

Contents

[Windows GDI](#)

[Fontsub.h](#)

[Overview](#)

[CFP_ALLOCPROC callback function](#)

[CFP_FREEPROC callback function](#)

[CFP_REALLOCPROC callback function](#)

[CreateFontPackage function](#)

[MergeFontPackage function](#)

[Mmsystem.h](#)

[Overview](#)

[DIBINDEX macro](#)

[Prnasnot.h](#)

[Overview](#)

[Prntvpt.h](#)

[Overview](#)

[T2embapi.h](#)

[Overview](#)

[TTCharToUnicode function](#)

[TTDeleteEmbeddedFont function](#)

[TTEmbedFont function](#)

[TTEmbedFontEx function](#)

[TTEmbedFontFromFileA function](#)

[TTEMBEDINFO structure](#)

[TTEnableEmbeddingForFacename function](#)

[TTGetEmbeddedFontInfo function](#)

[TTGetEmbeddingType function](#)

[TTGetNewFontName function](#)

[TTIsEmbeddingEnabled function](#)

[TTIsEmbeddingEnabledForFacename function](#)

[TTLoadEmbeddedFont function](#)

[TTLOADINFO structure](#)

[TTRunValidationTests function](#)

[TTRunValidationTestsEx function](#)

[TTVALIDATIONTESTSPARAMS structure](#)

[TTVALIDATIONTESTSPARAMSEX structure](#)

[Tvout.h](#)

[Overview](#)

[Windef.h](#)

[Overview](#)

[Windowsx.h](#)

[Overview](#)

[DeleteFont macro](#)

[SelectFont macro](#)

[Wingdi.h](#)

[Overview](#)

[ABC structure](#)

[ABCFLOAT structure](#)

[AbortPath function](#)

[AddFontMemResourceEx function](#)

[AddFontResourceA function](#)

[AddFontResourceExA function](#)

[AddFontResourceExW function](#)

[AddFontResourceW function](#)

[AlphaBlend function](#)

[AngleArc function](#)

[AnimatePalette function](#)

[Arc function](#)

[ArcTo function](#)

[AXESLISTA structure](#)

[AXESLISTW structure](#)

[AXISINFOA structure](#)

[AXISINFOW structure](#)
[BeginPath function](#)
[BitBlt function](#)
[BITMAP structure](#)
[BITMAPCOREHEADER structure](#)
[BITMAPCOREINFO structure](#)
[BITMAPFILEHEADER structure](#)
[BITMAPINFO structure](#)
[BITMAPV4HEADER structure](#)
[BITMAPV5HEADER structure](#)
[BLENDFUNCTION structure](#)
[CancelDC function](#)
[Chord function](#)
[CloseEnhMetaFile function](#)
[CloseFigure function](#)
[CloseMetaFile function](#)
[COLORADJUSTMENT structure](#)
[CombineRgn function](#)
[CombineTransform function](#)
[CopyEnhMetaFileA function](#)
[CopyEnhMetaFileW function](#)
[CopyMetaFileA function](#)
[CopyMetaFileW function](#)
[CreateBitmap function](#)
[CreateBitmapIndirect function](#)
[CreateBrushIndirect function](#)
[CreateCompatibleBitmap function](#)
[CreateCompatibleDC function](#)
[CreateDCA function](#)
[CreateDCW function](#)
[CreateDIBitmap function](#)
[CreateDIBPatternBrush function](#)

[CreateDIBPatternBrushPt](#) function
[CreateDIBSection](#) function
[CreateDiscardableBitmap](#) function
[CreateEllipticRgn](#) function
[CreateEllipticRgnIndirect](#) function
[CreateEnhMetaFileA](#) function
[CreateEnhMetaFileW](#) function
[CreateFontA](#) function
[CreateFontIndirectA](#) function
[CreateFontIndirectExA](#) function
[CreateFontIndirectExW](#) function
[CreateFontIndirectW](#) function
[CreateFontW](#) function
[CreateHalftonePalette](#) function
[CreateHatchBrush](#) function
[CreateICA](#) function
[CreateICW](#) function
[CreateMetaFileA](#) function
[CreateMetaFileW](#) function
[CreatePalette](#) function
[CreatePatternBrush](#) function
[CreatePen](#) function
[CreatePenIndirect](#) function
[CreatePolygonRgn](#) function
[CreatePolyPolygonRgn](#) function
[CreateRectRgn](#) function
[CreateRectRgnIndirect](#) function
[CreateRoundRectRgn](#) function
[CreateScalableFontResourceA](#) function
[CreateScalableFontResourceW](#) function
[CreateSolidBrush](#) function
[DeleteDC](#) function

[DeleteEnhMetaFile](#) function
[DeleteMetaFile](#) function
[DeleteObject](#) function
[DESIGNVECTOR](#) structure
[DIBSECTION](#) structure
[DISPLAY_DEVICEA](#) structure
[DISPLAY_DEVICEW](#) structure
[DPtoLP](#) function
[DrawEscape](#) function
[Ellipse](#) function
[EMR](#) structure
[EMRABORTPATH](#) structure
[EMRALPHABLEND](#) structure
[EMRANGLEARC](#) structure
[EMRARC](#) structure
[EMRBITBLT](#) structure
[EMRCOLORMATCHTOTARGET](#) structure
[EMRCREATEBRUSHINDIRECT](#) structure
[EMRCREATECOLORSPACE](#) structure
[EMRCREATECOLORSPACEW](#) structure
[EMRCREATEDIBPATTERNBRUSHPT](#) structure
[EMRCREATEMONOBRUSH](#) structure
[EMRCREATEPALETTE](#) structure
[EMRCREATEPEN](#) structure
[EMRELLIPSE](#) structure
[EMREOF](#) structure
[EMREXCLUDECLIPRECT](#) structure
[EMREXTCREATEFONTINDIRECTW](#) structure
[EMREXTCREATEPEN](#) structure
[EMREXTFLOODFILL](#) structure
[EMREXTSELECTCLIPRGN](#) structure

EMREXTTEXTOUTA structure
EMRFILLPATH structure
EMRFILLRGN structure
EMRFORMAT structure
EMRFRAMERGN structure
EMRGDICOMMENT structure
EMRGLSBOUNDEDRECORD structure
EMRGLSRECORD structure
EMRGRADIENTFILL structure
EMRINVERTRGN structure
EMRLINETO structure
EMRMASKBLT structure
EMRMODIFYWORLDTRANSFORM structure
EMROFFSETCLIPRGN structure
EMRPIXELFORMAT structure
EMRPLGBLT structure
EMRPOLYDRAW structure
EMRPOLYDRAW16 structure
EMRPOLYLINE structure
EMRPOLYLINE16 structure
EMRPOLYPOLYLINE structure
EMRPOLYPOLYLINE16 structure
EMRPOLYTEXTOUTA structure
EMRRESIZEPALETTE structure
EMRRESTOREDC structure
EMRROUNDRECT structure
EMRSCALEVIEWPORTEXTEX structure
EMRSELECTCLIPPATH structure
EMRSELECTOBJECT structure
EMRSELECTPALETTE structure
EMRSETARCDIRECTION structure
EMRSETBKCOLOR structure

[EMRSETCOLORADJUSTMENT](#) structure
[EMRSETCOLORSPACE](#) structure
[EMRSETDIBITSTODEVICE](#) structure
[EMRSETICMPROFILE](#) structure
[EMRSETMAPPERFLAGS](#) structure
[EMRSETMITERLIMIT](#) structure
[EMRSETPALETTEENTRIES](#) structure
[EMRSETPIXELV](#) structure
[EMRSETVIEWPORTEXTEX](#) structure
[EMRSETVIEWPORTORGEX](#) structure
[EMRSETWORLDTRANSFORM](#) structure
[EMRSTRETCHBLT](#) structure
[EMRSTRETCHDIBITS](#) structure
[EMRTEXT](#) structure
[EMRTRANSPARENTBLT](#) structure
EndPath function
[ENHMETAHEADER](#) structure
[ENHMETARECORD](#) structure
[ENHMFENUMPROC](#) callback function
[EnumEnhMetaFile](#) function
[EnumFontFamiliesA](#) function
[EnumFontFamiliesExA](#) function
[EnumFontFamiliesExW](#) function
[EnumFontFamiliesW](#) function
[EnumFontsA](#) function
[EnumFontsW](#) function
[ENUMLOGFONTA](#) structure
[ENUMLOGFONTEXA](#) structure
[ENUMLOGFONTEXDVA](#) structure
[ENUMLOGFONTEXDVW](#) structure
[ENUMLOGFONTEXW](#) structure
[ENUMLOGFONTW](#) structure

[EnumMetaFile](#) function
[EnumObjects](#) function
[ENUMTEXTMETRICA](#) structure
[ENUMTEXTMETRICW](#) structure
[EqualRgn](#) function
[ExcludeClipRect](#) function
[ExtCreatePen](#) function
[ExtCreateRegion](#) function
[ExtFloodFill](#) function
[EXTLOGFONTA](#) structure
[EXTLOGFONTW](#) structure
[EXTLOGPEN](#) structure
[ExtSelectClipRgn](#) function
[ExtTextOutA](#) function
[ExtTextOutW](#) function
[FillPath](#) function
[FillRgn](#) function
[FIXED](#) structure
[FlattenPath](#) function
[FloodFill](#) function
[FrameRgn](#) function
[GCP_RESULTSA](#) structure
[GCP_RESULTSW](#) structure
[GdiAlphaBlend](#) function
[GdiComment](#) function
[GdiFlush](#) function
[GdiGetBatchLimit](#) function
[GdiGradientFill](#) function
[GdiSetBatchLimit](#) function
[GdiTransparentBlt](#) function
[GetArcDirection](#) function
[GetAspectRatioFilterEx](#) function

[GetBitmapBits function](#)
[GetBitmapDimensionEx function](#)
[GetBkColor function](#)
[GetBkMode function](#)
[GetBoundsRect function](#)
[GetBrushOrgEx function](#)
[GetBValue macro](#)
[GetCharABCWidthsA function](#)
[GetCharABCWidthsFloatA function](#)
[GetCharABCWidthsFloatW function](#)
[GetCharABCWidthsI function](#)
[GetCharABCWidthsW function](#)
[GetCharacterPlacementA function](#)
[GetCharacterPlacementW function](#)
[GetCharWidth32A function](#)
[GetCharWidth32W function](#)
[GetCharWidthA function](#)
[GetCharWidthFloatA function](#)
[GetCharWidthFloatW function](#)
[GetCharWidthI function](#)
[GetCharWidthW function](#)
[GetClipBox function](#)
[GetClipRgn function](#)
[GetColorAdjustment function](#)
[GetCurrentObject function](#)
[GetCurrentPositionEx function](#)
[GetDCBrushColor function](#)
[GetDCOrgEx function](#)
[GetDCPenColor function](#)
[GetDeviceCaps function](#)
[GetDIBColorTable function](#)
[GetDIBytes function](#)

[GetEnhMetaFileA](#) function
[GetEnhMetaFileBits](#) function
[GetEnhMetaFileDescriptionA](#) function
[GetEnhMetaFileDescriptionW](#) function
[GetEnhMetaFileHeader](#) function
[GetEnhMetaFilePaletteEntries](#) function
[GetEnhMetaFileW](#) function
[GetFontData](#) function
[GetFontLanguageInfo](#) function
[GetFontUnicodeRanges](#) function
[GetGlyphIndicesA](#) function
[GetGlyphIndicesW](#) function
[GetGlyphOutlineA](#) function
[GetGlyphOutlineW](#) function
[GetGraphicsMode](#) function
[GetGValue](#) macro
[GetKerningPairsA](#) function
[GetKerningPairsW](#) function
[GetLayout](#) function
[GetMapMode](#) function
[GetMetaFileA](#) function
[GetMetaFileBitsEx](#) function
[GetMetaFileW](#) function
[GetMetaRgn](#) function
[GetMiterLimit](#) function
[GetNearestColor](#) function
[GetNearestPaletteIndex](#) function
[GetObject](#) function
[GetObjectA](#) function
[GetObjectType](#) function
[GetObjectW](#) function
[GetOutlineTextMetricsA](#) function

[GetOutlineTextMetricsW function](#)

[GetPaletteEntries function](#)

[GetPath function](#)

[GetPixel function](#)

[GetPolyFillMode function](#)

[GetRandomRgn function](#)

[GetRasterizerCaps function](#)

[GetRegionData function](#)

[GetRgnBox function](#)

[GetROP2 function](#)

[GetRValue macro](#)

[GetStockObject function](#)

[GetStretchBltMode function](#)

[GetSystemPaletteEntries function](#)

[GetSystemPaletteUse function](#)

[GetTextAlign function](#)

[GetTextCharacterExtra function](#)

[GetTextColor function](#)

[GetTextExtentExPointA function](#)

[GetTextExtentExPointI function](#)

[GetTextExtentExPointW function](#)

[GetTextExtentPoint32A function](#)

[GetTextExtentPoint32W function](#)

[GetTextExtentPointA function](#)

[GetTextExtentPointI function](#)

[GetTextExtentPointW function](#)

[GetTextFaceA function](#)

[GetTextFaceW function](#)

[GetTextMetrics function](#)

[GetTextMetricsA function](#)

[GetTextMetricsW function](#)

[GetViewportExtEx function](#)

[GetViewportOrgEx](#) function
[GetWindowExtEx](#) function
[GetWindowOrgEx](#) function
[GetWinMetaFileBits](#) function
[GetWorldTransform](#) function
[GLYPHMETRICS](#) structure
[GLYPHSET](#) structure
[GOBJENUMPROC](#) callback function
[GRADIENT_RECT](#) structure
[GRADIENT_TRIANGLE](#) structure
[GradientFill](#) function
[HANDLETABLE](#) structure
[IntersectClipRect](#) function
[InvertRgn](#) function
[KERNINGPAIR](#) structure
[LineDDA](#) function
[LINEDDAPROC](#) callback function
[LineTo](#) function
[LOGBRUSH](#) structure
[LOGBRUSH32](#) structure
[LOGFONTA](#) structure
[LOGFONTW](#) structure
[LOGPALETTE](#) structure
[LOGPEN](#) structure
[LPtoDP](#) function
[MAKEPOINTS](#) macro
[MAKEROP4](#) macro
[MaskBlt](#) function
[MAT2](#) structure
[METAHEADER](#) structure
[METARECORD](#) structure
[MFENUMPROC](#) callback function

ModifyWorldTransform function
MoveToEx function
NEWTEXTMETRICA structure
NEWTEXTMETRICEXA structure
NEWTEXTMETRICEXW structure
NEWTEXTMETRICW structure
OffsetClipRgn function
OffsetRgn function
OffsetViewportOrgEx function
OffsetWindowOrgEx function
OUTLINETEXTMETRICA structure
OUTLINETEXTMETRICW structure
PaintRgn function
PALETTEINDEX macro
PALETERGB macro
PANOSE structure
PatBlt function
PathToRegion function
Pie function
PlayEnhMetaFile function
PlayEnhMetaFileRecord function
PlayMetaFile function
PlayMetaFileRecord function
PlgBlt function
POINTFX structure
PolyBezier function
PolyBezierTo function
PolyDraw function
Polygon function
Polyline function
PolylineTo function
PolyPolygon function

PolyPolyline function
POLYTEXTA structure
PolyTextOutA function
PolyTextOutW function
POLYTEXTW structure
PtInRegion function
PtVisible function
RASTERIZER_STATUS structure
RealizePalette function
Rectangle function
RectInRegion function
RectVisible function
RemoveFontMemResourceEx function
RemoveFontResourceA function
RemoveFontResourceExA function
RemoveFontResourceExW function
RemoveFontResourceW function
ResetDCA function
ResetDCW function
ResizePalette function
RestoreDC function
RGB macro
RGBQUAD structure
RGBTRIPLE structure
RGNDATA structure
RGNDATAHEADER structure
RoundRect function
SaveDC function
ScaleViewportExtEx function
ScaleWindowExtEx function
SelectClipPath function
SelectClipRgn function

[SelectObject function](#)
[SelectPalette function](#)
[SetArcDirection function](#)
[SetBitmapBits function](#)
[SetBitmapDimensionEx function](#)
[SetBkColor function](#)
[SetBkMode function](#)
[SetBoundsRect function](#)
[SetBrushOrgEx function](#)
[SetColorAdjustment function](#)
[SetDCBrushColor function](#)
[SetDCPenColor function](#)
[SetDIBColorTable function](#)
[SetDIBits function](#)
[SetDIBitsToDevice function](#)
[SetEnhMetaFileBits function](#)
[SetGraphicsMode function](#)
[SetLayout function](#)
[SetMapMode function](#)
[SetMapperFlags function](#)
[SetMetaFileBitsEx function](#)
[SetMetaRgn function](#)
[SetMiterLimit function](#)
[SetPaletteEntries function](#)
[SetPixel function](#)
[SetPixelV function](#)
[SetPolyFillMode function](#)
[SetRectRgn function](#)
[SetROP2 function](#)
[SetStretchBltMode function](#)
[SetSystemPaletteUse function](#)
[SetTextAlign function](#)

[SetTextCharacterExtra function](#)
[SetTextColor function](#)
[SetTextJustification function](#)
[SetViewportExtEx function](#)
[SetViewportOrgEx function](#)
[SetWindowExtEx function](#)
[SetWindowOrgEx function](#)
[SetWinMetaFileBits function](#)
[SetWorldTransform function](#)
[StretchBlt function](#)
[StretchDIBits function](#)
[StrokeAndFillPath function](#)
[StrokePath function](#)
[TEXTMETRICA structure](#)
[TEXTMETRICW structure](#)
[TextOutA function](#)
[TextOutW function](#)
[TransparentBlt function](#)
[TRIVERTEX structure](#)
[TTPOLYCURVE structure](#)
[TTPOLYGONHEADER structure](#)
[UnrealizeObject function](#)
[UpdateColors function](#)
[WCRANGE structure](#)
[WidenPath function](#)
[XFORM structure](#)

[Winuser.h](#)

[Overview](#)

