculated. By processing this message, an application can control the contents of the window's client area when the size or position of the window changes.

Parameter	Description
wParam	Specifies whether the application should specify which part of the client area contains valid information. Windows copies the valid information to the specified area within the new client area. If this parameter is TRUE, the application should specify which part of the client area is valid.
LParam	Points to an **NCCALCSIZE_PARAMS** data structure that contains information an application can use to calculate the new size and position of the client rectangle.

Regardless of the value of *wParam*, the first rectangle in the array specified by the rgrc member contains the coordinates of the window. For a child window, the coordinates are relative to the parent window's client area. For top-level windows, the coordinates are screen coordinates. An application should process WM_NCCALCSIZE by modifying the **rgrc[0]** rectangle to reflect the size and position of the client area. The **rgrc[1]** and **rgrc[2]** rectangles are valid only if *wParam* is TRUE. In this case, the **rgrc[1]** rectangle contains the coordinates of the window before it was moved or resized. The **rgrc[2]** rectangle contains the coordinates of the window's client area before the window was moved. All coordinates are relative to the parent window or screen.

Redrawing of the window can occur, depending on whether CS_HREDRAW or CS_VREDRAW is specified, which is the default, backward-compatible *DefWindowProc()* processing of this message (in addition to the usual client rectangle calculation described in the following table).

D.162.2 Returns

An application should return zero if *wParam* is FALSE.

An application can return zero or a valid combination of the following values if *wParam* is TRUE:

Value	Meaning
WVR_ALIGNTOP, WVR_ALIGNLEFT, WVR_ALIGNBOTTOM, WVR_ALIGNRIGHT	These values, used in combination, specify that the client area of the window is to be preserved and aligned appropriately relative to the new location of the client window. For example, to align the client area to the lower-left, return WVR_ALIGNLEFT | WVR_ALIGNTOP.
WVR_HREDRAW, WVR_VREDRAW	These values, used in combination with any other values, cause the window to be completely

redrawn if the client rectangle changed size horizontally or vertically. These values are similar to the CS_HREDRAW and CS_VREDRAWclass styles.

WVR_REDRAW
: This value causes the entire window to be redrawn. It is a combination of WVR_HREDRAW and WVR_VREDRAW.

WVR_VALIDRECTS
: This value indicates that, upon return from WM_NCCALCSIZE, the **rgrc[1]** and **rgrc[2]** rectangles contain valid source and destination area rectangles, respectively. Windows combines these rectangles to calculate the area of the window that can be preserved. Windows copies any part of the window image that is within the source rectangle and clips the image to the destination rectangle. Both rectangles are in parent-relative or screen-relative coordinates.

: This return value allows an application to implement more elaborate client-area preservation strategies, such as centering or preserving a subset of the client area.

If *wParam* is TRUE and an application returns zero, the old client area is preserved and is aligned with the upper-left corner of the new client area.

D.162.3 Cross-References

*DefWindowProc(), MoveWindow(), SetWindowPos()***,** **RECT**, WM_NCCALCSIZE

D.163 WM_NCCREATE

D.163.1 Description

The WM_NCCREATE message is sent prior to the WM_CREATE message when a window is first created.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Points to the **CREATESTRUCT** data structure for the window.

Scroll bars are initialized (the scroll bar position and range are set), and the window text is set. Memory used internally to create and maintain the window is allocated.

D.163.2 Returns

The return value is non-zero if the non-client area is created. It is zero if an error occurs. In this case, the *CreateWindow()* or *CreateWindowEx()* functions returns NULL.

D.163.3 Cross-References

CreateWindow(), WM_CREATE, **CREATESTRUCT**

D.164 WM_NCDESTROY

D.164.1 Description

The WM_NCDESTROY message informs a window that its non-client area is being destroyed. The *DestroyWindow()* function sends the WM_NCDESTROY message to the window following the WM_DESTROY message. WM_NCDESTROY is used to free the allocated memory object associated with the window.

This message frees any memory internally allocated for the window, and has no parameters.

D.164.2 Returns

An application should return zero if it processes this message.

D.164.3 Cross-References

DestroyWindow(), WM_NCCREATE

D.165 WM_NCHITTEST

D.165.1 Description

The WM_NCHITTEST message is sent to the window that contains the cursor or to the window that uses the *SetCapture()* function to capture the mouse input. It is sent every time the mouse is moved.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates.
	The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.
	The MAKEPOINT macro can be used to convert the *lParam* parameter to a **POINT** structure.

D.165.2 Returns

The return value of the *DefWindowProc()* function is one of the following values indicating the position of the cursor:

Value	Meaning
HTBORDER	The cursor is located in the border of a window that does not have a sizing border.
HTBOTTOM	The cursor is located in the lower horizontal border of a window.
HTBOTTOMLEFT	The cursor is located in the lower-left corner of a window border.
HTBOTTOMRIGHT	The cursor is located in the lower-right corner of a window border.
HTCAPTION	The cursor is located in a title bar area.
HTCLIENT	The cursor is located in a client area.
HTERROR	The cursor is located in the screen background or on a dividing line between windows (same as HTNOWHERE, except that the *DefWindowProc()* function produces a system beep to indicate an error).
HTGROWBOX	The cursor is located in a size box (same as HTSIZE).
HTHSCROLL	The cursor is located in the horizontal scroll bar.
HTLEFT	The cursor is located in the left border of a window.
HTMAXBUTTON	The cursor is located in a Maximize button.
HTMENU	The cursor is located in a menu area.
HTMINBUTTON	The cursor is located in a Minimize button.
HTNOWHERE	The cursor is located on the screen background or on a dividing line between windows.
HTREDUCE	The cursor is located in a Minimize button.
HTRIGHT	The cursor is located in the right border of a window.
HTSIZE	The cursor is located in a size box (same as HTGROWBOX).
HTSYSMENU	The cursor is located in a System menu (sometimes referred to as a Control menu) or in a close button in a child window.
HTTOP	The cursor is located in the upper horizontal border of a window.
HTTOPLEFT	The cursor is located in the upper-left corner of a window border.
HTTOPRIGHT	The cursor is located in the upper-right corner of a window border.
HTTRANSPARENT	The cursor is located in a window currently covered by another window.

HTVSCROLL	The cursor is located in the vertical scroll bar.
HTZOOM	The cursor is located in a Maximize button.

D.165.3 Cross-References

DefWindowProc(), GetCapture(), MAKEPOINT, **POINT**

D.166 WM_NCLBUTTONDBLCLK

D.166.1 Description

The WM_NCLBUTTONDBLCLK message is sent when the user double-clicks the left mouse button while the cursor is within a non-client area of the window.

Parameter	**Description**
wParam	Specifies the code returned by WM_NCHITTEST. For more information, see the description of the WM_NCHITTEST message.
lParam	The low-order word of *lParam* specifies the horizontal position of the cursor, in screen coordinates.
	The high-order word of *lParam* specifies the vertical position of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.166.2 Returns

An application should return zero if it processes this message.

D.166.3 Cross-References

WM_NCHITTEST, WM_SYSCOMMAND, **POINT**, WM_NCLBUTTONDBLCLK

D.167 WM_NCLBUTTONDOWN

D.167.1 Description

The WM_NCLBUTTONDOWN message is sent to a window when the user presses the left mouse button while the cursor is within a non-client area of the window.

Parameter	**Description**
wParam	Specifies the code returned by WM_NCHITTEST.
lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates.
	The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.167.2 Returns

An application should return zero if it processes this message.

D.167.3 Cross-References

WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONUP, WM_SYSCOMMAND, **POINT**

D.168 WM_NCLBUTTONUP

D.168.1 Description

The WM_NCLBUTTONUP message is sent to a window when the user releases the left mouse button while the cursor is within a non-client area of the window.

Parameter	**Description**
wParam	Specifies the code returned by WM_NCHITTEST.
lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates.

The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.168.2 Returns

An application should return zero if it processes this message.

D.168.3 Cross-References

WM_NCHITTEST, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_SYSCOMMAND

D.169 WM_NCMBUTTONDBLCLK

D.169.1 Description

The WM_NCMBUTTONDBLCLK message is sent to a window when the user double-clicks the middle mouse button while the cursor is within a non-client area of the window.

Parameter	Description
wParam	Specifies the code returned by WM_NCHITTEST.
lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates.
	The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.169.2 Returns

An application should return zero if it processes this message.

D.169.3 Cross-References

WM_NCHITTEST, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, **POINT**

D.170 WM_NCMBUTTONDOWN

D.170.1 Description

The WM_NCMBUTTONDOWN message is sent to a window when the user double-clicks the middle mouse button while the cursor is within a non-client area of the window.

Parameter	Description
wParam	Specifies the code returned by WM_NCHITTEST.
lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates.
	The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.170.2 Returns

An application should return zero if it processes this message.

D.170.3 Cross-References

WM_NCHITTEST, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONUP

D.171 WM_NCMBUTTONUP

D.171.1 Description

The WM_NCMBUTTONUP message is sent to a window when the user presses the middle mouse button while the cursor is within a non-client area of the window.

Parameter	Description

wParam	Specifies the code returned by WM_NCHITTEST.
lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates. The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.171.2 Returns

An application should return zero if it processes this message.

D.171.3 Cross-References

WM_NCHITTEST, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN

D.172 WM_NCMOUSEMOVE

D.172.1 Description

The WM_NCMOUSEMOVE message is sent to a window when the cursor is moved within a non-client area of the window.

Parameter	**Description**
wParam	Specifies the code returned by WM_NCHITTEST.
lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates. The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.172.2 Returns

An application should return zero if it processes this message.

D.172.3 Cross-References

WM_NCHITTEST, WM_SYSCOMMAND, **POINT**

D.173 WM_NCPAINT

D.173.1 Description

The WM_NCPAINT message is sent to a window when its frame needs painting.

This message has no parameters.

The *DefWindowProc()* function paints the window frame. An application can intercept this message and paint its own custom window frame. The clipping region for a window is always rectangular, even if the shape of the frame is altered.

D.173.2 Returns

An application should return zero if it processes this message.

D.173.3 Cross-References

DefWindowProc()

D.174 WM_NCRBUTTONDBLCLK

D.174.1 Description

The WM_NCRBUTTONDBLCLK message is sent to a window when the user double-clicks the right mouse button while the cursor is within a non-client area of the window.

Parameter	**Description**
wParam	Specifies the code returned by WM_NCHITTEST.

lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates. The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.174.2 Returns

An application should return zero if it processes this message.

D.174.3 Cross-References

WM_NCHITTEST, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, **POINT**

D.175 WM_NCRBUTTONDOWN

D.175.1 Description

The WM_NCRBUTTONDOWN message is sent to a window when the user presses the right mouse button while the cursor is within a non-client area of the window.

Parameter	Description
wParam	Specifies the code returned by WM_NCHITTEST.
lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates. The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.175.2 Returns

An application should return zero if it processes this message.

D.175.3 Cross-References

WM_NCHITTEST, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONUP, **POINT**

D.176 WM_NCRBUTTONUP

D.176.1 Description

The WM_NCRBUTTONUP message is sent to a window when the user releases the right mouse button while the cursor is within a non-client area of the window.

Parameter	Description
wParam	Specifies the code returned by WM_NCHITTEST.
lParam	The low-order word of *lParam* specifies the x-coordinate of the cursor, in screen coordinates. The high-order word of *lParam* specifies the y-coordinate of the cursor, in screen coordinates.