[BeginPaint function](#)
[ChangeDisplaySettingsA function](#)
[ChangeDisplaySettingsExA function](#)
[ChangeDisplaySettingsExW function](#)

[ChangeDisplaySettingsW](#) function
[ClientToScreen](#) function
[CopyRect](#) function
[DrawAnimatedRects](#) function
[DrawCaption](#) function
[DrawEdge](#) function
[DrawFocusRect](#) function
[DrawFrameControl](#) function
[DrawStateA](#) function
[DRAWSTATEPROC](#) callback function
[DrawStateW](#) function
[DrawText](#) function
[DrawTextA](#) function
[DrawTextExA](#) function
[DrawTextExW](#) function
[DRAWTEXTPARAMS](#) structure
[DrawTextW](#) function
[EndPaint](#) function
[EnumDisplayDevicesA](#) function
[EnumDisplayDevicesW](#) function
[EnumDisplayMonitors](#) function
[EnumDisplaySettingsA](#) function
[EnumDisplaySettingsExA](#) function
[EnumDisplaySettingsExW](#) function
[EnumDisplaySettingsW](#) function
[EqualRect](#) function
[ExcludeUpdateRgn](#) function
[FillRect](#) function
[FrameRect](#) function
[GetDC](#) function
[GetDCEx](#) function
[GetMonitorInfoA](#) function

[GetMonitorInfoW](#) function
[GetSysColorBrush](#) function
[GetTabbedTextExtentA](#) function
[GetTabbedTextExtentW](#) function
[GetUpdateRect](#) function
[GetUpdateRgn](#) function
[GetWindowDC](#) function
[GetWindowRgn](#) function
[GetWindowRgnBox](#) function
[GrayStringA](#) function
[GRAYSTRINGPROC](#) callback function
[GrayStringW](#) function
[InflateRect](#) function
[IntersectRect](#) function
[InvalidateRect](#) function
[InvalidateRgn](#) function
[InvertRect](#) function
[IsRectEmpty](#) function
[LoadBitmapA](#) function
[LoadBitmapW](#) function
[LockWindowUpdate](#) function
[MapWindowPoints](#) function
[MONITORENUMPROC](#) callback function
[MonitorFromPoint](#) function
[MonitorFromRect](#) function
[MonitorFromWindow](#) function
[MONITORINFO](#) structure
[MONITORINFOEXA](#) structure
[MONITORINFOEXW](#) structure
[OffsetRect](#) function
[PaintDesktop](#) function
[PAINTSTRUCT](#) structure

[POINTSTOPOINT macro](#)

[POINTTOPOINTS macro](#)

[PtInRect function](#)

[RedrawWindow function](#)

[ReleaseDC function](#)

[ScreenToClient function](#)

[SetRect function](#)

[SetRectEmpty function](#)

[SetWindowRgn function](#)

[SubtractRect function](#)

[TabbedTextOutA function](#)

[TabbedTextOutW function](#)

[UnionRect function](#)

[UpdateWindow function](#)

[ValidateRect function](#)

[ValidateRgn function](#)

[WindowFromDC function](#)

[Xpsprint.h](#)

[Overview](#)

Windows GDI

4/21/2022 • 55 minutes to read • [Edit Online](#)

Overview of the Windows GDI technology.

To develop Windows GDI, you need these headers:

- [fontsub.h](#)
- [mmsystem.h](#)
- [prnasnot.h](#)
- [prntvpt.h](#)
- [t2embapi.h](#)
- [tvout.h](#)
- [windef.h](#)
- [xpsprint.h](#)

For programming guidance for this technology, see:

- [Windows GDI](#)
- [Documents and Printing](#)

Functions

[AbortPath](#)

The AbortPath function closes and discards any paths in the specified device context.

[AddFontMemResourceEx](#)

The AddFontMemResourceEx function adds the font resource from a memory image to the system.

[AddFontResourceA](#)

The AddFontResource function adds the font resource from the specified file to the system font table. The font can subsequently be used for text output by any application.

[AddFontResourceExA](#)

The AddFontResourceEx function adds the font resource from the specified file to the system. Fonts added with the AddFontResourceEx function can be marked as private and not enumerable.

[AddFontResourceExW](#)

The AddFontResourceEx function adds the font resource from the specified file to the system. Fonts added with the AddFontResourceEx function can be marked as private and not enumerable.

[AddFontResourceW](#)

The AddFontResource function adds the font resource from the specified file to the system font table. The font can subsequently be used for text output by any application.

[AlphaBlend](#)

The AlphaBlend function displays bitmaps that have transparent or semitransparent pixels.

[AngleArc](#)

The AngleArc function draws a line segment and an arc.

[AnimatePalette](#)

The AnimatePalette function replaces entries in the specified logical palette.

[Arc](#)

The Arc function draws an elliptical arc.

[ArcTo](#)

The ArcTo function draws an elliptical arc.

[BeginPaint](#)

The BeginPaint function prepares the specified window for painting and fills a PAINTSTRUCT structure with information about the painting.

[BeginPath](#)

The BeginPath function opens a path bracket in the specified device context.

[BitBlt](#)

The BitBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

[CancelDC](#)

The CancelDC function cancels any pending operation on the specified device context (DC).

[CFP_ALLOCPROC](#)

Client-provided callback function, used by CreateFontPackage and MergeFontPackage to allocate memory.

[CFP_FREEPROC](#)

Client-provided callback function, used by CreateFontPackage and MergeFontPackage to free memory.

[CFP_REALLOCPROC](#)

Client-provided callback function, used by CreateFontPackage and MergeFontPackage to reallocate memory when the size of an allocated buffer needs to change.

[ChangeDisplaySettingsA](#)

The ChangeDisplaySettings function changes the settings of the default display device to the specified graphics mode.

[ChangeDisplaySettingsExA](#)

The ChangeDisplaySettingsEx function changes the settings of the specified display device to the specified graphics mode.

[ChangeDisplaySettingsExW](#)

The ChangeDisplaySettingsEx function changes the settings of the specified display device to the specified graphics mode.

[ChangeDisplaySettingsW](#)

The ChangeDisplaySettings function changes the settings of the default display device to the specified graphics mode.

[Chord](#)

The Chord function draws a chord (a region bounded by the intersection of an ellipse and a line segment, called a secant). The chord is outlined by using the current pen and filled by using the current brush.

[ClientToScreen](#)

The ClientToScreen function converts the client-area coordinates of a specified point to screen coordinates.

[CloseEnhMetaFile](#)

The CloseEnhMetaFile function closes an enhanced-metafile device context and returns a handle that identifies an enhanced-format metafile.

[CloseFigure](#)

The CloseFigure function closes an open figure in a path.

[CloseMetaFile](#)

The CloseMetaFile function closes a metafile device context and returns a handle that identifies a Windows-format metafile.

[CombineRgn](#)

The CombineRgn function combines two regions and stores the result in a third region. The two regions are combined according to the specified mode.

[CombineTransform](#)

The CombineTransform function concatenates two world-space to page-space transformations.

[CopyEnhMetaFileA](#)

The CopyEnhMetaFile function copies the contents of an enhanced-format metafile to a specified file.

[CopyEnhMetaFileW](#)

The CopyEnhMetaFile function copies the contents of an enhanced-format metafile to a specified file.

[CopyMetaFileA](#)

The CopyMetaFile function copies the content of a Windows-format metafile to the specified file.

[CopyMetaFileW](#)

The `CopyMetaFile` function copies the content of a Windows-format metafile to the specified file.

[CopyRect](#)

The `CopyRect` function copies the coordinates of one rectangle to another.

[CreateBitmap](#)

The `CreateBitmap` function creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel).

[CreateBitmapIndirect](#)

The `CreateBitmapIndirect` function creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel).

[CreateBrushIndirect](#)

The `CreateBrushIndirect` function creates a logical brush that has the specified style, color, and pattern.

[CreateCompatibleBitmap](#)

The `CreateCompatibleBitmap` function creates a bitmap compatible with the device that is associated with the specified device context.

[CreateCompatibleDC](#)

The `CreateCompatibleDC` function creates a memory device context (DC) compatible with the specified device.

[CreateDCA](#)

The `CreateDC` function creates a device context (DC) for a device using the specified name.

[CreateDCW](#)

The `CreateDC` function creates a device context (DC) for a device using the specified name.

[CreateDIBitmap](#)

The `CreateDIBitmap` function creates a compatible bitmap (DDB) from a DIB and, optionally, sets the bitmap bits.

[CreateDIBPatternBrush](#)

The `CreateDIBPatternBrush` function creates a logical brush that has the pattern specified by the specified device-independent bitmap (DIB).

[CreateDIBPatternBrushPt](#)

The `CreateDIBPatternBrushPt` function creates a logical brush that has the pattern specified by the device-independent bitmap (DIB).

[CreateDIBSection](#)

The `CreateDIBSection` function creates a DIB that applications can write to directly.

[CreateDiscardableBitmap](#)

The CreateDiscardableBitmap function creates a discardable bitmap that is compatible with the specified device.

[CreateEllipticRgn](#)

The CreateEllipticRgn function creates an elliptical region.

[CreateEllipticRgnIndirect](#)

The CreateEllipticRgnIndirect function creates an elliptical region.

[CreateEnhMetaFileA](#)

The CreateEnhMetaFile function creates a device context for an enhanced-format metafile. This device context can be used to store a device-independent picture.

[CreateEnhMetaFileW](#)

The CreateEnhMetaFile function creates a device context for an enhanced-format metafile. This device context can be used to store a device-independent picture.

[CreateFontA](#)

The CreateFont function creates a logical font with the specified characteristics. The logical font can subsequently be selected as the font for any device.

[CreateFontIndirectA](#)

The CreateFontIndirect function creates a logical font that has the specified characteristics. The font can subsequently be selected as the current font for any device context.

[CreateFontIndirectExA](#)

The CreateFontIndirectEx function specifies a logical font that has the characteristics in the specified structure. The font can subsequently be selected as the current font for any device context.

[CreateFontIndirectExW](#)

The CreateFontIndirectEx function specifies a logical font that has the characteristics in the specified structure. The font can subsequently be selected as the current font for any device context.

[CreateFontIndirectW](#)

The CreateFontIndirect function creates a logical font that has the specified characteristics. The font can subsequently be selected as the current font for any device context.

[CreateFontPackage](#)

The CreateFontPackage function creates a subset version of a specified TrueType font, typically in order to pass it to a printer.

[CreateFontW](#)

The CreateFont function creates a logical font with the specified characteristics. The logical font can subsequently be selected as the font for any device.

[CreateHalftonePalette](#)

The CreateHalftonePalette function creates a halftone palette for the specified device context (DC).

[CreateHatchBrush](#)

The CreateHatchBrush function creates a logical brush that has the specified hatch pattern and color.

[CreateIC](#)

The CreateIC function creates an information context for the specified device.

[CreateICW](#)

The CreateIC function creates an information context for the specified device.

[CreateMetaFileA](#)

The CreateMetaFile function creates a device context for a Windows-format metafile.

[CreateMetaFileW](#)

The CreateMetaFile function creates a device context for a Windows-format metafile.

[CreatePalette](#)

The CreatePalette function creates a logical palette.

[CreatePatternBrush](#)

The CreatePatternBrush function creates a logical brush with the specified bitmap pattern. The bitmap can be a DIB section bitmap, which is created by the CreateDIBSection function, or it can be a device-dependent bitmap.

[CreatePen](#)

The CreatePen function creates a logical pen that has the specified style, width, and color. The pen can subsequently be selected into a device context and used to draw lines and curves.

[CreatePenIndirect](#)

The CreatePenIndirect function creates a logical cosmetic pen that has the style, width, and color specified in a structure.

[CreatePolygonRgn](#)

The CreatePolygonRgn function creates a polygonal region.

[CreatePolyPolygonRgn](#)

The CreatePolyPolygonRgn function creates a region consisting of a series of polygons. The polygons can overlap.

[CreateRectRgn](#)

The CreateRectRgn function creates a rectangular region.

[CreateRectRgnIndirect](#)

The CreateRectRgnIndirect function creates a rectangular region.

[CreateRoundRectRgn](#)

The CreateRoundRectRgn function creates a rectangular region with rounded corners.

[CreateScalableFontResourceA](#)

The CreateScalableFontResource function creates a font resource file for a scalable font.

[CreateScalableFontResourceW](#)

The CreateScalableFontResource function creates a font resource file for a scalable font.

[CreateSolidBrush](#)

The CreateSolidBrush function creates a logical brush that has the specified solid color.

[DeleteDC](#)

The DeleteDC function deletes the specified device context (DC).

[DeleteEnhMetaFile](#)

The DeleteEnhMetaFile function deletes an enhanced-format metafile or an enhanced-format metafile handle.

[DeleteFont](#)

The DeleteFont macro deletes a font object, freeing all system resources associated with the font object.

[DeleteMetaFile](#)

The DeleteMetaFile function deletes a Windows-format metafile or Windows-format metafile handle.

[DeleteObject](#)

The DeleteObject function deletes a logical pen, brush, font, bitmap, region, or palette, freeing all system resources associated with the object. After the object is deleted, the specified handle is no longer valid.

[DIBINDEX](#)

The DIBINDEX macro takes an index to an entry in a DIB color table and returns a COLORREF value that specifies the color associated with the given index.

[DPtoLP](#)

The DPtoLP function converts device coordinates into logical coordinates. The conversion depends on the mapping mode of the device context, the settings of the origins and extents for the window and viewport, and the world transformation.

[DrawAnimatedRects](#)

Animates the caption of a window to indicate the opening of an icon or the minimizing or maximizing of a window.

[DrawCaption](#)

The DrawCaption function draws a window caption.

[DrawEdge](#)

The DrawEdge function draws one or more edges of rectangle.

[DrawEscape](#)

The DrawEscape function provides drawing capabilities of the specified video display that are not directly available through the graphics device interface (GDI).

[DrawFocusRect](#)

The DrawFocusRect function draws a rectangle in the style used to indicate that the rectangle has the focus.

[DrawFrameControl](#)

The DrawFrameControl function draws a frame control of the specified type and style.

[DrawStateA](#)

The DrawState function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.

[DRAWSTATEPROC](#)

The DrawStateProc function is an application-defined callback function that renders a complex image for the DrawState function.

[DrawStateW](#)

The DrawState function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.

[DrawText](#)

The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

[DrawTextA](#)

The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

[DrawTextExA](#)

The DrawTextEx function draws formatted text in the specified rectangle.

[DrawTextExW](#)

The DrawTextEx function draws formatted text in the specified rectangle.

[DrawTextW](#)

The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

[Ellipse](#)

The Ellipse function draws an ellipse. The center of the ellipse is the center of the specified bounding rectangle. The ellipse is outlined by using the current pen and is filled by using the current brush.

[EndPaint](#)

The EndPaint function marks the end of painting in the specified window. This function is required for each call to the BeginPaint function, but only after painting is complete.

[EndPath](#)

The EndPath function closes a path bracket and selects the path defined by the bracket into the specified device context.

[ENHMFENUMPROC](#)

The EnhMetaFileProc function is an application-defined callback function used with the EnumEnhMetaFile function.

[EnumDisplayDevicesA](#)

The EnumDisplayDevices function lets you obtain information about the display devices in the current session.

[EnumDisplayDevicesW](#)

The EnumDisplayDevices function lets you obtain information about the display devices in the current session.

[EnumDisplayMonitors](#)

The EnumDisplayMonitors function enumerates display monitors (including invisible pseudo-monitors associated with the mirroring drivers) that intersect a region formed by the intersection of a specified clipping rectangle and the visible region of a device context. EnumDisplayMonitors calls an application-defined MonitorEnumProc callback function once for each monitor that is enumerated. Note that GetSystemMetrics (SM_CMONITORS) counts only the display monitors.

[EnumDisplaySettingsA](#)

The EnumDisplaySettings function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.

[EnumDisplaySettingsExA](#)

The EnumDisplaySettingsEx function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.

[EnumDisplaySettingsExW](#)

The EnumDisplaySettingsEx function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.

[EnumDisplaySettingsW](#)

The EnumDisplaySettings function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.

[EnumEnhMetaFile](#)

The `EnumEnhMetaFile` function enumerates the records within an enhanced-format metafile by retrieving each record and passing it to the specified callback function.

[EnumFontFamiliesA](#)

The `EnumFontFamilies` function enumerates the fonts in a specified font family that are available on a specified device.

[EnumFontFamiliesExA](#)

The `EnumFontFamiliesEx` function enumerates all uniquely-named fonts in the system that match the font characteristics specified by the `LOGFONT` structure. `EnumFontFamiliesEx` enumerates fonts based on typeface name, character set, or both.

[EnumFontFamiliesExW](#)

The `EnumFontFamiliesEx` function enumerates all uniquely-named fonts in the system that match the font characteristics specified by the `LOGFONT` structure. `EnumFontFamiliesEx` enumerates fonts based on typeface name, character set, or both.

[EnumFontFamiliesW](#)

The `EnumFontFamilies` function enumerates the fonts in a specified font family that are available on a specified device.

[EnumFontsA](#)

The `EnumFonts` function enumerates the fonts available on a specified device.

[EnumFontsW](#)

The `EnumFonts` function enumerates the fonts available on a specified device.

[EnumMetaFile](#)

The `EnumMetaFile` function enumerates the records within a Windows-format metafile by retrieving each record and passing it to the specified callback function.

[EnumObjects](#)

The `EnumObjects` function enumerates the pens or brushes available for the specified device context (DC).

[EqualRect](#)

The `EqualRect` function determines whether the two specified rectangles are equal by comparing the coordinates of their upper-left and lower-right corners.

[EqualRgn](#)

The `EqualRgn` function checks the two specified regions to determine whether they are identical. The function considers two regions identical if they are equal in size and shape.

[ExcludeClipRect](#)

The `ExcludeClipRect` function creates a new clipping region that consists of the existing clipping region minus the specified rectangle.

[ExcludeUpdateRgn](#)

The ExcludeUpdateRgn function prevents drawing within invalid areas of a window by excluding an updated region in the window from a clipping region.

[ExtCreatePen](#)

The ExtCreatePen function creates a logical cosmetic or geometric pen that has the specified style, width, and brush attributes.

[ExtCreateRegion](#)

The ExtCreateRegion function creates a region from the specified region and transformation data.

[ExtFloodFill](#)

The ExtFloodFill function fills an area of the display surface with the current brush.

[ExtSelectClipRgn](#)

The ExtSelectClipRgn function combines the specified region with the current clipping region using the specified mode.

[ExtTextOutA](#)

The ExtTextOut function draws text using the currently selected font, background color, and text color. You can optionally provide dimensions to be used for clipping, opaquing, or both.

[ExtTextOutW](#)

The ExtTextOut function draws text using the currently selected font, background color, and text color. You can optionally provide dimensions to be used for clipping, opaquing, or both.

[FillPath](#)

The FillPath function closes any open figures in the current path and fills the path's interior by using the current brush and polygon-filling mode.

[FillRect](#)

The FillRect function fills a rectangle by using the specified brush. This function includes the left and top borders, but excludes the right and bottom borders of the rectangle.

[FillRgn](#)

The FillRgn function fills a region by using the specified brush.

[FlattenPath](#)

The FlattenPath function transforms any curves in the path that is selected into the current device context (DC), turning each curve into a sequence of lines.

[FloodFill](#)

The FloodFill function fills an area of the display surface with the current brush. The area is assumed to be bounded as specified by the color parameter.

[FrameRect](#)

The FrameRect function draws a border around the specified rectangle by using the specified brush. The width and height of the border are always one logical unit.

[FrameRgn](#)

The FrameRgn function draws a border around the specified region by using the specified brush.

[GdiAlphaBlend](#)

The GdiAlphaBlend function displays bitmaps that have transparent or semitransparent pixels.

[GdiComment](#)

The GdiComment function copies a comment from a buffer into a specified enhanced-format metafile.

[GdiFlush](#)

The GdiFlush function flushes the calling thread's current batch.

[GdiGetBatchLimit](#)

The GdiGetBatchLimit function returns the maximum number of function calls that can be accumulated in the calling thread's current batch. The system flushes the current batch whenever this limit is exceeded.

[GdiGradientFill](#)

The GdiGradientFill function fills rectangle and triangle structures.

[GdiSetBatchLimit](#)

The GdiSetBatchLimit function sets the maximum number of function calls that can be accumulated in the calling thread's current batch. The system flushes the current batch whenever this limit is exceeded.

[GdiTransparentBlt](#)

The GdiTransparentBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

[GetArcDirection](#)

The GetArcDirection function retrieves the current arc direction for the specified device context. Arc and rectangle functions use the arc direction.

[GetAspectRatioFilterEx](#)

The GetAspectRatioFilterEx function retrieves the setting for the current aspect-ratio filter.

[GetBitmapBits](#)

The GetBitmapBits function copies the bitmap bits of a specified device-dependent bitmap into a buffer.

[GetBitmapDimensionEx](#)

The GetBitmapDimensionEx function retrieves the dimensions of a compatible bitmap. The retrieved dimensions must have been set by the SetBitmapDimensionEx function.

[GetBkColor](#)

The GetBkColor function returns the current background color for the specified device context.

[GetBkMode](#)

The GetBkMode function returns the current background mix mode for a specified device context. The background mix mode of a device context affects text, hatched brushes, and pen styles that are not solid lines.

[GetBoundsRect](#)

The GetBoundsRect function obtains the current accumulated bounding rectangle for a specified device context.

[GetBrushOrgEx](#)

The GetBrushOrgEx function retrieves the current brush origin for the specified device context. This function replaces the GetBrushOrg function.

[GetValue](#)

The GetValue macro retrieves an intensity value for the blue component of a red, green, blue (RGB) value.

[GetCharABCWidthsA](#)

The GetCharABCWidths function retrieves the widths, in logical units, of consecutive characters in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.

[GetCharABCWidthsFloatA](#)

The GetCharABCWidthsFloat function retrieves the widths, in logical units, of consecutive characters in a specified range from the current font.

[GetCharABCWidthsFloatW](#)

The GetCharABCWidthsFloat function retrieves the widths, in logical units, of consecutive characters in a specified range from the current font.

[GetCharABCWidthsI](#)

The GetCharABCWidthsI function retrieves the widths, in logical units, of consecutive glyph indices in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.

[GetCharABCWidthsW](#)

The GetCharABCWidths function retrieves the widths, in logical units, of consecutive characters in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.

[GetCharacterPlacementA](#)

The GetCharacterPlacement function retrieves information about a character string, such as character widths, caret positioning, ordering within the string, and glyph rendering.

[GetCharacterPlacementW](#)

The GetCharacterPlacement function retrieves information about a character string, such as character widths, caret positioning, ordering within the string, and glyph rendering.

[GetCharWidth32A](#)

The GetCharWidth32 function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

[GetCharWidth32W](#)

The GetCharWidth32 function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

[GetCharWidthA](#)

The GetCharWidth function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

[GetCharWidthFloatA](#)

The GetCharWidthFloat function retrieves the fractional widths of consecutive characters in a specified range from the current font.

[GetCharWidthFloatW](#)

The GetCharWidthFloat function retrieves the fractional widths of consecutive characters in a specified range from the current font.

[GetCharWidthI](#)

The GetCharWidthI function retrieves the widths, in logical coordinates, of consecutive glyph indices in a specified range from the current font.

[GetCharWidthW](#)

The GetCharWidth function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

[GetClipBox](#)

The GetClipBox function retrieves the dimensions of the tightest bounding rectangle that can be drawn around the current visible area on the device.

[GetClipRgn](#)

The GetClipRgn function retrieves a handle identifying the current application-defined clipping region for the specified device context.

[GetColorAdjustment](#)

The GetColorAdjustment function retrieves the color adjustment values for the specified device context (DC).

[GetCurrentObject](#)

The `GetCurrentObject` function retrieves a handle to an object of the specified type that has been selected into the specified device context (DC).

[GetCurrentPositionEx](#)

The `GetCurrentPositionEx` function retrieves the current position in logical coordinates.

[GetDC](#)

The `GetDC` function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen.

[GetDCBrushColor](#)

The `GetDCBrushColor` function retrieves the current brush color for the specified device context (DC).

[GetDCEx](#)

The `GetDCEx` function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen.

[GetDCOrgEx](#)

The `GetDCOrgEx` function retrieves the final translation origin for a specified device context (DC).

[GetDCPenColor](#)

The `GetDCPenColor` function retrieves the current pen color for the specified device context (DC).

[GetDeviceCaps](#)

The `GetDeviceCaps` function retrieves device-specific information for the specified device.

[GetDIBColorTable](#)

The `GetDIBColorTable` function retrieves RGB (red, green, blue) color values from a range of entries in the color table of the DIB section bitmap that is currently selected into a specified device context.

[GetDIBits](#)

The `GetDIBits` function retrieves the bits of the specified compatible bitmap and copies them into a buffer as a DIB using the specified format.

[GetEnhMetaFileA](#)

The `GetEnhMetaFile` function creates a handle that identifies the enhanced-format metafile stored in the specified file.

[GetEnhMetaFileBits](#)

The `GetEnhMetaFileBits` function retrieves the contents of the specified enhanced-format metafile and copies them into a buffer.

[GetEnhMetaFileDescriptionA](#)

The GetEnhMetaFileDescription function retrieves an optional text description from an enhanced-format metafile and copies the string to the specified buffer.

[GetEnhMetaFileDescriptionW](#)

The GetEnhMetaFileDescription function retrieves an optional text description from an enhanced-format metafile and copies the string to the specified buffer.

[GetEnhMetaFileHeader](#)

The GetEnhMetaFileHeader function retrieves the record containing the header for the specified enhanced-format metafile.

[GetEnhMetaFilePaletteEntries](#)

The GetEnhMetaFilePaletteEntries function retrieves optional palette entries from the specified enhanced metafile.

[GetEnhMetaFileW](#)

The GetEnhMetaFile function creates a handle that identifies the enhanced-format metafile stored in the specified file.

[GetFontData](#)

The GetFontData function retrieves font metric data for a TrueType font.

[GetFontLanguageInfo](#)

The GetFontLanguageInfo function returns information about the currently selected font for the specified display context. Applications typically use this information and the GetCharacterPlacement function to prepare a character string for display.

[GetFontUnicodeRanges](#)

The GetFontUnicodeRanges function returns information about which Unicode characters are supported by a font. The information is returned as a GLYPHSET structure.

[GetGlyphIndicesA](#)

The GetGlyphIndices function translates a string into an array of glyph indices. The function can be used to determine whether a glyph exists in a font.

[GetGlyphIndicesW](#)

The GetGlyphIndices function translates a string into an array of glyph indices. The function can be used to determine whether a glyph exists in a font.

[GetGlyphOutlineA](#)

The GetGlyphOutline function retrieves the outline or bitmap for a character in the TrueType font that is selected into the specified device context.

[GetGlyphOutlineW](#)

The GetGlyphOutline function retrieves the outline or bitmap for a character in the TrueType font that is selected into the specified device context.

[GetGraphicsMode](#)

The GetGraphicsMode function retrieves the current graphics mode for the specified device context.

[GetGValue](#)

The GetGValue macro retrieves an intensity value for the green component of a red, green, blue (RGB) value.

[GetKerningPairsA](#)

The GetKerningPairs function retrieves the character-kerning pairs for the currently selected font for the specified device context.

[GetKerningPairsW](#)

The GetKerningPairs function retrieves the character-kerning pairs for the currently selected font for the specified device context.

[GetLayout](#)

The GetLayout function returns the layout of a device context (DC).

[GetMapMode](#)

The GetMapMode function retrieves the current mapping mode.

[GetMetaFileA](#)

The GetMetaFile function creates a handle that identifies the metafile stored in the specified file.

[GetMetaFileBitsEx](#)

The GetMetaFileBitsEx function retrieves the contents of a Windows-format metafile and copies them into the specified buffer.

[GetMetaFileW](#)

The GetMetaFile function creates a handle that identifies the metafile stored in the specified file.

[GetMetaRgn](#)

The GetMetaRgn function retrieves the current metaregion for the specified device context.

[GetMiterLimit](#)

The GetMiterLimit function retrieves the miter limit for the specified device context.

[GetMonitorInfoA](#)

The GetMonitorInfo function retrieves information about a display monitor.

[GetMonitorInfoW](#)

The GetMonitorInfo function retrieves information about a display monitor.

[GetNearestColor](#)

The GetNearestColor function retrieves a color value identifying a color from the system palette that will be displayed when the specified color value is used.

[GetNearestPaletteIndex](#)

The GetNearestPaletteIndex function retrieves the index for the entry in the specified logical palette most closely matching a specified color value.

[GetObject](#)

The GetObject function retrieves information for the specified graphics object.

[GetObjectA](#)

The GetObject function retrieves information for the specified graphics object.

[GetObjectType](#)

The GetObjectType retrieves the type of the specified object.

[GetObjectW](#)

The GetObject function retrieves information for the specified graphics object.

[GetOutlineTextMetricsA](#)

The GetOutlineTextMetrics function retrieves text metrics for TrueType fonts.

[GetOutlineTextMetricsW](#)

The GetOutlineTextMetrics function retrieves text metrics for TrueType fonts.

[GetPaletteEntries](#)

The GetPaletteEntries function retrieves a specified range of palette entries from the given logical palette.

[GetPath](#)

The GetPath function retrieves the coordinates defining the endpoints of lines and the control points of curves found in the path that is selected into the specified device context.

[GetPixel](#)

The GetPixel function retrieves the red, green, blue (RGB) color value of the pixel at the specified coordinates.

[GetPolyFillMode](#)

The GetPolyFillMode function retrieves the current polygon fill mode.

[GetRandomRgn](#)

The GetRandomRgn function copies the system clipping region of a specified device context to a specific region.

[GetRasterizerCaps](#)

The GetRasterizerCaps function returns flags indicating whether TrueType fonts are installed in the system.

[GetRegionData](#)

The GetRegionData function fills the specified buffer with data describing a region. This data includes the dimensions of the rectangles that make up the region.

[GetRgnBox](#)

The GetRgnBox function retrieves the bounding rectangle of the specified region.

[GetROP2](#)

The GetROP2 function retrieves the foreground mix mode of the specified device context. The mix mode specifies how the pen or interior color and the color already on the screen are combined to yield a new color.

[GetRValue](#)

The GetRValue macro retrieves an intensity value for the red component of a red, green, blue (RGB) value.

[GetStockObject](#)

The GetStockObject function retrieves a handle to one of the stock pens, brushes, fonts, or palettes.

[GetStretchBltMode](#)

The GetStretchBltMode function retrieves the current stretching mode. The stretching mode defines how color data is added to or removed from bitmaps that are stretched or compressed when the StretchBlt function is called.

[GetSysColorBrush](#)

The GetSysColorBrush function retrieves a handle identifying a logical brush that corresponds to the specified color index.

[GetSystemPaletteEntries](#)

The GetSystemPaletteEntries function retrieves a range of palette entries from the system palette that is associated with the specified device context (DC).

[GetSystemPaletteUse](#)

The GetSystemPaletteUse function retrieves the current state of the system (physical) palette for the specified device context (DC).

[GetTabbedTextExtentA](#)

The GetTabbedTextExtent function computes the width and height of a character string.

[GetTabbedTextExtentW](#)

The GetTabbedTextExtent function computes the width and height of a character string.

[GetTextAlign](#)

The GetTextAlign function retrieves the text-alignment setting for the specified device context.

[GetTextCharacterExtra](#)

The `GetTextCharacterExtra` function retrieves the current intercharacter spacing for the specified device context.

[GetTextColor](#)

The `GetTextColor` function retrieves the current text color for the specified device context.

[GetTextExtentExPointA](#)

The `GetTextExtentExPoint` function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.

[GetTextExtentExPointI](#)

The `GetTextExtentExPointI` function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.

[GetTextExtentExPointW](#)

The `GetTextExtentExPoint` function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.

[GetTextExtentPoint32A](#)

The `GetTextExtentPoint32` function computes the width and height of the specified string of text.

[GetTextExtentPoint32W](#)

The `GetTextExtentPoint32` function computes the width and height of the specified string of text.

[GetTextExtentPointA](#)

The `GetTextExtentPoint` function computes the width and height of the specified string of text.

[GetTextExtentPointI](#)

The `GetTextExtentPointI` function computes the width and height of the specified array of glyph indices.

[GetTextExtentPointW](#)

The `GetTextExtentPoint` function computes the width and height of the specified string of text.

[GetTextFaceA](#)

The `GetTextFace` function retrieves the typeface name of the font that is selected into the specified device context.

[GetTextFaceW](#)

The `GetTextFace` function retrieves the typeface name of the font that is selected into the specified device context.

[GetTextMetrics](#)

The `GetTextMetrics` function fills the specified buffer with the metrics for the currently selected font.

[GetTextMetricsA](#)

The `GetTextMetrics` function fills the specified buffer with the metrics for the currently selected font.

[GetTextMetricsW](#)

The `GetTextMetrics` function fills the specified buffer with the metrics for the currently selected font.

[GetUpdateRect](#)

The `GetUpdateRect` function retrieves the coordinates of the smallest rectangle that completely encloses the update region of the specified window.

[GetUpdateRgn](#)

The `GetUpdateRgn` function retrieves the update region of a window by copying it into the specified region. The coordinates of the update region are relative to the upper-left corner of the window (that is, they are client coordinates).

[GetViewportExtEx](#)

The `GetViewportExtEx` function retrieves the x-extent and y-extent of the current viewport for the specified device context.

[GetViewportOrgEx](#)

The `GetViewportOrgEx` function retrieves the x-coordinates and y-coordinates of the viewport origin for the specified device context.

[GetWindowDC](#)

The `GetWindowDC` function retrieves the device context (DC) for the entire window, including title bar, menus, and scroll bars.

[GetWindowExtEx](#)

This function retrieves the x-extent and y-extent of the window for the specified device context.

[GetWindowOrgEx](#)

The `GetWindowOrgEx` function retrieves the x-coordinates and y-coordinates of the window origin for the specified device context.

[GetWindowRgn](#)

The `GetWindowRgn` function obtains a copy of the window region of a window.

[GetWindowRgnBox](#)

The `GetWindowRgnBox` function retrieves the dimensions of the tightest bounding rectangle for the window region of a window.

[GetWinMetaFileBits](#)

The `GetWinMetaFileBits` function converts the enhanced-format records from a metafile into Windows-format records and stores the converted records in the specified buffer.

[GetWorldTransform](#)

The GetWorldTransform function retrieves the current world-space to page-space transformation.

[GOBJENUMPROC](#)

The EnumObjectsProc function is an application-defined callback function used with the EnumObjects function.

[GradientFill](#)

The GradientFill function fills rectangle and triangle structures.

[GrayStringA](#)

The GrayString function draws gray text at the specified location.

[GRAYSTRINGPROC](#)

The OutputProc function is an application-defined callback function used with the GrayString function.

[GrayStringW](#)

The GrayString function draws gray text at the specified location.

[InflateRect](#)

The InflateRect function increases or decreases the width and height of the specified rectangle.

[IntersectClipRect](#)

The IntersectClipRect function creates a new clipping region from the intersection of the current clipping region and the specified rectangle.

[IntersectRect](#)

The IntersectRect function calculates the intersection of two source rectangles and places the coordinates of the intersection rectangle into the destination rectangle.

[InvalidateRect](#)

The InvalidateRect function adds a rectangle to the specified window's update region. The update region represents the portion of the window's client area that must be redrawn.

[InvalidateRgn](#)

The InvalidateRgn function invalidates the client area within the specified region by adding it to the current update region of a window.

[InvertRect](#)

The InvertRect function inverts a rectangle in a window by performing a logical NOT operation on the color values for each pixel in the rectangle's interior.

[InvertRgn](#)

The InvertRgn function inverts the colors in the specified region.

[IsRectEmpty](#)

The IsRectEmpty function determines whether the specified rectangle is empty.

[LineDDA](#)

The LineDDA function determines which pixels should be highlighted for a line defined by the specified starting and ending points.

[LINEDDAPROC](#)

The LineDDAProc function is an application-defined callback function used with the LineDDA function.