If appropriate, WM_SYSCOMMAND messages are sent.

D.176.2 Returns

An application should return zero if it processes this message.

D.176.3 Cross-References

WM_NCHITTEST, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, **POINT**

D.177 WM_NEXTDLGCTL

D.177.1 Description

An application sends the WM_NEXTDLGCTL message to a dialog box procedure to set the focus to a different control in a dialog box.

Parameter	Description
wParam	If the value of the *wParam* parameter is non-zero, the *wParam* parameter is the handle of the control that receives the focus. If the low-order word of *lParam* is zero, *wParam* is a flag that indicates whether the next or previous control with the WS_TABSTOP style receives the focus. If *wParam* is zero, the next control receives the focus. Otherwise, the previous control with the WS_TABSTOP style receives the focus.
lParam	The low-order word of *lParam* indicates how Windows uses the *wParam* parameter. If the low-order word of *lParam* is non-zero, *wParam* is a handle associated with the control that receives the focus. Otherwise, *wParam* is a flag that indicates whether the next or previous control with the WS_TABSTOP style receives the focus.

The effect of this message differs from that of the *SetFocus()* function because WM_NEXTDLGCTL modifies the border around the default button. Do not use the *SendMessage()* function to send a WM_NEXTDLGCTL message if your application will concurrently process other messages that set the control focus. In this case, use the *PostMessage()* function instead.

D.177.2 Returns

An application should return zero if it processes this message.

D.177.3 Cross-References

PostMessage(), SendMessage(), SetFocus()

D.178 WM_PAINT

D.178.1 Description

The WM_PAINT message is sent when Windows or an application makes a request to repaint a portion of an application's window. The message is sent when the *UpdateWindow()* or *RedrawWindow()* function is called or by the *DispatchMessage()* function when the application obtains a WM_PAINT message by using the *GetMessage()* or *PeekMessage()* function.

This message has no parameters.

The *DispatchMessage()* function sends this message when there are no other messages in the application's message queue.

A window may receive internal paint messages as a result of calling the *RedrawWindow()* function with the RDW_INTERNALPAINT flag set. In this case, the window cannot have an update region. An application should call the *GetUpdateRect()* function to determine whether the window has an update region. If *GetUpdateRect()* returns zero, the application should not call the *BeginPaint()* and *EndPaint()* functions. A WM_PAINT message may have been caused by both an invalid area and a call to the *RedrawWindow()* function with the RDW_INTERNALPAINT flag set. For this reason, an application must check each WM_PAINT message for any necessary internal repainting or updating by looking at its internal data structures. An internal WM_PAINT message is sent only once by Windows. After an internal WM_PAINT message is returned from the *GetMessage()* or *PeekMessage()* function or is sent to a window by the *UpdateWindow()* function, no further WM_PAINT messages are sent or posted until the window is invalidated or until the *RedrawWindow()* function is called again with the RDW_INTERNALPAINT flag set.

D.178.2 Returns

An application should return zero if it processes this message.

D.178.3 Cross-References

BeginPaint(), DispatchMessage(), EndPaint(), GetMessage(), PeekMessage(), RedrawWindow(), UpdateWindow()

D.179 WM_PAINTCLIPBOARD

D.179.1 Description

The WM_PAINTCLIPBOARD message is sent by a clipboard viewer to the clipboard owner when the owner has placed data on the clipboard in the CF_OWNERDISPLAY format and the clipboard viewer's client area needs repainting.

Parameter	Description
wParam	Specifies a handle to the clipboard viewer window.
lParam	The low-order word of *lParam* points to a **PAINTSTRUCT** data structure that defines which part of the client area to paint.

To determine whether the entire client area or just a portion of it needs repainting, the clipboard owner must compare the dimensions of the drawing area given in the **rcPaint** member of the **PAINTSTRUCT** structure to the dimensions given in the most recent WM_SIZECLIPBOARD message.

An application must use the *GlobalLock()* function to lock the memory that contains the **PAINTSTRUCT** data structure. The application should unlock that memory by using the *GlobalUnlock()* function before it yields or returns control.

D.179.2 Returns

An application should return zero if it processes this message.

D.179.3 Cross-References

GlobalLock(), GlobalUnlock(), WM_SIZECLIPBOARD, **PAINTSTRUCT**

D.180 WM_PALETTECHANGED

D.180.1 Description

The WM_PALETTECHANGED message is sent to all top-level and overlapped windows after the window with the input focus has realized its logical palette, thereby changing the system palette. This message allows a window without the input focus that uses a color palette to realize its logical palette and update its client area.

Parameter	Description
wParam	Specifies the handle of the window that caused the system palette to change.
lParam	Not used. Must be set to zero.

In addition to being sent to all top-level and overlapped windows, this message is also sent to the window that changed the system palette and caused this message to be sent. If any child windows use a color palette, this message must be passed on to them. To avoid an infinite loop, a window that receives this message should not realize its palette unless it determines that *wParam* does not contain its own window handle.

D.180.2 Returns

An application should return zero if it processes this message.

D.180.3 Cross-References

WM_PALETTEISCHANGING, WM_QUERYNEWPALETTE, *RealizePalette()*

D.181 WM_PALETTEISCHANGING

D.181.1 Description

The WM_PALETTEISCHANGING message informs applications that an application is going to realize its logical palette.

Parameter	Description
wParam	Specifies the handle of the window that is going to realize its logical palette.
lParam	Not used. Must be set to zero.

D.181.2 Returns

An application should return zero if it processes this message.

D.181.3 Cross-References

WM_PALETTECHANGED, WM_QUERYNEWPALETTE

D.182 WM_PARENTNOTIFY

D.182.1 Description

The WM_PARENTNOTIFY message is sent to the parent of a child window when the child window is created or destroyed, or when the user clicks a mouse button while the cursor is over the child window. When the child window is being created, the system sends WM_PARENTNOTIFY just before the *CreateWindow()* or *CreateWindowEx()* function that creates the window returns. When the child window is destroyed, the system sends the message before any processing to destroy the window takes place.

Parameter	**Description**
wParam	Specifies the event for which the parent is being notified. It can be any of the following values:
lParam	If the low-order word of *lParam* is WM_CREATE or WM_DESTROY, this parameter specifies the handle of the child window. Otherwise, it specifies the x-coordinate of the cursor.

Value	**Description**
WM_CREATE	The child window will be created.
WM_DESTROY	The child window will be destroyed.
WM_LBUTTONDOWN	The user has placed the mouse cursor over the child window and clicked the left mouse button.
WM_MBUTTONDOWN	The user has placed the mouse cursor over the child window and clicked the middle mouse button.
WM_RBUTTONDOWN	The user has placed the mouse cursor over the child window and clicked the right mouse button.

If the high-order word of *lParam* is WM_CREATE or WM_DESTROY, this parameter specifies the identifier of the child window. Otherwise, it specifies the y-coordinate of the cursor.

This message is also sent to all ancestor windows of the child window, including the top-level window. All child windows except those that have the WS_EX_NOPARENTNOTIFY send this message to their parent windows. By default, child windows in a dialog box have the WS_EX_NOPARENTNOTIFY style unless the *CreateWindowEx()* function was called to create the child window without this style.

D.182.2 Returns

An application should return zero if it processes this message.

D.182.3 Cross-References

WM_CREATE, WM_DESTROY, WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN

D.183 WM_PASTE

D.183.1 Description

An application sends the WM_PASTE message to an edit control or combo box to insert the data from the clipboard into the edit control at the current cursor position. Data is inserted only if the clipboard contains data in CF_TEXT format.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.183.2 Returns

The return value is non-zero if this message is sent to an edit control or a combo box.

D.183.3 Cross-References

WM_CLEAR, WM_COPY, WM_CUT

D.184 WM_POWER

D.184.1 Description

The WM_POWER message is sent when the system, typically a battery-powered personal computer, is about to enter the suspended mode.

Parameter	Description
wParam	Specifies a power-event notification message. This parameter may be one of the following values:

Value	Meaning
PWR_SUSPENDREQUEST	Indicates that the system is about to enter the suspended mode.
PWR_SUSPENDRESUME	Indicates that the system is resuming operation after entering the suspended mode normally – that is, the system sent a PWR_SUSPENDREQUEST notification message to the application before the system was suspended. An application should perform any necessary recovery actions.
PWR_CRITICALRESUME	Indicates that the system is resuming operation after entering the suspended mode without first sending a PWR_SUSPENDREQUEST notification message to the application. An application should perform any necessary recovery actions.

lParam	Not used. Must be set to zero.

This message is sent only to an application that is running on a system that conforms to the advanced power management (APM) basic input-and-output system (BIOS) specification. The message is sent by the power-management driver to each window returned by the *EnumWindows()* function.

The suspended mode is the state in which the greatest amount of power savings occurs, but all operational data and parameters are preserved. Random-access memory (RAM) contents are preserved, but many devices are likely to be turned off.

D.184.2 Returns

The value an application returns depends on the value of the *wParam* parameter, which may be one of the following:

PWR_SUSPENDREQUEST	PWR_FAIL to prevent the system from entering the suspended state. Otherwise, the value is PWR_OK.
PWR_SUSPENDRESUME	0
PWR_CRITICALRESUME	0

D.184.3 Cross-References

EnumWindows()

D.185 WM_QUERYDRAGICON

D.185.1 Description

The WM_QUERYDRAGICON message is sent to a minimized (iconic) window that does not have an icon defined for its class. The system sends this message whenever it needs to display an icon for the window.

This message has no parameters.

If an application returns the handle of an icon or cursor, the system converts it to black-and-white. The application can call the *LoadCursor()* or *LoadIcon()* functions to load a cursor or icon from the resources in its executable file and to obtain this handle.

D.185.2 Returns

An application should return a double-word value that contains a cursor or icon handle in the low-order word. The cursor or icon must be compatible with the display driver's resolution. If the application returns NULL, the system displays the default cursor. The default return value is NULL.

D.185.3 Cross-References

LoadCursor(), LoadIcon()

D.186 WM_QUERYENDSESSION

D.186.1 Description

The WM_QUERYENDSESSION message is sent when the user chooses to end the Windows session, or when an application calls the *ExitWindows()* function. If any application returns zero, the Windows session is not ended. Windows stops sending WM_QUERYENDSESSION messages as soon as one application returns zero, and sends WM_ENDSESSION messages, with the wParam parameter set to FALSE, to any applications that have already returned non-zero.

This message has no parameters.

The *DefWindowProc()* function returns non-zero when it processes this message.

D.186.2 Returns

An application should return non-zero if it can conveniently terminate. Otherwise, it should return zero.

D.186.3 Cross-References

DefWindowProc(), ExitWindows(), WM_ENDSESSION

D.187 WM_QUERYNEWPALETTE

D.187.1 Description

The WM_QUERYNEWPALETTE message informs an application that it is about to receive the input focus, giving the application an opportunity to realize its logical palette when it receives the focus.

This message has no parameters.

D.187.2 Returns

An application should return non-zero if it realizes its logical palette. Otherwise, it should return zero.

D.187.3 Cross-References

WM_PALETTECHANGED, WM_PALETTEISCHANGING

D.188 WM_QUERYOPEN

D.188.1 Description

The WM_QUERYOPEN message is sent to a minimized window when the user requests that the window be restored to its preminimized size and position.

This message has no parameters.