[LineTo](#)

The LineTo function draws a line from the current position up to, but not including, the specified point.

[LoadBitmapA](#)

The LoadBitmap function loads the specified bitmap resource from a module's executable file.

[LoadBitmapW](#)

The LoadBitmap function loads the specified bitmap resource from a module's executable file.

[LockWindowUpdate](#)

The LockWindowUpdate function disables or enables drawing in the specified window. Only one window can be locked at a time.

[LPtoDP](#)

The LPtoDP function converts logical coordinates into device coordinates. The conversion depends on the mapping mode of the device context, the settings of the origins and extents for the window and viewport, and the world transformation.

[MAKEPOINTS](#)

The MAKEPOINTS macro converts a value that contains the x- and y-coordinates of a point into a POINTS structure.

[MAKEROP4](#)

The MAKEROP4 macro creates a quaternary raster operation code for use with the MaskBlt function.

[MapWindowPoints](#)

The MapWindowPoints function converts (maps) a set of points from a coordinate space relative to one window to a coordinate space relative to another window.

[MaskBlt](#)

The MaskBlt function combines the color data for the source and destination bitmaps using the specified mask and raster operation.

[MergeFontPackage](#)

The MergeFontPackage function manipulates fonts created by CreateFontPackage.

[MFENUMPROC](#)

The `EnumMetaFileProc` function is an application-defined callback function that processes Windows-format metafile records.

[ModifyWorldTransform](#)

The `ModifyWorldTransform` function changes the world transformation for a device context using the specified mode.

[MONITORENUMPROC](#)

A `MonitorEnumProc` function is an application-defined callback function that is called by the `EnumDisplayMonitors` function.

[MonitorFromPoint](#)

The `MonitorFromPoint` function retrieves a handle to the display monitor that contains a specified point.

[MonitorFromRect](#)

The `MonitorFromRect` function retrieves a handle to the display monitor that has the largest area of intersection with a specified rectangle.

[MonitorFromWindow](#)

The `MonitorFromWindow` function retrieves a handle to the display monitor that has the largest area of intersection with the bounding rectangle of a specified window.

[MoveToEx](#)

The `MoveToEx` function updates the current position to the specified point and optionally returns the previous position.

[OffsetClipRgn](#)

The `OffsetClipRgn` function moves the clipping region of a device context by the specified offsets.

[OffsetRect](#)

The `OffsetRect` function moves the specified rectangle by the specified offsets.

[OffsetRgn](#)

The `OffsetRgn` function moves a region by the specified offsets.

[OffsetViewportOrgEx](#)

The `OffsetViewportOrgEx` function modifies the viewport origin for a device context using the specified horizontal and vertical offsets.

[OffsetWindowOrgEx](#)

The `OffsetWindowOrgEx` function modifies the window origin for a device context using the specified horizontal and vertical offsets.

[PaintDesktop](#)

The `PaintDesktop` function fills the clipping region in the specified device context with the desktop pattern or wallpaper. The function is provided primarily for shell desktops.

[PaintRgn](#)

The PaintRgn function paints the specified region by using the brush currently selected into the device context.

[PALETTEINDEX](#)

The PALETTEINDEX macro accepts an index to a logical-color palette entry and returns a palette-entry specifier consisting of a COLORREF value that specifies the color associated with the given index.

[PALETTERGB](#)

The PALETTERGB macro accepts three values that represent the relative intensities of red, green, and blue and returns a palette-relative red, green, blue (RGB) specifier consisting of 2 in the high-order byte and an RGB value in the three low-order bytes. An application using a color palette can pass this specifier, instead of an explicit RGB value, to functions that expect a color.

[PatBlt](#)

The PatBlt function paints the specified rectangle using the brush that is currently selected into the specified device context. The brush color and the surface color or colors are combined by using the specified raster operation.

[PathToRegion](#)

The PathToRegion function creates a region from the path that is selected into the specified device context. The resulting region uses device coordinates.

[Pie](#)

The Pie function draws a pie-shaped wedge bounded by the intersection of an ellipse and two radials. The pie is outlined by using the current pen and filled by using the current brush.

[PlayEnhMetaFile](#)

The PlayEnhMetaFile function displays the picture stored in the specified enhanced-format metafile.

[PlayEnhMetaFileRecord](#)

The PlayEnhMetaFileRecord function plays an enhanced-metafile record by executing the graphics device interface (GDI) functions identified by the record.

[PlayMetaFile](#)

The PlayMetaFile function displays the picture stored in the given Windows-format metafile on the specified device.

[PlayMetaFileRecord](#)

The PlayMetaFileRecord function plays a Windows-format metafile record by executing the graphics device interface (GDI) function contained within that record.

[PlgBlt](#)

The PlgBlt function performs a bit-block transfer of the bits of color data from the specified rectangle in the source device context to the specified parallelogram in the destination device context.

[POINTSTOPOINT](#)

The POINTSTOPOINT macro copies the contents of a POINTS structure into a POINT structure.

[POINTTOPONTS](#)

The POINTTOPONTS macro converts a POINT structure to a POINTS structure.

[PolyBezier](#)

The PolyBezier function draws one or more Bézier curves.

[PolyBezierTo](#)

The PolyBezierTo function draws one or more Bézier curves.

[PolyDraw](#)

The PolyDraw function draws a set of line segments and Bézier curves.

[Polygon](#)

The Polygon function draws a polygon consisting of two or more vertices connected by straight lines. The polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode.

[Polyline](#)

The Polyline function draws a series of line segments by connecting the points in the specified array.

[PolylineTo](#)

The PolylineTo function draws one or more straight lines.

[PolyPolygon](#)

The PolyPolygon function draws a series of closed polygons. Each polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode. The polygons drawn by this function can overlap.

[PolyPolyline](#)

The PolyPolyline function draws multiple series of connected line segments.

[PolyTextOutA](#)

The PolyTextOut function draws several strings using the font and text colors currently selected in the specified device context.

[PolyTextOutW](#)

The PolyTextOut function draws several strings using the font and text colors currently selected in the specified device context.

[PtInRect](#)

The PtInRect function determines whether the specified point lies within the specified rectangle.

PtInRegion

The PtInRegion function determines whether the specified point is inside the specified region.

PtVisible

The PtVisible function determines whether the specified point is within the clipping region of a device context.

RealizePalette

The RealizePalette function maps palette entries from the current logical palette to the system palette.

Rectangle

The Rectangle function draws a rectangle. The rectangle is outlined by using the current pen and filled by using the current brush.

RectInRegion

The RectInRegion function determines whether any part of the specified rectangle is within the boundaries of a region.

RectVisible

The RectVisible function determines whether any part of the specified rectangle lies within the clipping region of a device context.

RedrawWindow

The RedrawWindow function updates the specified rectangle or region in a window's client area.

ReleaseDC

The ReleaseDC function releases a device context (DC), freeing it for use by other applications. The effect of the ReleaseDC function depends on the type of DC. It frees only common and window DCs. It has no effect on class or private DCs.

RemoveFontMemResourceEx

The RemoveFontMemResourceEx function removes the fonts added from a memory image file.

RemoveFontResourceA

The RemoveFontResource function removes the fonts in the specified file from the system font table.

RemoveFontResourceExA

The RemoveFontResourceEx function removes the fonts in the specified file from the system font table.

RemoveFontResourceExW

The RemoveFontResourceEx function removes the fonts in the specified file from the system font table.

RemoveFontResourceW

The RemoveFontResource function removes the fonts in the specified file from the system font table.

[ResetDCA](#)

The ResetDC function updates the specified printer or plotter device context (DC) using the specified information.

[ResetDCW](#)

The ResetDC function updates the specified printer or plotter device context (DC) using the specified information.

[ResizePalette](#)

The ResizePalette function increases or decreases the size of a logical palette based on the specified value.

[RestoreDC](#)

The RestoreDC function restores a device context (DC) to the specified state. The DC is restored by popping state information off a stack created by earlier calls to the SaveDC function.

[RGB](#)

The RGB macro selects a red, green, blue (RGB) color based on the arguments supplied and the color capabilities of the output device.

[RoundRect](#)

The RoundRect function draws a rectangle with rounded corners. The rectangle is outlined by using the current pen and filled by using the current brush.

[SaveDC](#)

The SaveDC function saves the current state of the specified device context (DC) by copying data describing selected objects and graphic modes (such as the bitmap, brush, palette, font, pen, region, drawing mode, and mapping mode) to a context stack.

[ScaleViewportExtEx](#)

The ScaleViewportExtEx function modifies the viewport for a device context using the ratios formed by the specified multiplicands and divisors.

[ScaleWindowExtEx](#)

The ScaleWindowExtEx function modifies the window for a device context using the ratios formed by the specified multiplicands and divisors.

[ScreenToClient](#)

The ScreenToClient function converts the screen coordinates of a specified point on the screen to client-area coordinates.

[SelectClipPath](#)

The SelectClipPath function selects the current path as a clipping region for a device context, combining the new region with any existing clipping region using the specified mode.

[SelectClipRgn](#)

The SelectClipRgn function selects a region as the current clipping region for the specified device context.

SelectFont

The SelectFont macro selects a font object into the specified device context (DC). The new font object replaces the previous font object.

SelectObject

The SelectObject function selects an object into the specified device context (DC). The new object replaces the previous object of the same type.

SelectPalette

The SelectPalette function selects the specified logical palette into a device context.

SetArcDirection

The SetArcDirection sets the drawing direction to be used for arc and rectangle functions.

SetBitmapBits

The SetBitmapBits function sets the bits of color data for a bitmap to the specified values.

SetBitmapDimensionEx

The SetBitmapDimensionEx function assigns preferred dimensions to a bitmap. These dimensions can be used by applications; however, they are not used by the system.

SetBkColor

The SetBkColor function sets the current background color to the specified color value, or to the nearest physical color if the device cannot represent the specified color value.

SetBkMode

The SetBkMode function sets the background mix mode of the specified device context. The background mix mode is used with text, hatched brushes, and pen styles that are not solid lines.

SetBoundsRect

The SetBoundsRect function controls the accumulation of bounding rectangle information for the specified device context.

SetBrushOrgEx

The SetBrushOrgEx function sets the brush origin that GDI assigns to the next brush an application selects into the specified device context.

SetColorAdjustment

The SetColorAdjustment function sets the color adjustment values for a device context (DC) using the specified values.

SetDCBrushColor

SetDCBrushColor function sets the current device context (DC) brush color to the specified color value. If the device cannot represent the specified color value, the color is set to the nearest physical color.

[SetDCPenColor](#)

The SetDCPenColor function sets the current device context (DC) pen color to the specified color value. If the device cannot represent the specified color value, the color is set to the nearest physical color.

[SetDIBColorTable](#)

The SetDIBColorTable function sets RGB (red, green, blue) color values in a range of entries in the color table of the DIB that is currently selected into a specified device context.

[SetDIBits](#)

The SetDIBits function sets the pixels in a compatible bitmap (DDB) using the color data found in the specified DIB.

[SetDIBitsToDevice](#)

The SetDIBitsToDevice function sets the pixels in the specified rectangle on the device that is associated with the destination device context using color data from a DIB, JPEG, or PNG image.

[SetEnhMetaFileBits](#)

The SetEnhMetaFileBits function creates a memory-based enhanced-format metafile from the specified data.

[SetGraphicsMode](#)

The SetGraphicsMode function sets the graphics mode for the specified device context.

[SetLayout](#)

The SetLayout function changes the layout of a device context (DC).

[SetMapMode](#)

The SetMapMode function sets the mapping mode of the specified device context. The mapping mode defines the unit of measure used to transform page-space units into device-space units, and also defines the orientation of the device's x and y axes.

[SetMapperFlags](#)

The SetMapperFlags function alters the algorithm the font mapper uses when it maps logical fonts to physical fonts.

[SetMetaFileBitsEx](#)

The SetMetaFileBitsEx function creates a memory-based Windows-format metafile from the supplied data.

[SetMetaRgn](#)

The SetMetaRgn function intersects the current clipping region for the specified device context with the current metaregion and saves the combined region as the new metaregion for the specified device context.

[SetMiterLimit](#)

The SetMiterLimit function sets the limit for the length of miter joins for the specified device context.

[SetPaletteEntries](#)

The SetPaletteEntries function sets RGB (red, green, blue) color values and flags in a range of entries in a logical palette.

[SetPixel](#)

The SetPixel function sets the pixel at the specified coordinates to the specified color.

[SetPixelV](#)

The SetPixelV function sets the pixel at the specified coordinates to the closest approximation of the specified color. The point must be in the clipping region and the visible part of the device surface.

[SetPolyFillMode](#)

The SetPolyFillMode function sets the polygon fill mode for functions that fill polygons.

[SetRect](#)

The SetRect function sets the coordinates of the specified rectangle. This is equivalent to assigning the left, top, right, and bottom arguments to the appropriate members of the RECT structure.

[SetRectEmpty](#)

The SetRectEmpty function creates an empty rectangle in which all coordinates are set to zero.

[SetRectRgn](#)

The SetRectRgn function converts a region into a rectangular region with the specified coordinates.

[SetROP2](#)

The SetROP2 function sets the current foreground mix mode.

[SetStretchBltMode](#)

The SetStretchBltMode function sets the bitmap stretching mode in the specified device context.

[SetSystemPaletteUse](#)

The SetSystemPaletteUse function allows an application to specify whether the system palette contains 2 or 20 static colors.

[SetTextAlign](#)

The SetTextAlign function sets the text-alignment flags for the specified device context.

[SetTextCharacterExtra](#)

The SetTextCharacterExtra function sets the intercharacter spacing. Intercharacter spacing is added to each character, including break characters, when the system writes a line of text.

[SetTextColor](#)

The SetTextColor function sets the text color for the specified device context to the specified color.

[SetTextJustification](#)

The SetTextJustification function specifies the amount of space the system should add to the break characters in a string of text. The space is added when an application calls the TextOut or ExtTextOut functions.

[SetViewportExtEx](#)

Sets the horizontal and vertical extents of the viewport for a device context by using the specified values.

[SetViewportOrgEx](#)

The SetViewportOrgEx function specifies which device point maps to the window origin (0,0).

[SetWindowExtEx](#)

The SetWindowExtEx function sets the horizontal and vertical extents of the window for a device context by using the specified values.

[SetWindowOrgEx](#)

The SetWindowOrgEx function specifies which window point maps to the viewport origin (0,0).

[SetWindowRgn](#)

The SetWindowRgn function sets the window region of a window.

[SetWinMetaFileBits](#)

The SetWinMetaFileBits function converts a metafile from the older Windows format to the new enhanced format and stores the new metafile in memory.

[SetWorldTransform](#)

The SetWorldTransform function sets a two-dimensional linear transformation between world space and page space for the specified device context. This transformation can be used to scale, rotate, shear, or translate graphics output.

[StretchBlt](#)

The StretchBlt function copies a bitmap from a source rectangle into a destination rectangle, stretching or compressing the bitmap to fit the dimensions of the destination rectangle, if necessary.

[StretchDIBits](#)

The StretchDIBits function copies the color data for a rectangle of pixels in a DIB, JPEG, or PNG image to the specified destination rectangle.

[StrokeAndFillPath](#)

The StrokeAndFillPath function closes any open figures in a path, strokes the outline of the path by using the current pen, and fills its interior by using the current brush.

[StrokePath](#)

The StrokePath function renders the specified path by using the current pen.

[SubtractRect](#)

The SubtractRect function determines the coordinates of a rectangle formed by subtracting one rectangle from another.

[TabbedTextOutA](#)

The TabbedTextOut function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.

[TabbedTextOutW](#)

The TabbedTextOut function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.

[TextOutA](#)

The TextOut function writes a character string at the specified location, using the currently selected font, background color, and text color.

[TextOutW](#)

The TextOut function writes a character string at the specified location, using the currently selected font, background color, and text color.

[TransparentBlt](#)

The TransparentBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

[TTCharToUnicode](#)

Converts an array of 8-bit character code values to 16-bit Unicode values.

[TTDeleteEmbeddedFont](#)

Releases memory used by an embedded font, hFontReference.

[TTEmbedFont](#)

Creates a font structure containing the subsetted wide-character (16-bit) font. The current font of the device context (hDC) provides the font information.

[TTEmbedFontEx](#)

Creates a font structure containing the subsetted UCS-4 character (32-bit) font. The current font of the device context (hDC) provides the font information.

[TTEmbedFontFromFileA](#)

Creates a font structure containing the subsetted wide-character (16-bit) font. An external file provides the font information.

[TTEnableEmbeddingForFacename](#)

Adds or removes facenames from the typeface exclusion list.

[TTGetEmbeddedFontInfo](#)

Retrieves information about an embedded font, such as embedding permissions. TTGetEmbeddedFontInfo performs the same task as TTLoadEmbeddedFont but does not allocate internal data structures for the embedded font.

[TTGetEmbeddingType](#)

Obtains the embedding privileges of a font.

[TTGetNewFontName](#)

Obtains the family name for the font loaded through TTLoadEmbeddedFont.

[TTIsEmbeddingEnabled](#)

Determines whether the typeface exclusion list contains a specified font.

[TTIsEmbeddingEnabledForFacename](#)

Determines whether embedding is enabled for a specified font.

[TTLoadEmbeddedFont](#)

Reads an embedded font from the document stream and installs it. Also allows a client to further restrict embedding privileges of the font.

[TTRunValidationTests](#)

Validates part or all glyph data of a wide-character (16-bit) font, in the size range specified.

[TTRunValidationTestsEx](#)

Validates part or all glyph data of a UCS-4 character (32-bit) font, in the size range specified.

[UnionRect](#)

The UnionRect function creates the union of two rectangles. The union is the smallest rectangle that contains both source rectangles.

[UnrealizeObject](#)

The UnrealizeObject function resets the origin of a brush or resets a logical palette.

[UpdateColors](#)

The UpdateColors function updates the client area of the specified device context by remapping the current colors in the client area to the currently realized logical palette.

[UpdateWindow](#)

The UpdateWindow function updates the client area of the specified window by sending a WM_PAINT message to the window if the window's update region is not empty.

[ValidateRect](#)

The ValidateRect function validates the client area within a rectangle by removing the rectangle from the update region of the specified window.

[ValidateRgn](#)

The ValidateRgn function validates the client area within a region by removing the region from the current update region of the specified window.

[WidenPath](#)

The WidenPath function redefines the current path as the area that would be painted if the path were stroked using the pen currently selected into the given device context.

[WindowFromDC](#)

The WindowFromDC function returns a handle to the window associated with the specified display device context (DC). Output functions that use the specified device context draw into this window.

Structures

[ABC](#)

The ABC structure contains the width of a character in a TrueType font.

[ABCFLOAT](#)

The ABCFLOAT structure contains the A, B, and C widths of a font character.

[AXESLISTA](#)

The AXESLIST structure contains information on all the axes of a multiple master font.

[AXESLISTW](#)

The AXESLIST structure contains information on all the axes of a multiple master font.

[AXISINFOA](#)

The AXISINFO structure contains information about an axis of a multiple master font.

[AXISINFOW](#)

The AXISINFO structure contains information about an axis of a multiple master font.

[BITMAP](#)

The BITMAP structure defines the type, width, height, color format, and bit values of a bitmap.

[BITMAPCOREHEADER](#)

The BITMAPCOREHEADER structure contains information about the dimensions and color format of a DIB.

[BITMAPCOREINFO](#)

The BITMAPCOREINFO structure defines the dimensions and color information for a DIB.

[BITMAPFILEHEADER](#)

The BITMAPFILEHEADER structure contains information about the type, size, and layout of a file that contains a DIB.

[BITMAPINFO](#)

The BITMAPINFO structure defines the dimensions and color information for a DIB.

[BITMAPV4HEADER](#)

The BITMAPV4HEADER structure is the bitmap information header file. It is an extended version of the BITMAPINFOHEADER structure. Applications can use the BITMAPV5HEADER structure for added functionality.

[BITMAPV5HEADER](#)

The BITMAPV5HEADER structure is the bitmap information header file. It is an extended version of the BITMAPINFOHEADER structure.

[BLENDFUNCTION](#)

The BLENDFUNCTION structure controls blending by specifying the blending functions for source and destination bitmaps.

[COLORADJUSTMENT](#)

The COLORADJUSTMENT structure defines the color adjustment values used by the StretchBlt and StretchDIBits functions when the stretch mode is HALFTONE. You can set the color adjustment values by calling the SetColorAdjustment function.

[DESIGNVECTOR](#)

The DESIGNVECTOR structure is used by an application to specify values for the axes of a multiple master font.

[DIBSECTION](#)

The DIBSECTION structure contains information about a DIB created by calling the CreateDIBSection function.

[DISPLAY_DEVICEA](#)

The DISPLAY_DEVICE structure receives information about the display device specified by the iDevNum parameter of the EnumDisplayDevices function.

[DISPLAY_DEVICEW](#)

The DISPLAY_DEVICE structure receives information about the display device specified by the iDevNum parameter of the EnumDisplayDevices function.

[DRAWTEXTPARAMS](#)

The DRAWTEXTPARAMS structure contains extended formatting options for the DrawTextEx function.

[EMR](#)

The EMR structure provides the base structure for all enhanced metafile records. An enhanced metafile record contains the parameters for a specific GDI function used to create part of a picture in an enhanced format metafile.

[EMRABORTPATH](#)

Contains data for the AbortPath, BeginPath, EndPath, CloseFigure, FlattenPath, WidenPath, SetMetaRgn, SaveDC, and RealizePalette enhanced metafile records.

[EMRALPHABLEND](#)

The EMRALPHABLEND structure contains members for the AlphaBlend enhanced metafile record.

[EMRANGLEARC](#)

The EMRANGLEARC structure contains members for the AngleArc enhanced metafile record.

[EMRARC](#)

The EMRARC, EMRARCTO, EMRCHORD, and EMRPIE structures contain members for the Arc, ArcTo, Chord, and Pie enhanced metafile records.

[EMRBITBLT](#)

The EMRBITBLT structure contains members for the BitBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

[EMRCOLORCORRECTPALETTE](#)

The EMRCOLORCORRECTPALETTE structure contains members for the ColorCorrectPalette enhanced metafile record.

[EMRCOLORMATCHTOTARGET](#)

The EMRCOLORMATCHTOTARGET structure contains members for the ColorMatchToTarget enhanced metafile record.

[EMRCREATEBRUSHINDIRECT](#)

The EMRCREATEBRUSHINDIRECT structure contains members for the CreateBrushIndirect enhanced metafile record.

[EMRCREATECOLORSPACE](#)

The EMRCREATECOLORSPACE structure contains members for the CreateColorSpace enhanced metafile record.

[EMRCREATECOLORSPACEW](#)

The EMRCREATECOLORSPACEW structure contains members for the CreateColorSpace enhanced metafile record. It differs from EMRCREATECOLORSPACE in that it has a Unicode logical color space and also has an optional array containing raw source profile data.

[EMRCREATEDIBPATTERNBRUSHPT](#)

The EMRCREATEDIBPATTERNBRUSHPT structure contains members for the CreateDIBPatternBrushPt enhanced metafile record. The BITMAPINFO structure is followed by the bitmap bits that form a packed device-independent bitmap (DIB).

[EMRCREATEMONOBRUSH](#)

The EMRCREATEMONOBRUSH structure contains members for the CreatePatternBrush (when passed a monochrome bitmap) or CreateDIBPatternBrush (when passed a monochrome DIB) enhanced metafile records.

[EMRCREATEPALETTE](#)

The EMRCREATEPALETTE structure contains members for the CreatePalette enhanced metafile record.

[EMRCREATEPEN](#)

The EMRCREATEPEN structure contains members for the CreatePen enhanced metafile record.

[EMRELLIPSE](#)

The EMRELLIPSE and EMRRECTANGLE structures contain members for the Ellipse and Rectangle enhanced metafile records.

[EMREOF](#)

The EMREOF structure contains data for the enhanced metafile record that indicates the end of the metafile.

[EMREXCLUDECLIPRECT](#)

The EMREXCLUDECLIPRECT and EMRINTERSECTCLIPRECT structures contain members for the ExcludeClipRect and IntersectClipRect enhanced metafile records.

[EMREXTCREATEFONTINDIRECTW](#)

The EMREXTCREATEFONTINDIRECTW structure contains members for the CreateFontIndirect enhanced metafile record.

[EMREXTCREATEPEN](#)

The EMREXTCREATEPEN structure contains members for the ExtCreatePen enhanced metafile record. If the record contains a BITMAPINFO structure, it is followed by the bitmap bits that form a packed device-independent bitmap (DIB).

[EMREXTFLOODFILL](#)

The EMREXTFLOODFILL structure contains members for the ExtFloodFill enhanced metafile record.

[EMREXTSELECTCLIPRGN](#)

The EMREXTSELECTCLIPRGN structure contains members for the ExtSelectClipRgn enhanced metafile record.

[EMREXTTEXTOUTA](#)

The EMREXTTEXTOUTA and EMREXTTEXTOUTW structures contain members for the ExtTextOut, TextOut, or DrawText enhanced metafile records.

[EMRFILLPATH](#)

The EMRFILLPATH,♦EMRSTROKEANDFILLPATH,♦ and EMRSTROKEPATH structures contain members for the FillPath, StrokeAndFillPath, and StrokePath enhanced metafile records.

[EMRFILLRGN](#)

The EMRFILLRGN structure contains members for the FillRgn enhanced metafile record.

[EMRFORMAT](#)

The EMRFORMAT structure contains information that identifies graphics data in an enhanced metafile. A GDICOMMENT_MULTIFORMATS enhanced metafile public comment contains an array of EMRFORMAT structures.

[EMRFRAMERGN](#)

The EMRFRAMERGN structure contains members for the FrameRgn enhanced metafile record.

[EMRGDICOMMENT](#)

The EMRGDICOMMENT structure contains application-specific data.

[EMRGLSBOUNDEDRECORD](#)

The EMRGLSBOUNDEDRECORD structure contains members for an enhanced metafile record generated by OpenGL functions. It contains data for OpenGL functions with information in pixel units that must be scaled when playing the metafile.

[EMRGLSRECORD](#)

The EMRGLSRECORD structure contains members for an enhanced metafile record generated by OpenGL functions. It contains data for OpenGL functions that scale automatically to the OpenGL viewport.

[EMRGRADIENTFILL](#)

The EMRGRADIENTFILL structure contains members for the GradientFill enhanced metafile record.

[EMRINVERTRGN](#)

The EMRINVERTRGN and EMRPAINTRGN structures contain members for the InvertRgn and PaintRgn enhanced metafile records.

[EMRLINETO](#)

The EMRLINETO and EMRMOVETOEX structures contains members for the LineTo and MoveToEx enhanced metafile records.

[EMRMASKBLT](#)

The EMRMASKBLT structure contains members for the MaskBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

[EMRMODIFYWORLDTRANSFORM](#)

The EMRMODIFYWORLDTRANSFORM structure contains members for the ModifyWorldTransform enhanced metafile record.

[EMROFFSETCLIPRGN](#)

The EMROFFSETCLIPRGN structure contains members for the OffsetClipRgn enhanced metafile record.

[EMRPIXELFORMAT](#)

The EMRPIXELFORMAT structure contains the members for the SetPixelFormat enhanced metafile record. The pixel format information in ENHMETAHEADER refers to this structure.

[EMRPLGBT](#)

The EMRPLGBT structure contains members for the PlgBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

[EMRPOLYDRAW](#)

The EMRPOLYDRAW structure contains members for the PolyDraw enhanced metafile record.

[EMRPOLYDRAW16](#)

The EMRPOLYDRAW16 structure contains members for the PolyDraw enhanced metafile record.

[EMRPOLYLINE](#)

The EMRPOLYLINE, EMRPOLYBEZIER, EMRPOLYGON, EMRPOLYBEZIERTO, and EMRPOLYLINETO structures contain members for the Polyline, PolyBezier, Polygon, PolyBezierTo, and PolylineTo enhanced metafile records.

[EMRPOLYLINE16](#)

The EMRPOLYLINE16, EMRPOLYBEZIER16, EMRPOLYGON16, EMRPOLYBEZIERTO16, and EMRPOLYLINETO16 structures contain members for the Polyline, PolyBezier, Polygon, PolyBezierTo, and PolylineTo enhanced metafile records.

[EMRPOLYPOLYLINE](#)

The EMRPOLYPOLYLINE and EMRPOLYPOLYGON structures contain members for the PolyPolyline and PolyPolygon enhanced metafile records.

[EMRPOLYPOLYLINE16](#)

The EMRPOLYPOLYLINE16 and EMRPOLYPOLYGON16 structures contain members for the PolyPolyline and PolyPolygon enhanced metafile records.

[EMRPOLYTEXTOUTA](#)

The EMRPOLYTEXTOUTA and EMRPOLYTEXTOUTW structures contain members for the PolyTextOut enhanced metafile record.

[EMRRESIZEPALETTE](#)

The EMRRESIZEPALETTE structure contains members for the ResizePalette enhanced metafile record.

[EMRRESTOREDC](#)

The EMRRESTOREDC structure contains members for the RestoreDC enhanced metafile record.

[EMRROUNDRECT](#)

The EMRROUNDRECT structure contains members for the RoundRect enhanced metafile record.

[EMRSCALEVIEWPORTEXTEX](#)

The EMRSCALEVIEWPORTEXTEX and EMRSCALEWINDOWEXTEX structures contain members for the ScaleViewportExtEx and ScaleWindowExtEx enhanced metafile records.

[EMRSELECTCLIPPATH](#)

Contains parameters for the SelectClipPath, SetBkMode, SetMapMode, SetPolyFillMode, SetROP2, SetStretchBltMode, SetTextAlign, SetICMMode , and SetLayout enhanced metafile records.

[EMRSELECTOBJECT](#)

The EMRSELECTOBJECT and EMRDELETEOBJECT structures contain members for the SelectObject and DeleteObject enhanced metafile records.

[EMRSELECTPALETTE](#)

The EMRSELECTPALETTE structure contains members for the SelectPalette enhanced metafile record. Note that the bForceBackground parameter in SelectPalette is always recorded as TRUE, which causes the palette to be realized as a background palette.

[EMRSETARCDIRECTION](#)

The EMRSETARCDIRECTION structure contains members for the SetArcDirection enhanced metafile record.

[EMRSETBKCOLOR](#)

The EMRSETBKCOLOR and EMRSETTEXTCOLOR structures contain members for the SetBkColor and SetTextColor enhanced metafile records.

[EMRSETCOLORADJUSTMENT](#)

The EMRSETCOLORADJUSTMENT structure contains members for the SetColorAdjustment enhanced metafile record.

[EMRSETCOLORSPACE](#)

The EMRSETCOLORSPACE, EMRSELECTCOLORSPACE, and EMRDELETECOLORSPACE structures contain members for the SetColorSpace and DeleteColorSpace enhanced metafile records.

[EMRSETDIBITSTODEVICE](#)

The EMRSETDIBITSTODEVICE structure contains members for the SetDIBitsToDevice enhanced metafile record.

[EMRSETICMPROFILE](#)

The EMRSETICMPROFILE structure contains members for the SetICMProfile enhanced metafile record.

[EMRSETMAPPERFLAGS](#)

The EMRSETMAPPERFLAGS structure contains members for the SetMapperFlags enhanced metafile record.

[EMRSETMITERLIMIT](#)

The EMRSETMITERLIMIT structure contains members for the SetMiterLimit enhanced metafile record.

[EMRSETPALETTEENTRIES](#)

The EMRSETPALETTEENTRIES structure contains members for the SetPaletteEntries enhanced metafile record.

[EMRSETPIXELV](#)

The EMRSETPIXELV structure contains members for the SetPixelV enhanced metafile record. When an enhanced metafile is created, calls to SetPixel are also recorded in this record.

[EMRSETVIEWPORTEXTEX](#)

The EMRSETVIEWPORTEXTEX and EMRSETWINDOWEXTEX structures contain members for the SetViewportExtEx and SetWindowExtEx enhanced metafile records.

[EMRSETVIEWPORTORGEX](#)

The EMRSETVIEWPORTORGEX, EMRSETWINDOWWORGEX, and EMRSETBRUSHORGEX structures contain members for the SetViewportOrgEx, SetWindowOrgEx, and SetBrushOrgEx enhanced metafile records.

[EMRSETWORLDTRANSFORM](#)

The EMRSETWORLDTRANSFORM structure contains members for the SetWorldTransform enhanced metafile record.

[EMRSTRETCHBLT](#)

The EMRSTRETCHBLT structure contains members for the StretchBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

[EMRSTRETCHDIBITS](#)

The EMRSTRETCHDIBITS structure contains members for the StretchDIBits enhanced metafile record.

[EMRTEXT](#)

The EMRTEXT structure contains members for text output.

[EMRTRANSPARENTBLT](#)

The EMRTRANSPARENTBLT structure contains members for the TransparentBlt enhanced metafile record.

[ENHMETAHEADER](#)

The ENHMETAHEADER structure contains enhanced-metafile data such as the dimensions of the picture stored in the enhanced metafile, the count of records in the enhanced metafile, the resolution of the device on which the picture was created, and so on. This structure is always the first record in an enhanced metafile.

[ENHMETARECORD](#)

The ENHMETARECORD structure contains data that describes a graphics device interface (GDI) function used to create part of a picture in an enhanced-format metafile.

[ENUMLOGFONTA](#)

The ENUMLOGFONT structure defines the attributes of a font, the complete name of a font, and the style of a font.

[ENUMLOGFONTEXA](#)

The ENUMLOGFONTEX structure contains information about an enumerated font.

[ENUMLOGFONTEXDVA](#)

The ENUMLOGFONTEXDV structure contains the information used to create a font.

[ENUMLOGFONTEXDVW](#)

The ENUMLOGFONTEXDV structure contains the information used to create a font.

[ENUMLOGFONTEXW](#)

The ENUMLOGFONTEX structure contains information about an enumerated font.

[ENUMLOGFONTW](#)

The ENUMLOGFONT structure defines the attributes of a font, the complete name of a font, and the style of a font.

[ENUMTEXTMETRICA](#)

The ENUMTEXTMETRIC structure contains information about a physical font.

[ENUMTEXTMETRICW](#)

The ENUMTEXTMETRIC structure contains information about a physical font.

[EXTLOGFONTA](#)

The EXTLOGFONT structure defines the attributes of a font.

[EXTLOGFONTW](#)

The EXTLOGFONT structure defines the attributes of a font.

[EXTLOGOPEN](#)

The EXTLOGOPEN structure defines the pen style, width, and brush attributes for an extended pen.

[FIXED](#)

The FIXED structure contains the integral and fractional parts of a fixed-point real number.

[GCP_RESULTS](#)

The GCP_RESULTS structure contains information about characters in a string. This structure receives the results of the GetCharacterPlacement function. For some languages, the first element in the arrays may contain more, language-dependent information.

[GCP_RESULTSW](#)

The GCP_RESULTS structure contains information about characters in a string. This structure receives the results of the GetCharacterPlacement function. For some languages, the first element in the arrays may contain more, language-dependent information.

[GLYPHMETRICS](#)

The GLYPHMETRICS structure contains information about the placement and orientation of a glyph in a character cell.

[GLYPHSET](#)

The GLYPHSET structure contains information about a range of Unicode code points.

[GRADIENT_RECT](#)

The GRADIENT_RECT structure specifies the index of two vertices in the pVertex array in the GradientFill function. These two vertices form the upper-left and lower-right boundaries of a rectangle.

[GRADIENT_TRIANGLE](#)

The GRADIENT_TRIANGLE structure specifies the index of three vertices in the pVertex array in the GradientFill function. These three vertices form one triangle.

[HANDLETABLE](#)

The HANDLETABLE structure is an array of handles, each of which identifies a graphics device interface (GDI) object.

[KERNINGPAIR](#)

The KERNINGPAIR structure defines a kerning pair.