While processing this message, the application should not perform any action that would cause an activation or focus change. The *DefWindowProc()* function returns non-zero when it processes this message.

D.188.2 Returns

An application that processes this message should return a non-zero value if the icon can be opened, or zero to prevent the icon from being opened.

D.188.3 Cross-References

DefWindowProc()

D.189 WM_QUIT

D.189.1 Description

The WM_QUIT message indicates a request to terminate an application and is generated when the application calls the *PostQuitMessage()* function. It causes the *GetMessage()* function to return zero.

Parameter	Description
wParam	Specifies the exit code given in the *PostQuitMessage()* function.
lParam	Not used. Must be set to zero.

D.189.2 Returns

This message does not have a return value, because it causes the message loop to terminate before the message is sent to the application's window procedure.

D.189.3 Cross-References

GetMessage(), PostQuitMessage()

D.190 WM_RBUTTONDBLCLK

D.190.1 Description

The WM_RBUTTONDBLCLK message is sent when the user double-clicks the right mouse button.

Parameter	Description	
wParam	Key status, which can be one or more of the following values:	
	MK_CONTROL	The CTRL key is pressed.
	MK_LBUTTON	The left button is pressed.
	MK_MBUTTON	The middle button is pressed.
	MK_RBUTTON	The right button is pressed.
	MK_SHIFT	The SHIFT key is pressed.
lParam	LOWORD is the horizontal position and HIWORD is the vertical position.	

Note: Only windows whose window class has the CS_DBLCLKS style receives double-click messages. Double-clicks are generated when the user presses and releases the mouse twice within the system's time limit. A double-click generates the following sequence of messages: WM_RBUTTONDOWN, WM_RBUTTONUP, WM_RBUTTONDBLCLK, followed by another WM_RBUTTONUP.

D.190.2 Returns

If the application processes this message, it should return zero.

D.190.3 Cross-References

WM_RBUTTONDOWN, WM_RBUTTONUP

D.191 WM_RBUTTONDOWN

D.191.1 Description

The WM_RBUTTONDOWN message is sent when the user presses the right mouse button.

Parameter	Description	
wParam	Key status, which can be one or more of the following values:	
	MK_CONTROL	The CTRL key is pressed.
	MK_LBUTTON	The left button is pressed.
	MK_MBUTTON	The middle button is pressed.
	MK_SHIFT	The SHIFT key is pressed.
lParam	LOWORD is the horizontal position and HIWORD is the vertical position.	

D.191.2 Returns

If the application processes this message, it should return zero.

D.191.3 Cross-References

WM_RBUTTONDBLCLK, WM_RBUTTONUP

D.192 WM_RBUTTONUP

D.192.1 Description

The WM_RBUTTONUP message is sent when the user releases the right mouse button.

Parameter	Description	
wParam	Key status, which can be one or more of the following values:	
	MK_CONTROL	The CTRL key is pressed.
	MK_LBUTTON	The left button is pressed.
	MK_MBUTTON	The middle button is pressed.
	MK_SHIFT	The SHIFT key is pressed.
lParam	LOWORD is the horizontal position and HIWORD is the vertical position.	

D.192.2 Returns

If the application processes this message, it should return zero.

D.192.3 Cross-References

WM_RBUTTONDBLCLK, WM_RBUTTONDOWN

D.193 WM_RENDERALLFORMATS

D.193.1 Description

The WM_RENDERALLFORMATS message is sent to the clipboard owner when the owner application is being destroyed.

Parameter	Description
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

Each clipboard owner should pass a data handle to *SetClipboardData()* for each format it supports, thereby ensuring valid data even though the application is being destroyed.

D.193.2 Returns

If the application processes this message, it should return zero.

D.193.2 Cross-References

SetClipboardData(), WM_RENDERFORMAT

D.194 WM_RENDERFORMAT

D.194.1 Description

The WM_RENDERFORMAT message is sent to the clipboard owner when a particular data format needs to be rendered.

Parameter	Description
wParam	Clipboard data format.
lParam	Not used. Must be set to zero.

To process this message, data must be rendered using *SetClipboardData()* for the particular data type.

Note: During processing, the application should not call *OpenClipboard()* or *CloseClipboard().*

D.194.2 Returns

If the application processes this message, it should return zero.

D.194.3 Cross-References

SetClipboardData(), OpenClipboard(), CloseClipboard(), WM_RENDERFORMAT

D.195 WM_SETCURSOR

D.195.1 Description

The WM_SETCURSOR message is sent when the mouse causes cursor movement within a window and the mouse input is not captured.

Parameter	Description
wParam	The window that contains the cursor.
lParam	LOWORD is the hit-test area code, and HIWORD is the number of the mouse message.

When used by *DefWindowProc(),* WM_SETCURSOR is sent to the parent window before processing begins. This allows the parent window an opportunity to control the cursor's settings within a child window. If the application returns TRUE, processing is stopped.

DefWindowProc() uses this message to set the cursor to a pointer if it is not in the client area, or to set the cursor as registered for the class of the window if it is within the client area.

When a dialog box is going to set the cursor for one of its child window controls, it must force *DefDlgProc()* to return TRUE when processing WM_SETCURSOR. For the standard dialog box class, *DefDlgProc()* provides default processing. A dialog box procedure can return TRUE when processing the WM_SETCURSOR message by using *SetWindowLong()* and the DWL_MSGRESULT offset.

Note: If the hit-test code is HTERROR and the mouse message is a button-down message, it means that *MessageBeep()* was called.

D.195.2 Returns

TRUE stops further processing, while FALSE allows processing to continue.

D.195.3 Cross-References

DefWindowProc(), MessageBeep(), SetWindowLong()

D.196 WM_SETFOCUS

D.196.1 Description

The WM_SETFOCUS message is sent when a window has just gained focus.

Parameter	Description
wParam	The window that lost the focus.

lParam	Not used. Must be set to zero.

D.196.2 Returns

If the application processes this message, it should return zero.

D.196.3 Cross-References

SetFocus()

D.197 WM_SETFONT

D.197.1 Description

The WM_SETFONT message is sent by an application to a control to tell the control what font to use when drawing.

Parameter	Description
wParam	Handle to the font to be used.
lParam	LOWORD is TRUE if the control should be redrawn. The HIWORD is not used.

This message applies to dialog box controls, as well as other controls. When setting a new font, the old font should be deleted using *DeleteObject().* The control is not resized by changing the font. The control should resize before drawing. A dialog box with the DS_SETFONT style that is created using *CreateDialogIndirect(), CreateDialogIndirectParam(), DialogBoxIndirect(),* or *DialogBoxIndirectParam()* is sent a WM_SETFONT message.

D.197.2 Returns

If the application processes this message, it should return zero.

D.197.3 Cross-References

DeleteObject(), CreateDialogIndirect(), CreateDialogIndirectParam(), DialogBoxIndirect(), DialogBoxIndirectParam()

D.198 WM_SETREDRAW

D.198.1 Description

The WM_SETREDRAW message is sent to a window to allow changes to be drawn or to prevent the drawing of changes.

Parameter	Description
wParam	Redraw flag.
lParam	Not used. Must be 0L.

This message is used most often when several processing steps are anticipated, which would cause the window to draw and then redraw itself. This appears as flickering to the user. To avoid this condition, an application sends a WM_SETREDRAW message, where *wParam* is FALSE to ensure that changes that would affect the display of that window will not generate messages telling the window to redraw itself. Once the processing is complete, the application sends another WM_SETREDRAW message, except where *wParam* is TRUE. The message in itself does not cause the window to be drawn. To cause the window to be drawn, the application should call *InvalidateRect().*

D.198.2 Returns

If the application processes this message, it should return zero.

D.198.3 Cross-References

InvalidateRect()

D.199 WM_SETTEXT

D.199.1 Description

The WM_SETTEXT message is sent to a window to set its text.

Parameter	Description
wParam	Not used. Must be 0L.
lParam	Pointer to NULL terminated text string.

Note: For a combo or list box, setting the text does not change the selection.

D.199.2 Returns

The application returns LB_ERRSPACE or CB_ERRSPACE if there is insufficient space in a list box or combo box respectively, or CB_ERR if a combo box has no edit control.

D.199.3 Cross-References

SetWindowText(), WM_GETTEXT

D.200 WM_SHOWWINDOW

D.200.1 Description

The WM_SHOWWINDOW message is sent to a window when it is going to be shown or hidden.

Parameter	Description
wParam	Flag to indicate if the window is to be shown.
lParam	Status.

If the window is an overlapped window and it is going to be minimized, all of its pop-up windows are hidden. Conversely, if it is maximized or restored, then the pop-up windows are shown. If the status is zero, the message is due to a *ShowWindow()* function call. Otherwise, it is due to the receipt of a SW_PARENTCLOSING or SW_PARENTOPENING message that indicates the action of the parent window.

Note: A. WM_SHOWWINDOW message is not generated when the main window has either WS_MINIMIZE or WS_MAXIMIZE styles or *ShowWindow()* was called with SW_SHOWNORMAL.

D.200.2 Returns

If the application processes this message, it should return zero.

D.200.3 Cross-References

ShowWindow()

D.201 WM_SIZE

D.201.1 Description

The WM_SIZE message is sent to a window after its size has changed.

Parameter	Description	
wParam	Sizing status, which can be one of the following values:	
	SIZE_MAXIMIZED	The window was maximized.
	SIZE_MINIMIZED	The window was minimized.
	SIZE_RESTORED	The window was resized but not maximized or minimized.
	SIZE_MAXHIDE	Sent to pop-up windows to be hidden due to another window being maximized.
	SIZE_MAXSHOW	Sent to pop-up windows to be shown due to another window being restored.
lParam	LOWORD is width and HIWORD is height.	

Note: If a WM_SIZE message is received causing *SetScrollPos()* or *MoveWindow()* to be called for a child window, the repaint parameter should be TRUE (non-zero) so the window is repainted.

D.201.2 Returns

If the application processes this message, it should return zero.

D.201.3 Cross-References

SetScrollPos(), MoveWindow()

D.202 WM_SIZECLIPBOARD

D.202.1 Description

The WM_SIZECLIPBOARD message is sent to a clipboard owner of CF_OWNERDISPLAY data when the clipboard viewer's client area is resized.

Parameter	**Description**
wParam	Window handle of clipboard viewer.
lParam	Handle of global object.

The global object is a **RECT**. If the **RECT** is at location zero and of size zero, then the view will be minimized or destroyed.

D.202.2 Returns

If the application processes this message, it should return zero.

D.202.3 Cross-References

SetClipboardData(), SetClipboardViewer()

D.203 WM_SPOOLERSTATUS

D.203.1 Description

The WM_SPOOLERSTATUS message is sent by the printer manager whenever the print queue size changes.

Parameter	**Description**
wParam	Print job status.
lParam	Number of jobs in the queue.

The status indicates the SP_JOBSTATUS flag.

D.203.2 Returns

If the application processes this message, it should return zero.

D.203.3 Cross-References

SP_JOBSTATUS

D.204 WM_SYSCHAR

D.204.1 Description

The WM_SYSCHAR message is sent to the window with input focus when WM_SYSKEYDOWN and WM_SYSKEYUP messages are translated.

Parameter	**Description**	
wParam	Virtual key code.	
lParam	Key data:	
	Bits 0-15	Specify the repeat count.
	Bits 16-23	Specify the manufacturer’s scan code.
	Bit 24	Specifies whether the key was an extended key.