[LOGBRUSH](#)

The LOGBRUSH structure defines the style, color, and pattern of a physical brush. It is used by the CreateBrushIndirect and ExtCreatePen functions.

[LOGBRUSH32](#)

The LOGBRUSH32 structure defines the style, color, and pattern of a physical brush.

[LOGFONTA](#)

The LOGFONT structure defines the attributes of a font.

[LOGFONTW](#)

The LOGFONT structure defines the attributes of a font.

[LOGPALETTE](#)

The LOGPALETTE structure defines a logical palette.

[LOGPEN](#)

The LOGPEN structure defines the style, width, and color of a pen. The CreatePenIndirect function uses the LOGPEN structure.

[MAT2](#)

The MAT2 structure contains the values for a transformation matrix used by the GetGlyphOutline function.

[METAHEADER](#)

The METAHEADER structure contains information about a Windows-format metafile.

METARECORD

The METARECORD structure contains a Windows-format metafile record.

MONITORINFO

The MONITORINFO structure contains information about a display monitor. The GetMonitorInfo function stores information in a MONITORINFO structure or a MONITORINFOEX structure. The MONITORINFO structure is a subset of the MONITORINFOEX structure.

MONITORINFOEXA

The MONITORINFOEX structure contains information about a display monitor. The GetMonitorInfo function stores information into a MONITORINFOEX structure or a MONITORINFO structure. The MONITORINFOEX structure is a superset of the MONITORINFO structure.

MONITORINFOEXW

The MONITORINFOEX structure contains information about a display monitor. The GetMonitorInfo function stores information into a MONITORINFOEX structure or a MONITORINFO structure. The MONITORINFOEX structure is a superset of the MONITORINFO structure.

NEWTEXTMETRICA

The NEWTEXTMETRIC structure contains data that describes a physical font.

NEWTEXTMETRICEXA

The NEWTEXTMETRICEX structure contains information about a physical font.

NEWTEXTMETRICEXW

The NEWTEXTMETRICEX structure contains information about a physical font.

NEWTEXTMETRICW

The NEWTEXTMETRIC structure contains data that describes a physical font.

OUTLINETEXTMETRICA

The OUTLINETEXTMETRIC structure contains metrics describing a TrueType font.

OUTLINETEXTMETRICW

The OUTLINETEXTMETRIC structure contains metrics describing a TrueType font.

PAINTSTRUCT

The PAINTSTRUCT structure contains information for an application. This information can be used to paint the client area of a window owned by that application.

PANOSE

The PANOSE structure describes the PANOSE font-classification values for a TrueType font. These characteristics are then used to associate the font with other fonts of similar appearance but different names.

[POINTFX](#)

The POINTFX structure contains the coordinates of points that describe the outline of a character in a TrueType font.

[POLYTEXTA](#)

The POLYTEXT structure describes how the PolyTextOut function should draw a string of text.

[POLYTEXTW](#)

The POLYTEXT structure describes how the PolyTextOut function should draw a string of text.

[RASTERIZER_STATUS](#)

The RASTERIZER_STATUS structure contains information about whether TrueType is installed. This structure is filled when an application calls the GetRasterizerCaps function.

[RGBQUAD](#)

The RGBQUAD structure describes a color consisting of relative intensities of red, green, and blue.

[RGBTRIPLE](#)

The RGBTRIPLE structure describes a color consisting of relative intensities of red, green, and blue. The bmcColors member of the BITMAPCOREINFO structure consists of an array of RGBTRIPLE structures.

[RGNDATA](#)

The RGNDATA structure contains a header and an array of rectangles that compose a region. The rectangles are sorted top to bottom, left to right. They do not overlap.

[RGNDATAHEADER](#)

The RGNDATAHEADER structure describes the data returned by the GetRegionData function.

[TEXTMETRICA](#)

The TEXTMETRIC structure contains basic information about a physical font. All sizes are specified in logical units; that is, they depend on the current mapping mode of the display context.

[TEXTMETRICW](#)

The TEXTMETRIC structure contains basic information about a physical font. All sizes are specified in logical units; that is, they depend on the current mapping mode of the display context.

[TRIVERTEX](#)

The TRIVERTEX structure contains color information and position information.

[TTEMPBEDINFO](#)

The TTEMPBEDINFO structure contains a list of URLs from which the embedded font object may be legitimately referenced.

[TTLOADINFO](#)

The TTLOADINFO structure contains the URL from which the embedded font object has been obtained.

[TTPOLYCURVE](#)

The TTPOLYCURVE structure contains information about a curve in the outline of a TrueType character.

[TTPOLYGONHEADER](#)

The TTPOLYGONHEADER structure specifies the starting position and type of a contour in a TrueType character outline.

[TTVALIDATIONTESTSPARAMS](#)

The TTVALIDATIONTESTSPARAMS structure contains parameters for testing a Microsoft OpenType font.

[TTVALIDATIONTESTSPARAMSEX](#)

The TTVALIDATIONTESTSPARAMSEX structure contains parameters for testing a Microsoft OpenType font.

[WCRANGE](#)

The WCRANGE structure specifies a range of Unicode characters.

[XFORM](#)

The XFORM structure specifies a world-space to page-space transformation.

fontsub.h header

4/21/2022 • 2 minutes to read • [Edit Online](#)

This header is used by Windows GDI. For more information, see:

- [Windows GDI](#)

fontsub.h contains the following programming interfaces:

Functions

[CreateFontPackage](#)

The CreateFontPackage function creates a subset version of a specified TrueType font, typically in order to pass it to a printer.

[MergeFontPackage](#)

The MergeFontPackage function manipulates fonts created by CreateFontPackage.

Callback functions

[CFP_ALLOCPROC](#)

Client-provided callback function, used by CreateFontPackage and MergeFontPackage to allocate memory.

[CFP_FREEPROC](#)

Client-provided callback function, used by CreateFontPackage and MergeFontPackage to free memory.

[CFP_REALLOCPROC](#)

Client-provided callback function, used by CreateFontPackage and MergeFontPackage to reallocate memory when the size of an allocated buffer needs to change.

CFP_ALLOCPROC callback function (fontsub.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Client-provided callback function, used by [CreateFontPackage](#) and [MergeFontPackage](#) to allocate memory.

Syntax

```
CFP_ALLOCPROC CfpAllocproc;  
  
void * CfpAllocproc(  
    size_t unnamedParam1  
)  
{...}
```

Parameters

unnamedParam1

Number of bytes to allocate.

Return value

Returns a void pointer to the allocated space, or **NULL** if there is insufficient memory available.

Remarks

`malloc` conforms to this type; the application can either use `malloc` or a more specialized function for memory allocation. Whatever function is chosen, there must also be appropriate functions to reallocate and to free this memory.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	fontsub.h

See also

[CFP_FREEPROC](#)

[CFP_REALLOCPROC](#)

[CreateFontPackage](#)

[MergeFontPackage](#)

CFP_FREEPROC callback function (fontsub.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Client-provided callback function, used by [CreateFontPackage](#) and [MergeFontPackage](#) to free memory.

Syntax

```
CFP_FREEPROC CfpFreeproc;  
  
void CfpFreeproc(  
    void *unnamedParam1  
)  
{...}
```

Parameters

unnamedParam1

Previously allocated memory block to be freed.

Return value

Deallocates a memory block (*memblock*) that was previously allocated by a call to a [CFP_ALLOCPROC](#) or [CFP_REALLOCPROC](#) callback function. If *memblock* is **NULL**, the pointer should be ignored and the function should return immediately. The function is not required to correctly handle being passed an invalid pointer (a pointer to a memory block that was not allocated by the appropriate [CFP_ALLOCPROC](#) or [CFP_REALLOCPROC](#) callback function).

Remarks

[free](#) conforms to this type; the application can either use [free](#) or a more specialized function. Whatever function is chosen, there must also be appropriate functions to allocate and to reallocate this memory.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	fontsub.h

See also

[CFP_ALLOCPROC](#)

[CFP_REALLOCPROC](#)

[CreateFontPackage](#)

[MergeFontPackage](#)

CFP_REALLOCPROC callback function (fontsub.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Client-provided callback function, used by [CreateFontPackage](#) and [MergeFontPackage](#) to reallocate memory when the size of an allocated buffer needs to change.

Syntax

```
CFP_REALLOCPROC CfpReallocproc;  
  
void * CfpReallocproc(  
    void *unnamedParam1,  
    size_t unnamedParam2  
)  
{...}
```

Parameters

unnamedParam1

Pointer to previously allocated memory block.

unnamedParam2

New size in bytes.

Return value

Returns a void pointer to the reallocated (and possibly moved) memory block. The return value should be **NULL** if the size is zero and the *memblock* argument is not **NULL**, or if there is not enough available memory to expand the block to the given size. In the first case, the original block should be freed. In the second, the original block should be unchanged.

Remarks

[realloc](#) conforms to this type; the application can either use [realloc](#) or a more specialized function for memory reallocation. Whatever function is chosen, there must also be appropriate functions for initial allocation and to free this memory.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	fontsub.h

See also

[CFP_ALLOCPROC](#)

[CFP_FREEPROC](#)

[CreateFontPackage](#)

[MergeFontPackage](#)

CreateFontPackage function (fontsub.h)

4/21/2022 • 5 minutes to read • [Edit Online](#)

The **CreateFontPackage** function creates a subset version of a specified TrueType font, typically in order to pass it to a printer. In order to allow for the fact that pages later in a document may need characters or glyphs that were not used on the first page, this function can create an initial subset font package, then create "Delta" font packages that can be merged with the original subset font package, effectively extending it.

Syntax

```
unsigned long CreateFontPackage(
    [in] const unsigned char *puchSrcBuffer,
    [in] const unsigned long ulSrcBufferSize,
    [out] unsigned char **ppuchFontPackageBuffer,
    [out] unsigned long *pulFontPackageBufferSize,
    [out] unsigned long *pulBytesWritten,
    [in] const unsigned short usFlag,
    [in] const unsigned short usTTCIndex,
    [in] const unsigned short usSubsetFormat,
    [in] const unsigned short usSubsetLanguage,
    [in] const unsigned short usSubsetPlatform,
    [in] const unsigned short usSubsetEncoding,
    [in] const unsigned short *pusSubsetKeepList,
    [in] const unsigned short usSubsetListCount,
    [in] CFP_ALLOCPROC lpfnAllocate,
    [in] CFP_REALLOCPROC lpfnReAllocate,
    [in] CFP_FREEPROC lpfnFree,
    [in] void *lpvReserved
);
```

Parameters

[in] *puchSrcBuffer*

Points to a buffer containing source TTF or TTC data, describing the font that is to be subsetted.

[in] *ulSrcBufferSize*

Specifies size of **puchSrcBuffer*, in bytes.

[out] *ppuchFontPackageBuffer*

Points to a variable of type *unsigned char**. The **CreateFontPackage** function will allocate a buffer ***puchFontPackageBuffer*, using *lpfnAllocate* and *lpfnReAllocate*. On successful return, the buffer will contain the subset font or font package. The application is responsible for eventually freeing the buffer.

[out] *pulFontPackageBufferSize*

Points to an *unsigned long*, which on successful return will specify the allocated size of buffer ***puchFontPackageBuffer*.

[out] *pulBytesWritten*

Points to an *unsigned long*, which on successful return will specify the number of bytes actually used in buffer ***puchFontPackageBuffer*.

[in] usFlag

Specifies whether this font should be subsetted, compressed, or both; whether it is a TTF or TTC; and whether *pusSubsetKeepList represents character codes or glyph indices. Any combination of the following flags may be specified:

VALUE	MEANING
TTFCFP_FLAGS_SUBSET	If set, requests subsetting.
TTFCFP_FLAGS_COMPRESS	If set, requests compression. The currently shipping version of this function does not do compression. This flag allows for future implementation of this capability, but is currently ignored.
TTFCFP_FLAGS_TTC	If set, specifies that the font in <i>puchSrcBuffer</i> is a TTC; otherwise, it must be a TTF.
TTFCFP_FLAGS_GLYPHLIST	If set, specifies that *pusSubsetKeepList is a list of glyph indices; otherwise, it must be a list of character codes.

[in] usTTCIndex

The zero based TTC Index; only used if TTFCFP_FLAGS_TTC is set in *usFlags*.

[in] usSubsetFormat

The format of the file to create. Select one of these values; they cannot be combined.

VALUE	MEANING
TTFCFP_SUBSET	Create a standalone Subset font that cannot be merged with later.
TTFCFP_SUBSET1	Create a Subset font package that can be merged with later.
TTFCFP_DELTA	Create a Delta font package that can merge with a previous subset font.

[in] usSubsetLanguage

The language in the Name table to retain. If Set to 0, all languages will be retained. Used only for initial subsetting: that is, used only if *usSubsetFormat* is either TTFCFP_SUBSET or TTFCFP_SUBSET1, and the TTFCFP_FLAGS_SUBSET flag is set in *usFlags*.

[in] usSubsetPlatform

In conjunction with *usSubsetEncoding*, specifies which CMAP to use. Used only if *pusSubsetKeepList is a list of characters: that is, used only if TTFCFP_FLAGS_GLYPHLIST is not set in *usFlags*. In that case, by this CMAP subtable is applied to *pusSubsetKeepList* to create a list of glyphs to retain in the output font or font package.

If used, this must take one of the following values; they cannot be combined:

VALUE	MEANING

TTFCFP_UNICODE_PLATFORMID	
TTFCFP_APPLE_PLATFORMID	
TTFCFP_ISO_PLATFORMID	
TTFCFP_MS_PLATFORMID	

[in] usSubsetEncoding

In conjunction with *usSubsetPlatform*, specifies which CMAP to use. Used only if **pusSubsetKeepList* is a list of characters: that is, used only if TTFCFP_FLAGS_GLYPHLIST is not set in *usFlags*.

If used, this must take one of the following values; they cannot be combined:

VALUE	MEANING
TTFCFP_STD_MAC_CHAR_SET	Can be used only if <i>usSubsetPlatform</i> == TTFCFP_APPLE_PLATFORMID.
TTFCFP_SYMBOL_CHAR_SET	Can be used only if <i>usSubsetPlatform</i> == TTFSUB_MS_PLATFORMID.
TTFCFP_UNICODE_CHAR_SET	Can be used only if <i>usSubsetPlatform</i> == TTFSUB_MS_PLATFORMID.
TTFCFP_DONT_CARE	

[in] pusSubsetKeepList

Points to an array of integers which comprise a list of character codes or glyph indices that should be retained in the output font or font package. If this list contains character codes (that is, if if TTFCFP_FLAGS_GLYPHLIST is not set in *usFlags*), this list may be either Unicode or some other type of encoding, depending on the Platform-Encoding CMAP specified by *usSubsetPlatform* and *usSubsetEncoding*.

[in] usSubsetListCount

The number of elements in the list **pusSubsetKeepList*.

[in] lpfnAllocate

The callback function to allocate initial memory for *puchFontPackageBuffer* and for temporary buffers.

[in] lpfnReAllocate

The callback function to reallocate memory for *puchFontPackageBuffer* and for temporary buffers.

[in] lpfnFree

The callback function to free up memory allocated by *lpfnAllocate* and *lpfnReAllocate*.

[in] lpvReserved

Must be set to **NULL**.

Return value

If the function is successful, returns zero.

Otherwise, returns a nonzero value. See [Font-Package Function Error Messages](#) for possible error returns.

Remarks

By specifying a value of `TTFCFP_SUBSET` for *usSubsetFormat*, you can directly create a working font rather than a font package. This does not allow for future merging, but if there is no need for merging, this skips a step in the downstream processing: a font package needs to be converted back to a working font before it can be used.

By specifying a value of `TTFCFP_SUBSET1` for *usSubsetFormat*, you can create a font package that allows later merging. For example, consider the case where an application calls this function at the start of a large print job. Part way through the print job, the application discovers that it needs glyphs that are not in the subset it has built. The application can make another call to [CreateFontPackage](#), this time specifying a value of `TTFCFP_DELTA` for *usSubsetFormat*. The printer can use [MergeFontPackage](#) to merge in these additional glyphs.

A CMAP maps from character encodings to glyphs. If **pusSubsetKeepList* is a list of character values, then the application uses parameters *usSubsetPlatform* and *usSubsetEncoding* to specify what type of CMAP is being used, so that character values can be mapped to glyphs.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	fontsub.h
Library	FontSub.lib
DLL	FontSub.dll

See also

[CFP_ALLOCPROC](#)

[CFP_FREEPROC](#)

[CFP_REALLOCPROC](#)

[MergeFontPackage](#)

MergeFontPackage function (fontsub.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **MergeFontPackage** function manipulates fonts created by [CreateFontPackage](#). It is slightly more flexible than its name might suggest: it can appropriately handle all of the subset fonts and font packages created by [CreateFontPackage](#). It can turn a font package into a working font, and it can merge a Delta font package into an appropriately prepared working font.

Typically, [CreateFontPackage](#) creates subset fonts and font packages to pass to a printer or print server; **MergeFontPackage** runs on that printer or print server.

Syntax

```
unsigned long MergeFontPackage(
    [in] const unsigned char *puchMergeFontBuffer,
    [in] const unsigned long ulMergeFontBufferSize,
    [in] const unsigned char *puchFontPackageBuffer,
    [in] const unsigned long ulFontPackageBufferSize,
    [out] unsigned char **ppuchDestBuffer,
    [out] unsigned long *pulDestBufferSize,
    [out] unsigned long *pulBytesWritten,
    [in] const unsigned short usMode,
    [in] CFP_ALLOCPROC lpfnAllocate,
    [in] CFP_REALLOCPROC lpfnReAllocate,
    [in] CFP_FREEPROC lpfnFree,
    [in] void *lpvReserved
);
```

Parameters

[in] `puchMergeFontBuffer`

A pointer to a buffer containing a font to merge with. This is used only when *usMode* is TTFMFP_DELTA.

[in] `ulMergeFontBufferSize`

Specifies size of `*puchMergeFontBuffer`, in bytes.

[in] `puchFontPackageBuffer`

A pointer to a buffer containing a font package.

[in] `ulFontPackageBufferSize`

Specifies size of `*puchFontPackageBuffer`, in bytes.

[out] `ppuchDestBuffer`

A pointer to a variable of type `unsigned char*`. The **MergeFontPackage** function will allocate a buffer `**ppuchDestBuffer`, using `lpfnAllocate` and `lpfnReAllocate`. On successful return, that buffer will contain the resulting merged or expanded font. The application is responsible for eventually freeing that buffer.

[out] `pulDestBufferSize`

Points to an unsigned long, which on successful return will specify the allocated size of buffer `**ppuchDestBuffer`.

.

[out] pulBytesWritten

Points to an unsigned long, which on successful return will specify the number of bytes actually used in buffer ** *ppuchDestBuffer*.

[in] usMode

Specifies what kind of process to perform. Select one of these values; they cannot be combined.

VALUE	MEANING
TTFMFP_SUBSET	Copies a simple working font; see remarks below. <i>puchMergeFontBuffer</i> will be ignored; <i>puchFontPackageBuffer</i> should contain a working font created by CreateFontPackage with <i>usSubsetFormat</i> set to TTFCFP_SUBSET; this working font will simply be copied to <i>ppuchDestBuffer</i> .
TTFMFP_SUBSET1	Turns a font package into a mergeable working font; see remarks below. <i>puchMergeFontBuffer</i> will be ignored; <i>puchFontPackageBuffer</i> should contain a mergeable working font created by CreateFontPackage with <i>usSubsetFormat</i> set to TTFCFP_SUBSET1. The result in ** <i>ppuchDestBuffer</i> will be a working font that may be merged with later.
TTFMFP_DELTA	Merges a Delta font package into a mergeable working font; see remarks below. * <i>puchFontPackageBuffer</i> should contain a font package created by CreateFontPackage with <i>usSubsetFormat</i> set to TTFCFP_DELTA and <i>puchMergeFontBuffer</i> should contain a font package created by a prior call to MergeFontPackage with <i>usMode</i> set to TTFMFP_SUBSET1 or TTFMFP_DELTA. The resulting merged font in ** <i>ppuchDestBuffer</i> will be a working font that may be merged with later.

[in] lpfnAllocate

The callback function to allocate initial memory for *ppuchDestBuffer* and for temporary buffers.

[in] lpfnReAllocate

The callback function to reallocate memory for *ppuchDestBuffer* and for temporary buffers.

[in] lpfnFree

The callback function to free up memory allocated by *lpfnAllocate* and *lpfnReAllocate*.

[in] lpvReserved

Must be set to **NULL**.

Return value

If the function is successful, returns zero.

Otherwise, returns a nonzero value. See [Font-Package Function Error Messages](#) for possible error returns.

Remarks

This function handles four distinct, related entities, each representing a subset font:

ENTITY	PRODUCED BY	DIRECTLY USABLE AS A FONT
Simple working font	CreateFontPackage with <i>usSubsetFormat</i> set to TFCFP_SUBSET.	Yes
Font package	CreateFontPackage with <i>usSubsetFormat</i> set to TTFCFP_SUBSET1.	No
Delta font package	CreateFontPackage with <i>usSubsetFormat</i> set to TTFCFP_DELTA.	No
Mergeable working font	MergeFontPackage with <i>usMode</i> set to TTFMFP_SUBSET1 or TTFMFP_DELTA.	Yes

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	fontsub.h
Library	FontSub.lib
DLL	FontSub.dll

See also

[CFP_ALLOCPROC](#)

[CFP_FREEPROC](#)

[CFP_REALLOCPROC](#)

[CreateFontPackage](#)

mmsystem.h header

4/21/2022 • 2 minutes to read • [Edit Online](#)

This header is used by multiple technologies. For more information, see:

- [Windows GDI](#)
- [Windows Multimedia](#)

mmsystem.h contains the following programming interfaces:

Functions

DIBINDEX

The DIBINDEX macro takes an index to an entry in a DIB color table and returns a COLORREF value that specifies the color associated with the given index.

DIBINDEX macro (mmsystem.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DIBINDEX** macro takes an index to an entry in a DIB color table and returns a [COLORREF](#) value that specifies the color associated with the given index. An application using a device context with a DIB section selected into it can pass this specifier, instead of an explicit red, green, blue (RGB) value, to GDI functions that expect a color. This allows the function to use the color at the specified color table index.

Syntax

```
void DIBINDEX(  
    n  
);
```

Parameters

n

Specifies an index to the color table entry containing the color to be used for a graphics operation.

Return value

None

Remarks

DIBINDEX indexes colors in a DIB color table in a manner similar to the way [PALETTEINDEX](#) indexes colors in a logical palette.

DIBINDEX also works with 16-bit bitmaps and device contexts (DCs).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	mmsystem.h (include Windows.h)

See also

[COLORREF](#)

[Color Macros](#)

[Colors Overview](#)

PALETTEINDEX

RGB

prnasnot.h header

4/21/2022 • 2 minutes to read • [Edit Online](#)

This header is used by Windows GDI. For more information, see:

- [Windows GDI](#)

prnasnot.h contains the following programming interfaces:

Interfaces

[IPrintAsyncNotifyCallback](#)

Creates and manages a communication channel used by applications and components that are hosted by the print spooler.

[IPrintAsyncNotifyChannel](#)

Represents a communication channel that components that are hosted by the print spooler use to send notifications to applications. If the channel is bidirectional, applications can use the same channel to send responses back to the component.

[IPrintAsyncNotifyDataObject](#)

Encapsulates the data sent in a notification channel.

Functions

[CreatePrintAsyncNotifyChannel](#)

Creates a communication channel between a Print Spooler-hosted printing component, such as a print driver or port monitor, and an application that receives notifications from the component.

[RegisterForPrintAsyncNotifications](#)

Enables an application to register for notifications from Print Spooler-hosted printing components such as printer drivers, print processors, and port monitors.

[UnRegisterForPrintAsyncNotifications](#)

Enables an application that has registered to receive notifications from Print Spooler-hosted printing components to unregister.

Enumerations

[PrintAsyncNotifyConversationStyle](#)

Specifies whether communication is bidirectional or unidirectional between applications and Print Spooler-hosted components such as printer drivers, print processors, and port monitors.

[PrintAsyncNotifyError](#)

Specifies the error code portion of the HRESULT returned after an asynchronous notification failure.

[PrintAsyncNotifyUserFilter](#)

Specifies whether notifications will go only to listening applications that are associated with the same user as the Print Spooler-hosted sender, or go to a broader set of listening applications.

prntvpt.h header

4/21/2022 • 2 minutes to read • [Edit Online](#)

This header is used by multiple technologies. For more information, see:

- [Windows GDI](#)
- [XPS Documents](#)

prntvpt.h contains the following programming interfaces:

Functions

[PTCloseProvider](#)

Closes a print ticket provider handle.

[PTConvertDevModeToPrintTicket](#)

Converts a DEVMODE structure to a print ticket inside an IStream.

[PTConvertPrintTicketToDevMode](#)

Converts a print ticket into a DEVMODE structure.

[PTGetPrintCapabilities](#)

Retrieves the printer's capabilities formatted in compliance with the XML Print Schema.

[PTGetPrintDeviceCapabilities](#)

Retrieves the device printer's capabilities formatted in compliance with the XML Print Schema.

[PTGetPrintDeviceResources](#)

It retrieves the print devices resources for a printer formatted in compliance with the XML Print Schema.

[PTMergeAndValidatePrintTicket](#)

Merges two print tickets and returns a valid, viable print ticket.

[PTOpenProvider](#)

Opens an instance of a print ticket provider.

[PTOpenProviderEx](#)

Opens an instance of a print ticket provider.

[PTQuerySchemaVersionSupport](#)

Retrieves the highest (latest) version of the Print Schema that the specified printer supports.

[PTReleaseMemory](#)

Releases buffers associated with print tickets and print capabilities.

Enumerations

[EDefaultDevmodeType](#)

Enables users to specify which DEVMODE to use as the source of default values when a print ticket does not specify all possible settings.

[EPrintTicketScope](#)

Specifies the scope of a print ticket.

t2embapi.h header

4/21/2022 • 2 minutes to read • [Edit Online](#)

This header is used by Windows GDI. For more information, see:

- [Windows GDI](#)

t2embapi.h contains the following programming interfaces:

Functions

[TTCharToUnicode](#)

Converts an array of 8-bit character code values to 16-bit Unicode values.

[TTDeleteEmbeddedFont](#)

Releases memory used by an embedded font, hFontReference.

[TTEmbedFont](#)

Creates a font structure containing the subsetted wide-character (16-bit) font. The current font of the device context (hDC) provides the font information.

[TTEmbedFontEx](#)

Creates a font structure containing the subsetted UCS-4 character (32-bit) font. The current font of the device context (hDC) provides the font information.

[TTEmbedFontFromFileA](#)

Creates a font structure containing the subsetted wide-character (16-bit) font. An external file provides the font information.

[TTEnableEmbeddingForFacename](#)

Adds or removes facenames from the typeface exclusion list.

[TTGetEmbeddedFontInfo](#)

Retrieves information about an embedded font, such as embedding permissions. TTGetEmbeddedFontInfo performs the same task as TTLoadEmbeddedFont but does not allocate internal data structures for the embedded font.

[TTGetEmbeddingType](#)

Obtains the embedding privileges of a font.

[TTGetNewFontName](#)

Obtains the family name for the font loaded through TTLoadEmbeddedFont.

[TTIsEmbeddingEnabled](#)

Determines whether the typeface exclusion list contains a specified font.

[TTIsEmbeddingEnabledForFacename](#)

Determines whether embedding is enabled for a specified font.

[TTLoadEmbeddedFont](#)

Reads an embedded font from the document stream and installs it. Also allows a client to further restrict embedding privileges of the font.

[TTRunValidationTests](#)

Validates part or all glyph data of a wide-character (16-bit) font, in the size range specified.

[TTRunValidationTestsEx](#)

Validates part or all glyph data of a UCS-4 character (32-bit) font, in the size range specified.

Structures

[TTEMBEDINFO](#)

The TTEMBEDINFO structure contains a list of URLs from which the embedded font object may be legitimately referenced.

[TTLOADINFO](#)

The TTLOADINFO structure contains the URL from which the embedded font object has been obtained.

[TTVALIDATIONTESTSPARAMS](#)

The TTVALIDATIONTESTSPARAMS structure contains parameters for testing a Microsoft OpenType font.

[TTVALIDATIONTESTSPARAMSEX](#)

The TTVALIDATIONTESTSPARAMSEX structure contains parameters for testing a Microsoft OpenType font.

TTCharToUnicode function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Converts an array of 8-bit character code values to 16-bit Unicode values.

Syntax

```
LONG TTCharToUnicode(
    [in]    HDC      hDC,
    [in]    UCHAR   *pucCharCodes,
    [in]    ULONG   ulCharCodeSize,
    [out]   USHORT  *pusShortCodes,
    [in]    ULONG   ulShortCodeSize,
    [in]    ULONG   ulFlags
);
```

Parameters

[in] hDC

A device context handle.

[in] pucCharCodes

A pointer to an array of 8-bit character codes to convert to 16-bit Unicode values. Must be set to a non-null value.

[in] ulCharCodeSize

The size of an 8-bit character code array.

[out] pusShortCodes

A pointer to an array that will be filled by this function with the Unicode equivalents of the 8-bit values in the *pucCharCodes* array. This parameter must be set to a non-null value.

[in] ulShortCodeSize

The size, in wide characters, of the character code array.

[in] ulFlags

This parameter is currently unused.

Return value

If successful, returns E_NONE.

Array **pusShortCodes* is filled with 16-bit Unicode values that correspond to the 8-bit character codes in **pucCharCodes.ulShortCodeSize* contains the size, in wide characters, of **pusShortCodes*.

Otherwise, returns an error code described in [Embedding Function Error Messages](#).

Remarks

This function may be useful to clients when creating a list of symbol characters to be subsetted.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[MultiByteToWideChar](#)

[WideCharToMultiByte](#)

TTDeleteEmbeddedFont function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Releases memory used by an embedded font, *hFontReference*.

By default, **TTDeleteEmbeddedFont** also removes the installed version of the font from the user's system. When an installable font is loaded, this function still must be called to release the memory used by the embedded font structure, but a flag can be specified indicating that the font should remain installed on the system.

Syntax

```
LONG TTDeleteEmbeddedFont(
    [in] HANDLE hFontReference,
    [in] ULONG ulFlags,
    [out] ULONG *pulStatus
);
```

Parameters

[in] *hFontReference*

Handle identifying font, as provided in the [TTLoadEmbeddedFont](#) function.

[in] *ulFlags*

Flag specifying font deletion options. Currently, this flag can be set to zero or the following value:

VALUE	MEANING
TTDELETE_DONTREMOVEFONT	Do not remove the installed font from the system, but release the memory previously occupied by the embedded font structure.

[out] *pulStatus*

Currently undefined.

Return value

If successful, **TTDeleteEmbeddedFont** returns a value of E_NONE.

The memory occupied by the embedded font structure is cleared. By default, the font indicated by *hFontReference* is also permanently removed (uninstalled and deleted) from the system.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

The client is responsible for calling this function to remove fonts when the embedding privileges do not allow a font to be permanently installed on a user's system.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTLoadEmbeddedFont](#)

TTEmbedFont function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Creates a font structure containing the subsetted wide-character (16-bit) font. The current font of the device context (hDC) provides the font information.

This function passes the data to a client-defined callback routine for insertion into the document stream.

Syntax

```
LONG TTEmbedFont(
    [in]          HDC           hdc,
    [in]          ULONG         ulFlags,
    [in]          ULONG         ulCharSet,
    [out]         ULONG         *pulPrivStatus,
    [out]         ULONG         *pulStatus,
    [in]          WRITEEMBEDPROC lpfnWriteToStream,
    [in]          LPVOID        lpvWriteStream,
    [in]          USHORT        *pusCharCodeSet,
    [in]          USHORT        usCharCodeCount,
    [in]          USHORT        usLanguage,
    [in, optional] TTEMBEDINFO  *pTTEmbedInfo
);
```

Parameters

[in] hdc

Device context handle.

[in] ulFlags

Flag specifying the embedding request. This flag can have zero or more of the following values.

VALUE	MEANING
TTEMBED_EMBEDEUDC	Include the associated EUDC font file data with the font structure.
TTEMBED_RAW	Return a font structure containing the full character set, non-compressed. This is the default behavior of the function.
TTEMBED_SUBSET	Return a subsetted font containing only the glyphs indicated by the <i>pusCharCodeSet</i> or <i>pulCharCodeSet</i> parameter. These character codes must be denoted as 16-bit or UCS-4 characters, as appropriate for the parameter.
TTEMBED_TTCOMPRESSED	Return a compressed font structure.

[in] ulCharSet

Flag specifying the character set of the font to be embedded. This flag can have one of the following values.

VALUE	MEANING
CHARSET_UNICODE	Unicode character set, requiring 16-bit character encoding.
CHARSET_SYMBOL	Symbol character set, requiring 16-bit character encoding.

[out] pulPrivStatus

Pointer to flag indicating embedding privileges of the font. This flag can have one of the following values. This function returns the least restrictive license granted.

VALUE	MEANING
EMBED_PREVIEWPRINT	Preview and Print Embedding.
EMBED_EDITABLE	Editable Embedding.
EMBED_INSTALLABLE	Installable Embedding.
EMBED_NOEMBEDDING	Restricted License Embedding.

[out] pulStatus

Pointer to a bitfield containing status information about the embedding request. This field is filled upon completion of this function. No bits are currently defined for this parameter.

lpfnWriteToStream

Pointer to the client-defined callback function, which writes the font structure to the document stream. See [WRITEEMBEDPROC](#).

[in] lpvWriteStream

A token to represent the output stream.

[in] pusCharCodeSet

Pointer to the buffer containing the optional Unicode character codes for subsetting. This field is only used for subsetting a font and is ignored if the *ulFlags* field does not specify TTEMBED_SUBSET.

[in] usCharCodeCount

The number of characters in the list of characters indicated by *pusCharCodeSet*. This field is only used for subsetting a font and is ignored if the *ulFlags* field does not specify TTEMBED_SUBSET.

[in] usLanguage

Specifies which language in the name table to keep when subsetting. Set to 0 to keep all languages. This field is only used for subsetting a font and is ignored if the *ulFlags* field does not specify TTEMBED_SUBSET.

[in, optional] pTTEmbedInfo

Pointer to a [TTEMBEDINFO](#) structure containing the URLs from which the embedded font object may be legitimately referenced. If *pTTEmbedInfo* is **NULL**, no URLs will be added to the embedded font object and no URL checking will occur when the client calls [TTLoadEmbeddedFont](#).

Return value

If the embedding is successful, returns [E_NONE](#).

The font structure is incorporated into the document stream by the client. *pulPrivStatus* is set, indicating the embedding privileges of the font; and *pulStatus* is set to provide results of the embedding operation.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

Clients are responsible for determining and indicating the character set of the font.

For information about embedding UCS-4 characters, see [TTEmbedFontEx](#). For information about embedding font characters from a file, see [TTEmbedFontFromFileA](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTEMBEDINFO](#)

[TTEmbedFontEx](#)

[TTEmbedFontFromFileA](#)

[TTLoadEmbeddedFont](#)

TTEmbedFontEx function (t2embapi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

Creates a font structure containing the subsetted UCS-4 character (32-bit) font. The current font of the device context (hDC) provides the font information.

This function passes the data to a client-defined callback routine for insertion into the document stream.

TTEmbedFontEx is used the same way as [TTEmbedFont](#), but accepts a character code set given in UCS-4 (32 bits).

Syntax

```
LONG TTEmbedFontEx(
    [in]          HDC          hdc,
    [in]          ULONG         ulFlags,
    [in]          ULONG         ulCharSet,
    [out]         ULONG         *pulPrivStatus,
    [out]         ULONG         *pulStatus,
    [in]          WRITEEMBEDPROC lpfnWriteToStream,
    [in]          LPVOID        lpvWriteStream,
    [in]          ULONG         *pulCharCodeSet,
    [in]          USHORT        usCharCodeCount,
    [in]          USHORT        usLanguage,
    [in, optional] TTEMBEDINFO  *pTTEmbedInfo
);
```

Parameters

[in] hdc

Device context handle.