Bits 25-26	Not used.
Bits 27-28	Used internally by the OS.
Bit 29	Context code that indicates if the ALT key was pressed.
Bit 30	Indicates the previous state of the key. It is set if the keywas down before the message was sent, or clear if the keywas up.
Bit 31	Indicates the transition status. It is set if the key is being released, or clear if it is being pressed.

The virtual key code is the one for a system menu key. If bit 29 is zero, *TranslateAccelerator()* can handle the message as though it were a normal key message, instead of one for the system menu. In this way, accelerator keys can be used by the active window, even though it does not have input focus.

D.204.2 Returns

If the application processes this message, it should return zero.

D.204.3 Cross-References

WM_SYSKEYDOWN, WM_SYSKEYUP, *TranslateAccelerator()*

D.205 WM_SYSCOLORCHANGE

D.205.1 Description

A WM_SYSCOLORCHANGE message is sent to all top-level windows after a system color change is made.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	Not used. Must be set to zero.

D.205.2 Returns

If the application processes this message, it should return zero.

D.205.3 Cross-References

WM_PAINT

D.206 WM_SYSCOMMAND

D.206.1 Description

A WM_SYSCOMMAND message is sent when a system menu item is selected. The message is also sent when the minimize or maximize buttons are pressed.

Parameter	**Description**	
wParam	Specifies the selected command and is one of the following values:	
	SC_CLOSE	Close window.
	SC_HOTKEY	Activate a window associated with the hot key.
	SC_HSCROLL	Horizontal scroll.
	SC_VSCROLL	Vertical scroll.
	SC_KEYMENU	Get a menu through a keystroke.
	SC_MAXIMZE	Maximize the window.
	SC_ZOOM	Same as SC_MAXIMIZE.
	SC_MINIMIZE	Minimize the window.
	SC_ICON	Same as SC_MINIMIZE.
	SC_MOUSEMENU	Get a menu through a mouse click.

	SC_MOVE	Move the window.
	SC_NEXTWINDOW	Select the next window.
	SC_PREVWINDOW	Select the previous window.
	SC_RESTORE	Restore window to its normal size and location.
	SC_SCREENSAVE	Execute the screen saver application.
	SC_SIZE	Size the window.
	SC_TASKLIST	Execute the Task Manager application.
lParam	The low-order word contains the x-coordinate if the system menu was chosen with the mouse. If the message is SC_HOTKEY, the low-order word identifies the window to activate. Otherwise, it is unused. The high-order word contains the y-coordinate if the system menu was chosen with the mouse. Otherwise, it is unused.	

The four low-order bits of *wParam* are reserved and must be masked of using the value 0xFFF0 for the results to be interpreted correctly.

D.206.2 Returns

If the application processes this message, it should return zero.

D.206.3 Cross-References

WM_COMMAND

D.207 WM_SYSDEADCHAR

D.207.1 Description

A WM_SYSDEADCHAR message is sent to the window with focus whenever the WM_SYSKEYDOWN or WM_SYSKEYUP messages are translated to specify the dead key character.

Parameter	**Description**
wParam	Specifies the dead key character.
lParam	The low-order word indicates the repeat count. The high-order word indicates the auto repeat count.

D.207.2 Returns

If the application processes this message, it should return zero.

D.207.3 Cross-References

WM_SYSKEYDOWN, WM_SYSKEYUP

D.208 WM_SYSKEYDOWN

D.208.1 Description

A WM_SYSKEYDOWN message is sent to the window with focus whenever a key is pressed in combination with the ALT key. If no window has focus, the message is sent to the active window.

Parameter	**Description**
wParam	The virtual key code of the key pressed.
lParam	Bits 0-15 specify the repeat count.
	Bit 24 is set if the key is extended.
	Bits 25-26 are unused.
	Bits 27-28 are reserved by the system.
	Bit 29 is set if the ALT key was held down. Otherwise, it indicates that the message was sent to the active window because no windows had focus.
	Bit 30 is set if the key was down before the message was sent. Otherwise, it was up.

Bit 31 is unused for the WM_SYSKEYDOWN message.

D.208.2 Returns

If the application processes this message, it should return zero.

D.208.3 Cross-References

WM_SYSKEYUP

D.209 WM_SYSKEYUP

D.209.1 Description

A WM_SYSKEYUP message is sent to the window with focus whenever a key is pressed in combination with the ALT key. If no window has focus, the message is sent to the active window.

Parameter	Description
wParam	The virtual key code of the key pressed.
lParam	Bits 0-15 specify the repeat count. Bits 16-23 specify the scan code. Bit 24 is set if the key is extended. Bits 25-26 are unused. Bits 27-28 are reserved by the system. Bit 29 is set if the ALT key was held down. Otherwise, it indicates that the message was sent to the active window because no windows had focus. Bit 30 is set if the key was down before the message was sent. Otherwise, it was up. Bit 31 is set if the key is being released; otherwise, it is being pressed.

D.209.2 Returns

If the application processes this message, it should return zero.

D.209.3 Cross-References

WM_SYSKEYDOWN

D.210 WM_TIMER

D.210.1 Description

A WM_TIMER message is sent to an application's message queue or an installed *TimerProc()* callback function after the specified timer interval is reached.

Parameter	Description
wParam	Specifies the identifier of the timer.
lParam	A pointer to a callback function that was passed to the *SetTimer()* function when the timer was installed. If *lParam* is not NULL, the callback function is called, as opposed to posting to the application's message queue.

D.210.2 Returns

If the application processes this message, it should return zero.

D.210.3 Cross-References

None.

D.211 WM_UNDO

D.211.1 Description

A WM_UNDO message is sent to an edit control to instruct it to undo the previous action.

Parameter	Description
wParam	Unused. Must be set to zero.
lParam	Unused. Must be set to zero.

D.211.2 Returns

The message returns TRUE if successful. If an error occurs, FALSE is returned.

D.211.3 Cross-References

WM_CLEAR, WM_COPY, WM_CUT, WM_PASTE

D.212 WM_VKEYTOITEM

D.212.1 Description

A WM_VKEYTOITEM message is sent by a list box to its owner after it receives a WM_KEYDOWN message. The WM_VKEYTOITEM is only sent by a list box that has the LBS_WANTKEYBOARDINPUT style.

Parameter	Description
wParam	Specifies the virtual key code.
lParam	The low-order word identifies the list box. The high-order word specifies the current location of the cursor.

The list box must have the LBS_HASSTRINGS style to receive this message.

D.212.2 Returns

The application returns -2 if it handled all aspects of the selecting item. It returns a -1 if the list box needs to perform the default action. It returns zero or greater if the item in the list box should perform the default action for the key on the specified item.

D.212.3 Cross-References

WM_KEYDOWN, WM_CHARTOITEM

D.213 WM_VSCROLL

D.213.1 Description

A WM_VSCROLL message is sent when the vertical scroll bar has been clicked.

Parameter	Description	
wParam	Specifies the scroll bar code and is one of the following.	
	SB_BOTTOM	Scroll to bottom.
	SB_TOP	Scroll to top.
	SB_ENDSCROLL	Scroll to end.
	SB_LINEDOWN	Scroll down one line.
	SB_LINEUP	Scroll up one line.
	SB_PAGEDOWN	Scroll down one page.
	SB_PAGEUP	Scroll up one page.
	SB_THUMBPOSITION	Scroll to a position specified in *lParam*.
	SB_THUMBTRACK	Move scroll box thumb to the position specified in *lParam*.
lParam	The low-order word specifies the position of the scroll box for the SB_THUMBPOSITION and SB_THUMBTRACK scroll bar codes. The high-order word specifies the control if VM_VSCROLL is the scroll bar code.	

D.213.2 Returns

The application should return zero if it processes the message.

D.213.3 Cross-References

WM_HSCROLL

D.214 WM_VSCROLLCLIPBOARD

D.214.1 Description

A WM_VSCROLLCLIPBOARD message is sent by the clipboard viewer to the clipboard owner for clipboard image scrolling and updating. The message is only sent to the owner if the clipboard data had the CF_OWNERDISPLAY format.

Parameter	**Description**
wParam	Specifies the clipboard viewer's handle.
lParam	The high-order word specifies the position of the scroll box for the SB_THUMBPOSITION scroll bar code. The low-order word specifies the scroll bar code and is one of the following:

SB_BOTTOM	Scroll to the lower right.
SB_TOP	Scroll to the upper left.
SB_ENDSCROLL	Scroll to end.
SB_LINEDOWN	Scroll down one line.
SB_LINEUP	Scroll up one line.
SB_PAGEDOWN	Scroll down one page.
SB_PAGEUP	Scroll up one page.
SB_THUMBPOSITION	Scroll to a position specified in *lParam*.

D.214.2 Returns

The application should return zero if it processes the message.

D.214.3 Cross-References

WM_HSCROLLCLIPBOARD

D.215 WM_WINDOWPOSCHANGED

D.215.1 Description

A WM_WINDOWPOSCHANGED message is sent to a window whose position or size has changed.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	A pointer to a **WINDOWPOS** structure containing information about the new size and position of the window.

D.215.2 Returns

The application should return zero if it processes the message.

D.215.3 Cross-References

WM_MOVE, WM_SIZE, WM_WINDOWPOSCHANGING

D.216 WM_WINDOWPOSCHANGING

D.216.1 Description

A WM_WINDOWPOSCHANGING message is sent to a window whose position or size is about to be changed.

Parameter	**Description**
wParam	Not used. Must be set to zero.

lParam	A pointer to a WINDOWPOS structure containing information about the new size and position of the window.

The fields in the **WINDOWPOS** structure can be modified to affect the windows size and position.

D.216.2 Returns

The application should return zero if it processes the message.

D.216.3 Cross-References

WM_WINDOWPOSCHANGED

D.217 WM_WININICHANGE

D.217.1 Description

An application sends the WM_WININICHANGE message after making a change to the WIN.INI file.

Parameter	**Description**
wParam	Not used. Must be set to zero.
lParam	A pointer to a string with the name of the section that was changed. If multiple sections were changed, the parameter is NULL.

D.217.2 Returns

The application should return zero if it processes the message.

D.217.3 Cross-References

WM_WINDOWPOSCHANGED

Annex E

Control Notifications

Description

This annex describes control notification messages.

E.1 BN_CLICKED

E.1.1 Description

The BN_CLICKED notification message is sent to the parent window when the user clicks a button.

Parameter	**Description**
wParam	Specifies the button control identifier.
lParam	The low-order word contains the button window handle and the high-order word contains BN_CLICKED notification code.

E.1.2 Cross-References

DRAWITEMSTRUCT, WM_DRAWITEM

E.2 BN_DISABLE

E.2.1 Description

The BN_DISABLE notification message is sent to the parent window when a button is disabled. This message has no parameters.

E.2.2 Cross-References

DRAWITEMSTRUCT, WM_DRAWITEM

E.3 BN_DOUBLECLICKED

E.3.1 Description

The BN_DOUBLECLICKED notification message is sent to the parent window when the user double-clicks a button. This message has no parameters.

E.3.2 Cross-References

DRAWITEMSTRUCT, WM_DRAWITEM

E.4 BN_HILITE

E.4.1 Description

The BN_HILITE notification message is sent to the parent window when the user highlights a button. This message has no parameters.

E.4.2 Cross-References

DRAWITEMSTRUCT, WM_DRAWITEM

E.5 BN_PAINT

E.5.1 Description

The BN_PAINT notification message is sent to the parent window when a button should be painted. This message has no parameters.

E.5.2 Cross-References

DRAWITEMSTRUCT, WM_DRAWITEM

E.6 BN_UNHILITE

E.6.1 Description

The BN_UNHILITE notification message is sent to the parent window when the highlight should be removed from a button. This message has no parameters.

E.6.2 Cross-References

DRAWITEMSTRUCT, WM_DRAWITEM

E.7 CBN_CLOSEUP

E.7.1 Description

The CBN_CLOSEUP notification message is sent when the list box of a combo box is about to be hidden. It is not sent to a combo box that has the CBS_SIMPLE style, since its list box is always visible.

Parameter	**Description**
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains combo box window handle and high-order word contains CBN_CLOSEUP notification code.

E.7.2 Cross-References

CBN_DROPDOWN, CBN_SELCHANGE, WM_COMMAND

E.8 CBN_DBLCLCK

E.8.1 Description

The CBN_DBLCLK notification message is sent to the parent window when the user double-clicks a string in the list box of a combo box. This applies only to combo boxes created with CBS_SIMPLE window style.

Parameter	**Description**
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains the combo box window handle and the high-order word contains the CBN_DBLCLK notification code.

E.8.2 Cross-References

CBN_SELCHANGE, WM_COMMAND

E.9 CBN_DROPDOWN

E.9.1 Description

The CBN_DROPDOWN notification message is sent when the list box of a combo box is about to be dropped down. This applies only to combo boxes created with the CBS_DROPDOWN or CBS_DROPDOWNLIST window style.

Parameter	**Description**
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains combo box window handle and high-order word contains the CBN_DROPDOWN notification code.

E.9.2 Cross-References

CBN_CLOSEUP, WM_COMMAND

E.10 CBN_EDITCHANGE

E.10.1 Description

The CBN_EDITCHANGE notification message is sent after the user has altered the text in the edit-control portion of a combo box. Unlike the CBN_EDITUPDATE notification message, this notification message is sent after the screen is updated. This notification is not sent to a combo box created with the CBS_DROPDOWNLIST window style.

Parameter	Description
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains the combo box window handle and the high-order word contains the CBN_EDITCHANGE notification code.

E.10.2 Cross-References

CBN_EDITUPDATE, WM_COMMAND

E.11 CBN_EDITUPDATE

E.11.1 Description

The CBN_EDITUPDATE notification message is sent to the parent window when the edit-control portion of a combo box is about to display altered text. This notification is sent after the text has been formatted, but before it is displayed in a window. This notification is not sent to a combo box created with CBS_DROPDOWNLOST window style.

Parameter	**Description**
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains the combo box window handle and the high-order word contains the CBN_EDITUPDATE notification code.

E.11.2 Cross-References

CBN_EDITCHANGE, WM_COMMAND

E.12 CBN_ERRSPACE

E.12.1 Description

The CBN_ERRSPACE notification message is sent to the parent window when a combo box cannot allocate enough memory to process a request.

Parameter	**Description**
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains the combo box window handle and the high-order word contains the CBN_ERRSPACE notification code.

E.12.2 Cross-References

WM_COMMAND

E.13 CBN_KILLFOCUS

E.13.1 Description

The CBN_KILLFOCUS notification message is sent to the parent window when a combo box loses the input focus.

Parameter	**Description**
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains the combo box window handle and the high-order word contains the CBN_KILLFOCUS notification code.

E.13.2 Cross-References

CBN_SETFOCUS, WM_COMMAND

E.14 CBN_SELCHANGE

E.14.1 Description

The CBN_SELCHANGE notification message is sent to the parent window when the selection in the list box of a combo box is about to be changed as a result of the user either clicking in the list box or changing the selection by using the arrow keys.

Parameter	**Description**
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains the combo box window handle and the high-order word contains the CBN_SELCHANGE notification code.

E.14.2 Cross-References

CBN_DBLCLK, CB_SETCURSEL, WM_COMMAND

E.15 CBN_SELENDCANCEL

E.15.1 Description

The CBN_SELENDCANCEL notification message is sent to the parent window when the user clicks an item, then clicks somewhere else and the list box of a combo box gets hidden. This notification message is sent before the CBN_CLOSEUP notification message and indicates that the user's selection should be ignored. It is sent always, even if the combo box has the CBS_SIMPLE window style.

Parameter	**Description**
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains the combo box window handle and the high-order word contains the CBN_SELENDCANCEL notification code.

E.15.2 Cross-References

CBN_SELENDOK, WM_COMMAND

E.16 CBN_SELENDOK

E.16.1 Description

The CBN_SELENDOK notification message is sent to the parent window when the user selects an item and then presses the ENTER or the DOWN ARROW key to hide the list box of a combo box. This notification message is sent before the CBN_CLOSEUP notification message to indicate that the user's selection should be considered valid. It is sent always, even if the combo box has the CBS_SIMPLE window style.

Parameter	**Description**
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains the combo box window handle and the high-order word contains the CBN_SELENDOK notification code.

E.16.2 Cross-References

CBN_SELENDCANCEL, WM_COMMAND

E.17 CBN_SETFOCUS

E.17.1 Description

The CBN_SETFOCUS notification message is sent to the parent window when a combo box receives the input focus.

Parameter	Description
wParam	Specifies the combo box control identifier.
lParam	The low-order word contains the combo box window handle and the high-order word contains the CBN_SETFOCUS notification code.

E.17.2 Cross-References

CBN_KILLFOCUS, WM_COMMAND

E.18 EN_CHANGE

E.18.1 Description

The EN_CHANGE notification message is sent to the parent window when the user has altered text in an edit control. Unlike the EN_UPDATE notification message, this notification message is sent after the edit control is updated on the screen.

Parameter	Description
wParam	Specifies the edit window identifier.
lParam	The low-order word contains the edit window handle and the high-order word contains the EN_CHANGE notification code.

E.18.2 Cross-References

EN_UPDATE, WM_COMMAND

E.19 EN_ERRSPACE

E.19.1 Description

The EN_ERRSPACE notification message is sent to the parent window when an edit control cannot allocate enough memory to process a request.

Parameter	Description
wParam	Specifies the edit window identifier.
lParam	The low-order word contains the edit window handle and the high-order word contains the EN_ERRSPACE notification code.

E.19.2 Cross-References

WM_COMMAND

E.20 EN_HSCROLL

E.20.1 Description

The parent window of an edit control is sent an EN_HSCROLL notification after the user has clicked the horizontal scroll bar. The WM_COMMAND message containing the notification is sent before the screen is updated.

Parameter	Description
wParam	Contains the edit control identifier.
lParam	Combines the EN_HSCROLL notification value in the high-order 16-bits and the 16-bit handle of the edit control in the low-order word.

E.20.2 Cross-References

EN_VSCROLL, WM_COMMAND

E.21 EN_KILLFOCUS

E.21.1 Description

The parent window of an edit control is sent an EN_KILLFOCUS notification in a WM_COMMAND message when the control loses focus.

Parameter	Description
wParam	Contains the edit control identifier.
lParam	Combines the EN_KILLFOCUS notification value in the high-order 16-bits and the 16-bit handle of the edit control in the low-order word.

E.21.2 Cross-References

EN_SETFOCUS, WM_COMMAND

E.22 EN_MAXTEXT

E.22.1 Description

The parent window of an edit control is sent an EN_MAXTEXT notification in a WM_COMMAND message after one of three conditions has occurred: 1) the current insertion exceeds the character limit of the control; 2) the current insertion exceeds the width of a control that does not have the ES_AUTOHSCROLL style; or 3) the current insertion exceeds the height of a control which does not have the ES_AUTOVSCROLL style.

Parameter	Description
wParam	Contains the edit control identifier.
lParam	Combines the EN_MAXTEXT notification value in the high-order 16-bits and the 16-bit handle of the edit control in the low-order word.

E.22.2 Cross-References

EM_LIMITTEXT, WM_COMMAND

E.23 EN_SETFOCUS

E.23.1 Description

The parent window of an edit control is sent an EN_SETFOCUS notification in a WM_COMMAND message when the control receives input focus.

Parameter	Description
wParam	Contains the edit control identifier.
lParam	Combines the EN_SETFOCUS notification value in the high-order 16-bits and the 16-bit handle of the edit control in the low-order word.

E.23.2 Cross-References

EN_KILLFOCUS, WM_COMMAND

E.24 EN_UPDATE

E.24.1 Description

The parent window of an edit control is sent an EN_UPDATE notification before a text change is displayed. The notification is sent after the text has been formatted, but before it has been displayed. This provides for the possibility of a resulting window size change.

Parameter	Description
wParam	Contains the edit control identifier.
lParam	Combines the EN_UPDATE notification value in the high-order 16-bits and the 16-bit handle of the edit control in the low-order word.

E.24.2 Cross-References

EN_CHANGE, WM_COMMAND

E.25 EN_VSCROLL

E.25.1 Description

The parent window of an edit control is sent an EN_VSCROLL notification after the user has clicked the vertical scroll bar. The WM_COMMAND message containing the notification is sent before the screen is updated.

Parameter	Description
wParam	Contains the edit control identifier.
lParam	Combines the EN_VSCROLL notification value in the high-order 16-bits and the 16-bit handle of the edit control in the low-order word.

E.25.2 Cross-References

EN_HSCROLL, WM_COMMAND

E.26 LBN_DBLCLK

E.26.1 Description

The parent window of a list box control is sent an LBN_DBLCLK notification in a WM_COMMAND message after the user has double-clicked a string in a list box. This notification is only sent if the list box control has the LBS_NOTIFY style.

Parameter	Description
wParam	Contains the list box control identifier.
lParam	Combines the LBN_DBLCLK notification value in the high-order 16-bits and the 16-bit handle of the list box control in the low-order word.

E.26.2 Cross-References

LBN_SELCHANGE, WM_COMMAND

E.27 LBN_ERRSPACE

E.27.1 Description

The parent window of a list box control is sent an LBN_ERRSPACE notification in a WM_COMMAND message when insufficient memory is available to meet the requirements of a list box operation.

Parameter	Description
wParam	Contains the list box control identifier.
lParam	Combines the LBN_ERRSPACE notification value in the high-order 16-bits and the 16-bit handle of the list box control in the low-order word.

E.27.2 Cross-References

WM_COMMAND

E.28 LBN_KILLFOCUS

E.28.1 Description

The parent window of a list box control is sent an LBN_KILLFOCUS notification in a WM_COMMAND message when the control loses focus.

Parameter	Description
wParam	Contains the list box control identifier.
lParam	Combines the LBN_KILLFOCUS notification value in the high-order 16-bits and the 16-bit handle of the list box control in the low-order word.

E.28.2 Cross-References

LBN_SETFOCUS, WM_COMMAND

E.29 LBN_SELCANCEL

E.29.1 Description

The parent window of a list box control is sent an LBN_SELCANCEL notification in a WM_COMMAND message when the user cancels the selection of an item in a list box. This notification is only sent if the list box control has the LBS_NOTIFY style.

Parameter	**Description**
wParam	Contains the list box control identifier.
lParam	Combines the LBN_ERRSPACE notification value in the high-order 16-bits and the 16-bit handle of the list box control in the low-order word.

E.29.2 Cross-References

LBN_DBLCLK, LBN_SELCHANGE, LB_SETCURSEL, WM_COMMAND

E.30 LBN_SELCHANGE

E.30.1 Description

The parent window of a list box control is sent an LBN_SELCHANGE notification in a WM_COMMAND message when the user changes the selection of an item in a list box. This notification is only sent if the list box control has the LBS_NOTIFY style, but is not sent if the selection changes in response to an LB_SETCURSEL message. For a multiple-selection list box, this notification is sent whenever the user presses an arrow key, regardless of whether the selection actually changes.

Parameter	**Description**
wParam	Contains the list box control identifier.
lParam	Combines the LBN_ERRSPACE notification value in the high-order 16-bits and the 16-bit handle of the list box control in the low-order word.

E.30.2 Cross-References

LBN_DBLCLK, LBN_SELCANCEL, LB_SETCURSEL, WM_COMMAND

E.31 LBN_SETFOCUS

E.31.1 Description

The parent window of a list box control is sent an LBN_SETFOCUS notification in a WM_COMMAND message when the control receives input focus.

Parameter	**Description**
wParam	Contains the list box control identifier.
lParam	Combines the LBN_ERRSPACE notification value in the high-order 16-bits and the 16-bit handle of the list box control in the low-order word.

E.31.2 Cross-References

LBN_KILLFOCUS, WM_COMMAND

Annex F

Window Styles

Description

This annex describes the following window styles: general window styles, button styles, combo box styles, edit control styles, list box styles, scroll bar styles, and static control styles.

F.1 GENERAL WINDOW STYLES

The *CreateWindow()* function's *dwStyle* parameter specifies the window styles of the new window being created. The value of the *dwStyle* parameter can be one or more of the following constant values OR'ed together:

Style	**Meaning**
MDIS_ALLCHILDSTYLES	Window is a multilple document interface (MDI) client window that can have any combination of window styles. If this style is not used, an MDI child window will have by default only the WS_MINIMIZE, WS_MAXIMIZE, WS_HSCROLL, and WS_VSCROLL styles set.
WS_BORDER	Window has a border.
WS_CAPTION	Window has a title bar and uses the WS_BORDER style. This style cannot be combined with the WS_DLGFRAME style.
WS_CHILD	Window is a child window. This style cannot be combined with the WS_POPUP style. Same as the WS_CHILDWINDOW style.
WS_CHILDWINDOW	Same as the WS_CHILD style.
WS_CLIPCHILDREN	When drawing within the parent window, the area occupied by child windows is excluded. This style is typically used when creating a parent window.
WS_CLIPSIBLINGS	When a child window receives a paint message and needs to be updated, this style clips all other overlapped child windows out of the child window's update region. If this style is not used and child windows overlap, it is possible to unintentionally draw within the client area of other neighboring, overlapping child windows. This style should only be used with the WS_CHILD style.
WS_DISABLED	Window is initially disabled.
WS_DLGFRAME	Window has a double border but no title.
WS_GROUP	This style is only used for dialog boxes. The style designates a control (a window) as being the first control in a group of controls. When a control in the group is selected, the arrow keys can be used to move from one control to any other control in the same group. If another control in the dialog box is encountered and it has the WS_GROUP style also set, it marks the end of the current group and the start of another group.
WS_MAXIMIZE	This style creates a window that has a maximum size.
WS_MAXIMIZEBOX	This style creates a window that has a Maximize box.
WS_MINIMIZE	This style creates a window that has a minimum size.
WS_MINIMIZEBOX	This style creates a window that has a Minimize box.
WS_OVERLAPPED	This style creates an overlapped window. An overlapped window is one that has a caption and a border.

WS_OVERLAPPEDWINDOW	This style creates an overlapped window that has the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles.
WS_POPUP	This style creates a pop-up window. This cannot be used with the WS_CHILD style.
WS_POPUPWINDOW	This style creates a pop-up window that has the WS_POPUP, WS_BORDER, and WS_SYSMENU styles. To make the System menu visible WS_CAPTION style must be combined with the WS_POPUPWINDOW style.
WS_SYSMENU	Creates a window that has a System-menu box in its title bar. This is used only for windows with title bars. If it is used with a child window, then this style creates a Close box instead of a System-menu box.

F.2 BUTTON STYLES

The following are styles used in the dwStyle parameter in CreateWindow() when creating buttons.

Style	**Meaning**
BS_3STATE	Creates a check box button that can be either checked, unchecked or grayed. The grayed state implies that the state of the check is undefined.
BS_AUTO3STATE	Same as BS_3STATE, except each time the user clicks on it the state changes to the next state in the cycle: checked, grayed, or unchecked.
BS_AUTOCHECKBOX	This check box alternates between being checked and unchecked each time the user clicks it.
BS_AUTORADIOBUTTON	This button highlights itself when a user clicks it and causes any other button in the same group to become unhighlighted.
BS_CHECKBOX	Creates a small square that can have an "X" within it, indicating that it is selected. It also has text displayed to the right of the square unless BS_LEFTTEXT is used.
BS_DEFPUSHBUTTON	Causes the button to have a heavy border and is automatically pushed if the user presses the ENTER key.
BS_GROUPBOX	Creates a rectangle to group other buttons. Any associated text is placed in the upper left corner of the rectangle.
BS_LEFTTEXT	Causes text to be placed on the left side of a radio button or check box.
BS_OWNERDRAW	Creates an owner-drawn button. Cannot be combined with any other button styles.
BS_PUSHBUTTON	Creates a rounded rectangle push button.
BS_RADIOBUTTON	Creates a radio button that is a small circle with text displayed to the right or left of the circle.

F.3 COMBO BOX STYLES

This section describes combo box styles.

Style	**Meaning**
CBS_AUTOHSCROLL	If the combo box's edit control is completely filled with text and the user enters more text at the end of the edit control line, the existing text is automatically scrolled. If this style is not set and the edit control is completely filled with text, no more text is allowed to be entered into the edit control.

CBS_DISABLENOSCROLL	A scroll bar is always shown in the combo box's list box. When the list box does not contain enough items to require scrolling, the scroll bar is disabled but still visible. If this style is not set, the scroll bar is only visible when there are enough items in the list box to require scrolling.
CBS_DROPDOWN	Similar to the CBS_SIMPLE style, the CBS_DROPDOWN style causes the combo box's list box to be hidden until the user presses the Arrow button located next to the combo box's edit control.
CBS_DROPDOWNLIST	Similar to the CBS_DROPDOWN style, the CBS_DROPDOWNLIST style causes the combo box's edit control to be set to read-only. The user cannot edit the contents of the edit control.
CBS_HASSTRINGS	Declares that the entries in an owner-drawn combo box are strings. When this style is set for an owner-drawn combo box, memory and pointer information is maintained for each entry in the combo box and thus allow an application to use the CB_GETLBTEXT message.
CBS_NOINTEGRALHEIGHT	If this style is not used, the system automatically resizes the height of the combo box so that none of its items are partially displayed. If this style is used, the system is prevented from automatically resizing the height of the combo box.
CBS_OEMCONVERT	When this style is used, text that is entered into the combo box's edit control is automatically converted from the system's character set to the OEM character set and then back again to the system's character set. This sequence ensures that proper character conversion can occur when an application calls the AnsiToOem() function to convert a string in the combo box's edit control to OEM characters. This style should only be used in combination with the CBS_SIMPLE or CBS_DROPDOWN styles. This style is best used on combo boxes that contain filenames.
CBS_OWNERDRAWFIXED	When this style is used, the system makes the owner of the combo box responsible for drawing its contents and the height of all of the list box's items is the same. When the combo box is created, the owner of the combo box receives a WM_MEASUREITEM message. Whenever a visible aspect of the combo box changes, the owner of the combo box receives a WM_DRAWITEM message.
CBS_OWNERDRAWVARIABLE	When this style is used, the system makes the owner of the combo box responsible for drawing its contents and the height of each of the list box's items is not the same. When the combo box is created, the owner of the combo box receives a WM_MEASUREITEM message. Whenever a visible aspect of the combo box changes, the owner of the combo box receives a WM_DRAWITEM message.
CBS_SIMPLE	When this style is used, the combo box's list box is always displayed. When an item is selected in the combo box's list box, the item's text is shown in the combo box's edit control.
CBS_SORT	The combo box's list box entries are automatically sorted.

F.4 EDIT CONTROL STYLES

This section describes edit control styles.

Style	Meaning
ES_AUTOHSCROLL	Creates an edit control that automatically scrolls horizontally as text is entered. With this style off, only the text within the visible area is valid for single-line edit controls. For multiline edit controls, without this style, the text is wrapped to the next line. If an edit control has a WS_HSCROLL style, the ES_AUTOHSCROLL style is applied automatically. This style cannot be used with center or right justified edit controls.
ES_AUTOVSCROLL	Creates an edit control that automatically scrolls vertically when there is more text than can be displayed within the control. This style is applicable to multiline edit controls only. With this style off, the edit control ignores input that cannot be displayed. If an edit control has a WS_VSCROLL style, the ES_AUTOVSCROLL style is applied automatically.
ES_CENTER	Specifies that multiline edit controls center justify text. Cannot be used for single-line edit controls. Also cannot be used in combination with the ES_AUTOHSCROLL or WS_HSCROLL styles.
ES_LEFT	Specifies that the edit control left justify its text.
ES_LOWERCASE	All uppercase characters entered into the edit control is displayed as lowercase.
ES_MULTILINE	Causes the edit control to be a multiline control.
ES_NOHIDESEL	Negates the default behavior for an edit control, which is to hide the selection when the control loses the input focus and invert the selection when the control receives the input focus.
ES_OEMCONVERT	Converts text entered in the edit control from the default character set to the OEM character set and then back to the default character set. This ensures proper character conversion for the AnsiToOem() function to convert a string in the edit control to OEM characters.
ES_PASSWORD	Hides all characters and displays them as an asterisk as they are typed into the edit control. An application can use the EM_SETPASSWORDCHAR message to change the default asterisk character.
ES_READONLY	Prevents the user from typing or editing text in the edit control.
ES_RIGHT	Right aligns text in a multiline edit control.
ES_UPPERCASE	Converts all characters to uppercase as they are typed into the edit control.
ES_WANTRETURN	Specifies that a carriage return be inserted when the user presses the ENTER key while entering text into a multiline edit control. Otherwise, pressing the ENTER key has the same effect as pressing the dialog box's default push button.

F.5 LIST BOX STYLES

The following are list box styles that an application can specify in the *dwStyle* parameter.

Style	Meaning
LBS_DISABLENOSCROLL	Shows a disabled vertical scroll bar for the list box when the box does not contain enough items to scroll. If this style is not specified, the scroll bar is hidden rather than displayed as disabled.

LBS_EXTENDEDSEL	Allows multiple items to be selected by using the SHIFT key and the mouse, or special key combinations.
LBS_HASSTRINGS	Specifies that a list box contains items consisting of strings. By default, all list boxes except owner-drawn list boxes, have this style. An application can create an owner-drawn list box either with or without this style.
LBS_MULTICOLUMN	Specifies a multicolumn list box that is scrolled horizontally. The LB_SETCOLUMNWIDTH message sets the width of the columns.
LBS_MULTIPLESEL	This list box style allows for the selection of any number of strings. Selection or deselection is made with each click or double-click.
LBS_NOINTEGRALHEIGHT	This list box style prevents the height of a list box from being adjusted so that partial items are not displayed.
LBS_NOREDRAW	This list box style suppresses screen updates when changes are made. An application can set or reset this style by sending a WM_SETREDRAW message to the list box control.
LBS_NOTIFY	This list box style enables the notification of the parent window when a string in the list box is clicked or double-clicked.
LBS_OWNERDRAWFIXED	This list box style specifies that the application is responsible for drawing the list box and that all the items have the same height. When the control is created, the owner is sent a WM_MEASUREITEM message. When the control changes its appearance, the owner is sent a WM_DRAWITEM message.
LBS_OWNERDRAWVARIABLE	This list box style specifies that the application is responsible for drawing the list box and that the items in the list box vary in height. When the control is created, the owner is sent a WM_MEASUREITEM message for each item. When the control changes its appearance, the owner is sent a WM_DRAWITEM message.
LBS_SORT	This list box style causes the list box strings to be sorted alphabetically.
LBS_STANDARD	This list box style combines the LBS_NOTIFY and LBS_SORT list box styles with the windows styles WS_BORDER and WS_VSCROLL.
LBS_USETABSTOPS	This list box style expands tab characters within its content strings. Tab positions may be set with an LB_SETTABSTOPS message. Otherwise, the default tab positions are 32 dialog box units.
LBS_WANTKEYBOARDINPUT	This list box style indicates that the owner receives WM_VKEYTOITEM or WM_CHARTOITEM messages when the list box has input focus and the user presses a key. If the control also has the LBS_HASSTRINGS style, only the WM_KEYTOITEM messages are sent. Otherwise, only WM_CHARTOITEM messages are sent.

F.6 SCROLL BAR STYLES

The *CreateWindow()* function's *dwStyle* parameter specifies the window styles of a new predefined control that is being created. The value of the *dwStyle* parameter can be one or more of the following scroll bar styles OR'ed together:

Style	**Meaning**
SBS_BOTTOMALIGN	The bottom edge of the scroll bar is aligned with the bottom edge of the rectangle given by the *CreateWindow()* function's *x*, *y*, *nWidth*, and *nHeight* parameters. The scroll bar has the default height for system scroll bars. This style should only be used with the SBS_HORZ style.

SBS_HORZ	Has a horizontal scroll bar. If the SBS_BOTTOMALIGN and SBS_TOPALIGN styles are not used, the scroll bar has the height, width, and position given by the *CreateWindow()* function's parameters.
SBS_LEFTALIGN	The left edge of the scroll bar is aligned with the left edge of the rectangle given by the *CreateWindow()* function's parameters. The scroll bar has the default width for system scroll bars. This style should only be used with the SBS_VERT style.
SBS_RIGHTALIGN	The right edge of the scroll bar is aligned with the right edge of the rectangle given by the *CreateWindow()* function's parameters. The scroll bar has the default width for system scroll bars. This style should only be used with the SBS_VERT style.
SBS_SIZEBOX	Has a size box. If the SBS_SIZEBOXBOTTOMRIGHTALIGN and SBS_SIZEBOXTOPLEFTALIGN styles are not used, the size box has the height, width, and position given by the *CreateWindow()* function's parameters.
SBS_SIZEBOXBOTTOMRIGHTALIGN	The lower-right corner of the size box is aligned with the lower-right corner of the rectangle given by the *CreateWindow()* function's parameters. The size box has the default size for system size boxes. This style should only be used with the SBS_SIZEBOX style.
SBS_ SIZEBOXTOPLEFTALIGN	The upper-left corner of the size box is aligned with the upper-left corner of the rectangle given by the *CreateWindow()* function's parameters. The size box has the default size for system size boxes. This style should only be used with the SBS_SIZEBOX style.
SBS_TOPALIGN	The top edge of the scroll bar is aligned with the top edge of the rectangle given by the *CreateWindow()* parameters. The scroll bar has the default height for system scroll bars. This style should only be used with the SBS_HORZ style.
SBS_VERT	Has a vertical scroll bar. If the SBS_RIGHTALIGN and SBS_LEFTALIGN styles are not used, the scroll bar has the height, width, and position given by the *CreateWindow()* function's parameters.

F.7 STATIC CONTROL STYLES

The *CreateWindow()* function's *dwStyle* parameter specifies the window styles of a new predefined control that is being created. The value of the *dwStyle* parameter can be one or more of the following static control styles OR'ed together:

Style	**Meaning**
SS_BLACKFRAME	Specifies a box with a frame drawn with the same color as window frames. By default, the color is black.
SS_BLACKRECT	Specifies a rectangle filled with the same color as window frames. By default, the color is black.
SS_CENTER	Specifies a simple rectangle and aligns the displayed text in the center. The text is formatted before it is displayed in the rectangle. Lines that are too long to fit in the rectangle are automatically wrapped to the beginning of the next line.
SS_GRAYFRAME	Specifies a box with a frame drawn with the same color as the desktop. By default, the color is gray.
SS_GRAYRECT	Specifies a rectangle filled with the color used to fill the screen background. This color is gray if the default Windows color scheme is selected

SS_ICON	Designates an icon displayed in the dialog box. The given text is the name of an icon (not a filename) defined elsewhere in the resource file. The *nWidth* and *nHeight* parameters are ignored; the icon auto sizes itself.
SS_LEFT	Designates a simple rectangle and left-aligns the displayed text. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next left-aligned line.
SS_LEFTNOWORDWRAP	Designates a simple rectangle and left-aligns the displayed text in the rectangle. Tabs are expanded, but words are not wrapped. Text that extends past the end of a line is clipped.
SS_NOPREFIX	Prevents interpretation of any ampersand (&) characters in the control's text as accelerator prefix characters (which are displayed with the & character removed and the next character in the string underlined). This static control style may be included with any of the defined static controls. You can combine SS_NOPREFIX with other styles by using the bitwise OR operator. This style is most often used when filenames or other strings that may contain an & character need to be displayed in a static control in a dialog box.
SS_RIGHT	Designates a simple rectangle and right-aligns the displayed text in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next right-aligned line.
SS_SIMPLE	Designates a simple rectangle and displays a single line of text left-aligned in the rectangle. The line of text cannot be shortened or altered in any way. (The control's parent window or dialog box must not process the WM_CTLCOLOR message.)
SS_WHITEFRAME	Designates a frame drawn in the same color as the window background. (The default color is white.)
SS_WHITERECT	Designates a rectangle filled with the window background color. (The default color is white.)

F.8 DIALOG BOX STYLES

The following are styles used in the *dwStyle* parameter in *CreateWindow()* when creating dialogs.

Style	**Meaning**
DS_MODALFRAME	Creates a dialog box with a modal frame.
DS_NOIDLEMSG	Tells the system not to send WM_ENTERIDLE messages to the owner of the dialog box while the dialog box is displayed.
DS_SYSMODAL	Creates a system modal dialog box.

Annex G

Macros

Description

This annex describes supported macros.

G.1 DECLARE_HANDLE

G.1.1 Synopsis

void DECLARE_HANDLE(char *DataTypeName);

G.1.2 Description

DECLARE_HANDLE is used to define a data type that has the name specified in the parameter *DataTypeName* and is a 16-bit handle.

G.1.3 Returns

None.

G.1.4 Errors

None.

G.1.5 Cross-References

DECLARE_HANDLE32

G.2 DECLARE_HANDLE32

G.2.1 Synopsis

DECLARE_HANDLE32(DataTypeName)

G.2.2 Description

DECLARE_HANDLE is used to define a data type that has the name specified in the parameter *DataTypeName* and is a 32-bit handle.

G.2.3 Returns

None.

G.2.4 Errors

None.

G.2.5 Cross-References

DECLARE_HANDLE

G.3 FIELDOFFSET

G.3.1 Synopsis

int FIELDOFFSET(char *StructureName, char *ElementName);

G.3.2 Description

FIELDOFFSET retrieves the address offset of an element that is inside of a structure. The parameter *StructureName* specifies the name of the structure. The parameter *ElementName* specifies the name of the element that is inside of the structure.

G.3.3 Returns

Returns the address offset of the specified element.

G.3.4 Errors

None.

G.3.5 Cross-References

None.

G.4 GetBValue

G.4.1 Synopsis

BYTE GetBValue(DWORD RGBValue);

G.4.2 Description

The macro returns a value that represents the intensity of blue color in a red-green-blue (RGB) value. The parameter *RGBValue* is a 32-bit RGB value whose intensity of blue color will be returned.

G.4.3 Returns

Returns a value that represents the intensity of blue color in a RGB value.

G.4.4 Errors

None.

G.4.5 Cross-References

None.

G.5 GetGValue

G.5.1 Synopsis

BYTE GetGValue(DWORD RGBValue);

G.5.2 Description

The macro returns a value that represents the intensity of green color in a red-green-blue (RGB) value. The parameter *RGBValue* is a 32-bit RGB value whose intensity of green color will be returned.

G.5.3 Returns

Returns a value that represents the intensity of green color in a RGB value.

G.5.4 Errors

None.

G.5.5 Cross-References

None.

G.6 GetRValue

G.6.1 Synopsis

BYTE GetRValue(DWORD RGBValue);

G.6.2 Description

The macro returns a value that represents the intensity of red color in a red-green-blue (RGB) value. The parameter *RGBValue* is a 32-bit RGB value whose intensity of red color will be returned.

G.6.3 Returns

Returns a value that represents the intensity of red color in a RGB value.

G.6.4 Errors

None.

G.6.5 Cross-References

None.

G.7 HIBYTE

G.7.1 Synopsis

BYTE HIBYTE(WORD Number);

G.7.2 Description

HIBYTE returns the value of the hi-order byte of a WORD value. The parameter *Number* is a WORD value whose high-order byte value will be returned.

G.7.3 Returns

Returns the value of the high-order byte of a WORD value.

G.7.4 Errors

None.

G.7.5 Cross-References

LOBYTE

G.8 HIWORD

G.8.1 Synopsis

WORD HIWORD(DWORD Number);

G.8.2 Description

HIWORD returns the value of the high-order WORD of a DWORD value. The parameter *Number* is a DWORD value whose high-order WORD value will be returned.

G.8.3 Returns

Returns the value of the high-order WORD of the specified DWORD value.

G.8.4 Errors

None.

G.8.5 Cross-References

LOWORD

G.9 LOBYTE

G.9.1 Synopsis

BYTE LOBYTE(WORD Number);

G.9.2 Description

LOBYTE returns the value of the low-order byte of a WORD value. The parameter *Number* is a WORD value whose low-order byte value will be returned.

G.9.3 Returns

Returns the value of the low-order byte of the specified WORD value.

G.9.4 Errors

None.

G.9.5 Cross-References

HIBYTE

G.10 LockData

G.10.1 Synopsis

HANDLE LockData(Unused);

G.10.2 Description

The macro locks the current data segment in memory and returns a handle to it. The parameter *Unused* is not used.

G.10.3 Returns

If the macro is successful, it returns a handle to the locked data segment. If the macro is not successful, it returns the value NULL.

G.10.4 Errors

None.

G.10.5 Cross-References

None.

G.11 LOWORD

G.11.1 Synopsis

WORD LOWORD(DWORD Number);

G.11.2 Description

LOWORD returns the value of the low-order WORD of a DWORD value. The parameter *Number* is a DWORD value whose low-order WORD value will be returned.

G.11.3 Returns

Returns the value of the low-order WORD of the specified DWORD value.

G.11.4 Errors

None.

G.11.5 Cross-References

HIWORD

G.12 MAKEINTATOM

G.12.1 Synopsis

LPCSTR MAKEINTATOM(WORD wValue);

G.12.2 Description

MAKEINTATOM creates an integer atom from a given WORD value. The parameter *wValue* is the value to use when creating the integer atom. The integer atom that is returned by the macro should only be used with one of the API's atom-management functions.

G.12.3 Returns

The macro returns a pointer to the integer atom created from the given WORD value.

G.12.4 Errors

Other than a return value, no other error information is provided by the macro.

G.12.5 Cross-References

AddAtom(), DeleteAtom(), GetAtomName()

G.13 MAKEINTRESOURCE

G.13.1 Synopsis

LPCSTR MAKEINTRESOURCE(WORD wResourceID);

G.13.2 Description

MAKEINTRESOURCE processed a resource’s identifier and returns it in a form that will be understood by the API’s resource-management functions. An application can use this macro instead of passing the name of the resource to one of the API’s resource-management functions. The parameter *wResourceID* is the identifier of the resource to be processed.

G.13.3 Returns

The macro returns the resource’s identifier in a form that will be understood by the API’s resource management functions.

G.13.4 Errors

Other than a return value, no other error information is provided by the macro.

G.13.5 Cross-References

MAKELP

G.14 MAKELONG

G.14.1 Synopsis

DWORD MAKELONG(WORD wLowValue, WORD wHighValue);

G.14.2 Description

MAKELONG returns a DWORD value with the specified high-order and low-order WORD values.

G.14.3 Returns

A DWORD value with the specified high-order and low-order WORD values.

G.14.4 Errors

Other than a return value, no other error information is provided by the macro.

G.14.5 Cross-References

MAKELP

G.15 MAKELP

G.15.1 Synopsis

void *MAKELP(WORD wSelector, WORD wOffset);

G.15.2 Description

MAKELP returns a pointer to the memory address specified by a specified segment selector and an address offset. The parameter *wSelector* specifies the segment selector. The parameter *wOffset* specifies the address offset.

G.15.3 Returns

A pointer to the memory address.

G.15.4 Errors

Other than a return value, no other error information is provided by the macro.

G.15.5 Cross-References

None.

G.16 MAKELPARAM

G.16.1 Synopsis

LPARAM MAKELPARAM(WORD wLowValue, WORD wHighValue);

G.16.2 Description

MAKELPARAM returns a value of type LPARAM with the specified high-order and low-order WORD values. The parameter *wLowValue* specifies the low-order value of the LPARAM value. The parameter *wHighValue* specifies the high-order value of the LPARAM value.

G.16.3 Returns

A value of type LPARAM with the specified high-order and low-order WORD values.

G.16.4 Errors

Other than a return value, no other error information is provided by the macro.

G.16.5 Cross-References

None.

G.17 MAKELRESULT

G.17.1 Synopsis

LRESULT MAKELRESULT(WORD wLowValue, WORD wHighValue);

G.17.2 Description

MAKELRESULT returns a value of type LRESULT with the specified high-order and low-order WORD values. The parameter *wLowValue* specifies the low-order value of the LRESULT value. The parameter *wHighValue* specifies the high-order value of the LRESULT value.

G.17.3 Returns

A value of type LRESULT with the specified high-order and low-order WORD values.

G.17.4 Errors

Other than a return value, no other error information is provided by the macro.

G.17.5 Cross-References

None.

G.18 MAKEPOINT

G.18.1 Synopsis

POINT MAKEPOINT(DWORD dwCoord);

G.18.2 Description

MAKEPOINT converts a specified DWORD value into a point’s coordinates and returns the coordinates in a **POINT** structure. The low-order word of the *dwCoord* parameter should contain the x-coordinate of the point. The high-order word of the *dwCoord* parameter should contain the y-coordinate of the point.

This macro can be used to convert a mouse message’s *lParam* value into mouse coordinates or to convert the value returned by the *GetMessagePos()* function into a **POINT** structure.

G.18.3 Returns

The MAKEPOINT macro returns a pointer to a **POINT** structure.

G.18.4 Errors

Other than a return value, no other error information is provided by the macro.

G.18.5 Cross-References

POINT

G.19 max

G.19.1 Synopsis

int max(FirstValue, SecondValue);

G.19.2 Description

The macro compares two values and returns the larger of the two values. The two values are specified in the parameters *FirstValue* and *SecondValue*. The types of the two values and the type of the return value will be the same. A numerical type can be passed to the macro.

G.19.3 Returns

The larger of the two values is returned.

G.19.4 Errors

Other than a return value, no other error information is provided by the macro.

G.19.5 Cross-References

min

G.20 min

G.20.1 Synopsis

int min(FirstValue, SecondValue);

G.20.2 Description

The macro compares two values and returns the lesser of the two values. The two values are specified in the parameters *FirstValue* and *SecondValue*. The types of the two values and the type of the return value will be the same. An numerical type can be passed to the macro.

G.20.3 Returns

The larger of the two values is returned.

G.20.4 Errors

Other than a return value, no other error information is provided by the macro.

G.20.5 Cross-References

max

G.21 OFFSETOF

G.21.1 Synopsis

WORD OFFSETOF(void *Pointer);

G.21.2 Description

The OFFSETOF macro retrieves the address offset of the given pointer. The parameter *Pointer* is the pointer whose address offset should be retrieved.

G.21.3 Returns

Retrieves the address offset of the given pointer.

G.21.4 Errors

Other than a return value, no other error information is provided by the macro.

G.21.5 Cross-References

SELECTOROF

G.22 PALETTEINDEX

G.22.1 Synopsis

COLORREF PALETTERGB(BYTE RedValue, BYTE GreenValue, BYTE BlueValue);

G.22.2 Description

PALETTERGB creates a palette-relative RGB specifier from the specified red, green, and blue relative intensity values passed to the macro. The parameter *RedValue* contains the level of red intensity desired. The parameter *GreenValue* contains the level of green intensity desired. The parameter *BlueValue* contains the level of blue intensity desired.

A palette-relative RGB specifier is a value of type COLORREF that contains an RGB value in the low-order byte and the value 2 in the high-order byte. An application can pass a palette-entry value instead of an RGB value to any graphics device interface (GDI) function that accepts an RGB value as one of its function arguments.

G.22.3 Returns

A palette-entry specifier is a value containing the index of the logical-color palette entry.

G.22.4 Errors

Other than a return value, no other error information is provided by the macro.

G.22.5 Cross-References

PALETTERGB, RGB

G.23 PALETTERGB

G.23.1 Synopsis

COLORREF PALETTERGB(WORD wIndexNum)

G.23.2 Description

PALETTERGB creates a palette-entry specifier using the index of a logical-color palette entry. The parameter *wIndexNum* is the index of a logical-color palette entry.

A palette-entry specifier is a value of type COLORREF that contains the index of a logical-color palette entry in the low-order byte and the value 1 in the high-order byte. An application can pass a palette-entry value instead of an RGB value to any API function that accepts an RGB value as one of its function arguments.

G.23.3 Returns

A palette-entry specifier is a value containing the index of the logical-color palette entry.

G.23.4 Errors

Other than a return value, no other error information is provided by the macro.

G.23.5 Cross-References

PALETTERGB, RGB

G.24 RGB

G.24.1 Synopsis

COLORREF RGB(BYTE RedValue, BYTE GreenValue, BYTE BlueValue);

G.24.2 Description

RGB returns a value of type COLORREF that contains the specified red, green, and blue relative intensity values passed to the macro. The parameter *RedValue* contains the level of red intensity desired. The parameter *GreenValue* contains the level of green intensity desired. The parameter *BlueValue* contains the level of blue intensity desired.

G.24.3 Returns

A value of type COLORREF that contains the specified red, green, and blue relative intensity values passed to the macro.

G.24.4 Errors

Other than a return value, no other error information is provided by the macro.

G.24.5 Cross-References

PALETTEINDEX, PALETTERGB

G.25 SELECTOROF

G.25.1 Synopsis

WORD SELECTOROF(void *Pointer);

G.25.2 Description

SELECTOROF retrieves the segment selector of the given pointer. The parameter *Pointer* is the pointer whose segment selector should be retrieved.

G.25.3 Returns

Retrieves the segment selector of the given pointer.

G.25.4 Errors

Other than a return value, no other error information is provided by the macro.

G.25.5 Cross-References

OFFSETOF

G.26 UnlockData

G.26.1 Synopsis

HANDLE UnlockData(Unused);

G.26.2 Description

The macro unlocks the current data segment. The parameter *Unused* is not used.

G.26.3 Returns

The macro returns the data segment’s lock count after the data segment’s lock count is decreased by one.

G.26.4 Errors

Other than a return value, no other error information is provided by the macro.

G.26.5 Cross-References

LockData, LockSegment(), UnlockSegment()

G.27 UnlockResource

G.27.1 Synopsis

BOOL UnlockResource(HGLOBAL hResource);

G.27.2 Description

The macro unlocks the handle of a resource. The parameter *hResource* is the handle of the resource to unlock.

G.27.3 Returns

The macro returns FALSE if the resource’s reference count is zero after the macro is executed. The macro returns TRUE if the resource’s reference count is not zero after the macro is executed.

G.27.4 Errors

Other than a return value, no other error information is provided by the macro.

G.27.5 Cross-References

GlobalUnlock()

Annex H

Binary Raster Operations

Raster Operation	Meaning
R2_BLACK	Sets the pixel value in the destination bitmap to black.
R2_WHITE	Sets the pixel value in the destination bitmap to white.
R2_COPYPEN	Replaces the pixel value in the destination with the pixel value of the pen.
R2_MASKNOTPEN	Replaces the pixel value of the destination with the result of the destination AND'ed with the INVERSE pixel value of the pen.
R2_MASKPEN	Replaces the pixel value of the destination with the result of the destination bitmap AND'ed with the pixel value of the pen.
R2_MASKPENNOT	Replaces the pixel value of the destination with the INVERSE of the destination bitmap pixel value AND'ed with the pixel value of the pen.
R2_MERGETNOTPEN	Replaces the pixel value of the destination with the result of the destination bitmap OR'ed with the INVERSE pixel value of the pen.
R2_MERGEPEN	Replaces the pixel value of the destination with the result of the destination OR'ed with the pixel value of the pen.
R2_MERGEPENNOT	Replaces the pixel value of the destination with the INVERSE of the destination bitmap pixel value OR'ed with the pixel value of the pen.
R2_NOP	The destination bitmap is not altered.
R2_NOT	INVERTs the value of the destination bitmap pixel value.
R2_NOTCOPYPEN	Replaces the pixel value in the destination bitmap with the INVERSE of the pixel value of the pen.
R2_NOTMASKPEN	Replaces the pixel value in the destination bitmap with the INVERSE result of the destination bitmap AND'ed with the pixel value of the pen.
R2_NOTMERGEPEN	Replaces the pixel value of the destination bitmap with the INVERSE result of the destination bitmap OR'ed with the pixel value of the pen.
R2_NOTXORPEN	Replaces the pixel value of the destination bitmap with the INVERSE result of the destination bitmap XOR'ed with the pixel value of the pen.
R2_XORPEN	Replaces the pixel value of the destination bitmap with the result of the destination bitmap XOR'ed with the pixel value of the pen.