[in] ulFlags

Flag specifying the embedding request. This flag can have zero or more of the following values.

VALUE	MEANING
TTEMBED_EMBEDEUDC	Include the associated EUDC font file data with the font structure.
TTEMBED_RAW	Return a font structure containing the full character set, non-compressed. This is the default behavior of the function.
TTEMBED_SUBSET	Return a subsetted font containing only the glyphs indicated by the <i>pusCharCodeSet</i> or <i>pulCharCodeSet</i> parameter. These character codes must be denoted as 16-bit or UCS-4 characters as appropriate for the parameter.
TTEMBED_TTCOMPRESSED	Return a compressed font structure.

[in] ulCharSet

Flag specifying the character set of the font to be embedded. This flag can have one of the following values.

VALUE	MEANING
CHARSET_UNICODE	Unicode character set, requiring 16-bit character encoding.
CHARSET_SYMBOL	Symbol character set, requiring 16-bit character encoding.

[out] pulPrivStatus

Pointer to flag indicating embedding privileges of the font. This flag can have one of the following values. This function returns the least restrictive license granted.

VALUE	MEANING
EMBED_PREVIEWPRINT	Preview and Print Embedding.
EMBED_EDITABLE	Editable Embedding.
EMBED_INSTALLABLE	Installable Embedding.
EMBED_NOEMBEDDING	Restricted License Embedding.

[out] pulStatus

Pointer to a bitfield containing status information about the embedding request. This field is filled upon completion of this function. No bits are currently defined for this parameter.

lpfnWriteToStream

Pointer to the client-defined callback function which writes the font structure to the document stream. See [WRITEEMBEDPROC](#).

[in] lpvWriteStream

A token to represent the output stream.

[in] pulCharCodeSet

Pointer to the buffer containing the optional UCS-4 character codes for subsetting. This field is only used for subsetting a font and is ignored if the *ulFlags* field does not specify TTEMBED_SUBSET.

[in] usCharCodeCount

The number of characters in the list of characters indicated by *pulCharCodeSet*. This field is only used for subsetting a font and is ignored if the *ulFlags* field does not specify TTEMBED_SUBSET.

[in] usLanguage

Specifies which language in the name table to keep when subsetting. Set to 0 to keep all languages. This field is

only used for subsetting a font and is ignored if the *uFlags* field does not specify TTEMBED_SUBSET.

[in, optional] *pTTEmbInfo*

Pointer to a [TTEMBEDINFO](#) structure containing the URLs from which the embedded font object may be legitimately referenced. If *pTTEmbInfo* is NULL, no URLs will be added to the embedded font object and no URL checking will occur when the client calls [TTLoadEmbeddedFont](#).

Return value

If the embedding is successful, returns E_NONE.

The font structure is incorporated into the document stream by the client.

pulPrivStatus is set, indicating the embedding privileges of the font; and *puStatus* is set to provide results of the embedding operation.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

This function references a client-defined callback routine for embedding the font structure into the document stream.

Clients are responsible for determining and indicating the character set of the font.

For information on embedding Unicode characters, see [TTEmbFont](#); for information on embedding Unicode characters from a file, see [TTEmbFontFromFileA](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTEMBEDINFO](#)

[TTEmbFont](#)

[TTEmbFontFromFileA](#)

[TTLoadEmbeddedFont](#)

TTEmbedFontFromFileA function (t2embapi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

Creates a font structure containing the subsetted wide-character (16-bit) font. An external file provides the font information.

This function passes the data to a client-defined callback routine for insertion into the document stream.

Syntax

```
LONG TTEmbedFontFromFileA(
    [in]          HDC           hdc,
    [in]          LPCSTR        szFontFileName,
    [in]          USHORT         usTTCIndex,
    [in]          ULONG          ulFlags,
    [in]          ULONG          ulCharSet,
    [out]         ULONG          *pulPrivStatus,
    [out]         ULONG          *pulStatus,
    [in]          WRITEEMBEDPROC lpfnWriteToStream,
    [in]          LPVOID         lpvWriteStream,
    [in]          USHORT         *pusCharCodeSet,
    [in]          USHORT         usCharCodeCount,
    [in]          USHORT         usLanguage,
    [in, optional] TTEMBEDINFO  *pTTEmbedInfo
);
```

Parameters

[in] hdc

Device context handle.

[in] szFontFileName

The font file name and path to embed. This is an ANSI string.

[in] usTTCIndex

Zero-based index into the font file (TTC) identifying the physical font to embed. If the file contains a single font (such as a TTF or OTF outline file), this parameter should be set to 0.

[in] ulFlags

Flag specifying the embedding request. This flag can have zero or more of the following values.

VALUE	MEANING
TTEMBED_EMBEDEUDC	Include the associated EUDC font file data with the font structure.
TTEMBED_RAW	Return a font structure containing the full character set, non-compressed. This is the default behavior of the function.

TTEMPBED_SUBSET	Return a subsetted font containing only the glyphs indicated by the <i>pusCharCodeSet</i> or <i>pulCharCodeSet</i> parameter. These character codes must be denoted as 16-bit or UCS-4 characters as appropriate for the parameter.
TTEMPBED_TTCOMPRESSED	Return a compressed font structure.

[in] *ul CharSet*

Flag specifying the character set of the font to be embedded. This flag can have one of the following values.

VALUE	MEANING
CHARSET_UNICODE	Unicode character set, requiring 16-bit character encoding.
CHARSET_SYMBOL	Symbol character set, requiring 16-bit character encoding.

[out] *pulPrivStatus*

Pointer to flag indicating embedding privileges of the font. This flag can have one of the following values. This function returns the least restrictive license granted.

VALUE	MEANING
EMBED_PREVIEWPRINT	Preview and Print Embedding.
EMBED_EDITABLE	Editable Embedding.
EMBED_INSTALLABLE	Installable Embedding.
EMBED_NOEMBEDDING	Restricted License Embedding.

[out] *pulStatus*

Pointer to a bitfield containing status information about the embedding request. This field is filled upon completion of this function. No bits are currently defined for this parameter.

lpfnWriteToStream

Pointer to the client-defined callback function that writes the font structure to the document stream. See [WRITEEMBEDPROC](#).

[in] *lpvWriteStream*

A token to represent the output stream.

[in] *pusCharCodeSet*

Pointer to the buffer containing the optional Unicode character codes for subsetting. This field is only used for

subsetting a font and is ignored if the *ulFlags* field does not specify TTEMPBED_SUBSET.

[in] *usCharCodeCount*

The number of characters in the list of characters indicated by *pusCharCodeSet*. This field is only used for subsetting a font and is ignored if the *ulFlags* field does not specify TTEMPBED_SUBSET.

[in] *usLanguage*

Specifies which language in the name table to keep when subsetting. Set to 0 to keep all languages. This field is only used for subsetting a font and is ignored if the *ulFlags* field does not specify TTEMPBED_SUBSET.

[in, optional] *pTTEmbedInfo*

Pointer to a [TTEMPBEDINFO](#) structure containing the URLs from which the embedded font object may be legitimately referenced. If *pTTEmbedInfo* is NULL, no URLs will be added to the embedded font object and no URL checking will occur when the client calls the [TTLoadEmbeddedFont](#) function.

Return value

If the embedding is successful, returns E_NONE.

The font structure is incorporated into the document stream by the client. *puPrivStatus* is set, indicating the embedding privileges of the font; and *puStatus* is set to provide results of the embedding operation.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

This function references a client-defined callback routine for embedding the font structure into the document stream.

Clients are responsible for determining and indicating the character set of the font.

For information on embedding Unicode characters from a device context, see [TTEmbedFont](#); for information on embedding UCS-4 characters from a device context, see [TTEmbedFontEx](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTEMPBEDINFO](#)

[TTEmbedFont](#)

[TTEmbedFontEx](#)

[TTLoadEmbeddedFont](#)

TTEMPBEDINFO structure (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **TTEMPBEDINFO** structure contains a list of URLs from which the embedded font object may be legitimately referenced.

Syntax

```
typedef struct {
    unsigned short usStructSize;
    unsigned short usRootStrSize;
    unsigned short *pusRootStr;
} TTEMPBEDINFO;
```

Members

usStructSize

Size, in bytes, of this structure. The client should set this value to `sizeof(TTEMPBEDINFO)`.

usRootStrSize

Size, in wide characters, of *pusRootStr* including NULL terminator(s).

pusRootStr

One or more full URLs that the embedded font object may be referenced from. Multiple URLs, separated by NULL terminators, can be specified.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	t2embapi.h

See also

[TTEmbedFont](#)

[TTEmbedFontEx](#)

[TTEmbedFontFromFileA](#)

[TTLoadEmbeddedFont](#)

TTEnableEmbeddingForFacename function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Adds or removes facenames from the typeface exclusion list.

Syntax

```
LONG TTEnableEmbeddingForFacename(
    [in] LPCSTR lpszFacename,
    [in] BOOL    bEnable
);
```

Parameters

[in] *lpszFacename*

Pointer to the facename of the font to be added or removed from the typeface exclusion list.

[in] *bEnable*

Boolean controlling operation on typeface exclusion list. If nonzero, then the facename will be removed from the list; if zero, the facename will be added to the list.

Return value

If successful, returns E_NONE.

The facename indicated by *lpszFacename* will be added or removed from the typeface exclusion list.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

The function **TTEnableEmbeddingForFacename** uses a typeface exclusion list to control whether a specific font can be embedded. This list identifies all fonts that should NOT be embedded and is shared by all authoring clients on a single system.

An authoring client can embed fonts without referencing the typeface exclusion list (that is, without using **TTEnableEmbeddingForFacename**). Embedding fonts in a document results in the following tradeoffs.

- Provides all font information within a document so the appropriate client can render the document.
- Adds size to a document.
- Complicates streaming read and write operations to a document and uses more processing bandwidth.
- Makes a document less readable by other applications.
- Can leave copyright issues unmanaged, if the type exclusion list is not used.

Two additional functions, **TTIsEmbeddingEnabled** and **TTIsEmbeddingEnabledForFacename**, access the typeface exclusion list to provide enabling status.

The typeface exclusion list is stored in the registry key

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\t2embed. The default typeface exclusion list should contain the following named value entries representing the Microsoft Windows core fonts.

Value Name	Data Type	Data Value
Arial	REG_DWORD	0
Arial Bold	REG_DWORD	0
Arial Bold Italic	REG_DWORD	0
Arial Italic	REG_DWORD	0
Courier New	REG_DWORD	0
Courier New Bold	REG_DWORD	0
Courier New Bold Italic	REG_DWORD	0
Courier New Italic	REG_DWORD	0
Times New Roman	REG_DWORD	0
Times New Roman Bold	REG_DWORD	0
Times New Roman Bold Italic	REG_DWORD	0
Times New Roman Italic	REG_DWORD	0

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTIsEmbeddingEnabled](#)

[TTIsEmbeddingEnabledForFacename](#)

TTGetEmbeddedFontInfo function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Retrieves information about an embedded font, such as embedding permissions. **TTGetEmbeddedFontInfo** performs the same task as [TTLoadEmbeddedFont](#) but does not allocate internal data structures for the embedded font.

Syntax

```
LONG TTGetEmbeddedFontInfo(
    [in] ULONG          ulFlags,
    [out] ULONG         *pulPrivStatus,
    [in] ULONG          ulPrivs,
    [out] ULONG          *pulStatus,
    READEMBEDPROC lpfnReadFromStream,
    [in] LPVOID         lpvReadStream,
    [in] TTLOADINFO     *pTTLoadInfo
);
```

Parameters

[in] ulFlags

Flags specifying the request. This flag can have zero or more of the following values.

VALUE	MEANING
TTEMBED_EMBEDEUDC	Include the associated EUDC font file data with the font structure.
TTEMBED_RAW	Return a font structure containing the full character set, non-compressed. This is the default behavior of the function.
TTEMBED_SUBSET	Return a subsetted font containing only the glyphs indicated by the <i>pusCharCodeSet</i> or <i>puCharCodeSet</i> parameter. These character codes must be denoted as 16-bit or UCS-4 characters, as appropriate for the parameter.
TTEMBED_TTCOMPRESSED	Return a compressed font structure.

[out] pulPrivStatus

On completion, indicates embedding privileges of the font. A list of possible values follows:

VALUE	MEANING
EMBED_PREVIEWPRINT	Preview and Print Embedding.

EMBED_EDITABLE	Editable Embedding.
EMBED_INSTALLABLE	Installable Embedding.
EMBED_NOEMBEDDING	Restricted License Embedding.

[in] `ulPrivs`

Flag indicating a further restriction of embedding privileges, imposed by the client. See [TTLoadEmbeddedFont](#) for additional information.

This flag must have one of the following values.

VALUE	MEANING
LICENSE_PREVIEWPRINT	Preview and Print Embedding.
LICENSE_EDITABLE	Editable Embedding.
LICENSE_INSTALLABLE	Installable Embedding.
LICENSE_NOEMBEDDING	Restricted License Embedding.
LICENSE_DEFAULT	Use default embedding level.

[out] `pulStatus`

Pointer to a bitfield containing status information, and is filled upon completion of this function. The status can be zero or the following value:

VALUE	MEANING
TTLOAD_FONT_SUBSETTED	The font loaded is a subset of the original font.

`lpfnReadFromStream`

[callback] Pointer to the client-defined callback function that reads the font structure from the document stream.

[in] `lpvReadStream`

Currently undefined. Reserved for a pointer to the stream (font structure).

[in] `pTTLoadInfo`

Pointer to a [TTLOADINFO](#) structure containing the URL from which the embedded font object has been obtained.

Return value

If successful, returns E_NONE.

The location referenced by **puPrivStatus* identifies embedding privileges of the font. The location referenced by **puStatus* identifies whether a subset of the font is embedded.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTGetEmbeddingType](#)

[TTGetNewFontName](#)

[TTLOADINFO](#)

[TTLoadEmbeddedFont](#)

TTGetEmbeddingType function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Obtains the embedding privileges of a font.

Syntax

```
LONG TTGetEmbeddingType(
    [in] HDC     hDC,
    [in] ULONG *pulEmbedType
);
```

Parameters

[in] hDC

Device context handle.

[in] pulEmbedType

Pointer to flag indicating embedding privileges of the font. This flag can have one of the following values. This function returns the least restrictive license granted.

VALUE	MEANING
EMBED_PREVIEWPRINT	Preview and Print Embedding.
EMBED_EDITABLE	Editable Embedding.
EMBED_INSTALLABLE	Installable Embedding.
EMBED_NOEMBEDDING	Restricted License Embedding.

Return value

If successful, returns E_NONE.

This function reads the embedding privileges stored in the font and transfers the privileges to *pu/PrivStatus*.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

Alternatively, an application can determine embedding privileges by using [TTLoadEmbeddedFont](#) and then checking the value returned in *pu/PrivStatus* for success or failure of the function.

Requirements

Requirements	
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTGetEmbeddedFontInfo](#)

[TTGetNewFontName](#)

[TTLoadEmbeddedFont](#)

TTGetNewFontName function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Obtains the family name for the font loaded through [TTLoadEmbeddedFont](#).

Syntax

```
LONG TTGetNewFontName(
    [in]  HANDLE *phFontReference,
    [out] LPWSTR wzWinFamilyName,
    [in]  LONG   cchMaxWinName,
    [out] LPSTR  szMacFamilyName,
    [in]  LONG   cchMaxMacName
);
```

Parameters

[in] phFontReference

Handle that identifies the embedded font that has been installed. The handle references an internal structure, not an Hfont.

[out] wzWinFamilyName

Buffer to hold the new 16-bit-character Microsoft Windows family name.

[in] cchMaxWinName

Length of the string allocated for the Windows name (*szWinFamilyName*). Must be at least LF_FACESIZE long.

[out] szMacFamilyName

Buffer to hold the new 8-bit-character Macintosh family name.

[in] cchMaxMacName

Length of the string allocated for the Macintosh name (*szMacFamilyName*). Must be at least LF_FACESIZE long.

Return value

If successful, returns E_NONE.

The font family name is a string in *szWinFamilyName* or *szMacFamilyName*.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

Note This function returns the font family name in the appropriate string buffer, either for Windows or the Macintosh. The buffer for the other operating system is not used.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTGetEmbeddedFontInfo](#)

[TTGetEmbeddingType](#)

[TTLoadEmbeddedFont](#)

TTIsEmbeddingEnabled function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Determines whether the typeface exclusion list contains a specified font.

Syntax

```
LONG TTIsEmbeddingEnabled(
    [in]  HDC  hDC,
    [out] BOOL *pbEnabled
);
```

Parameters

[in] *hDC*

Device context handle.

[out] *pbEnabled*

Pointer to a boolean, set upon completion of the function. A nonzero value indicates the font is not in the typeface exclusion list and, therefore, can be embedded without conflict.

Return value

If successful, returns E_NONE.

The parameter *pbEnabled* is filled with a boolean that indicates whether embedding is currently enabled within a device context.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

If the specified font is listed, the client should not embed the font.

For additional information on the exclusion list, see the Remarks section of [TTEnableEmbeddingForFacename](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib

DLL	T2embed.dll
-----	-------------

See also

[TTEnableEmbeddingForFacename](#)

[TTGetEmbeddedFontInfo](#)

[TTGetEmbeddingType](#)

[TTIsEmbeddingEnabledForFacename](#)

TTIsEmbeddingEnabledForFacename function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Determines whether embedding is enabled for a specified font.

Syntax

```
LONG TTIsEmbeddingEnabledForFacename(
    [in]  LPCSTR lpszFacename,
    [out] BOOL   *pbEnabled
);
```

Parameters

[in] lpszFacename

Pointer to the facename of the font, such as Arial Bold.

[out] pbEnabled

Pointer to a boolean, set upon completion of the function. A nonzero value indicates the font is not in the typeface exclusion list, and, therefore, can be embedded without conflict.

Return value

If successful, returns E_NONE.

pbEnabled is filled with a boolean that indicates whether embedding is currently enabled within a device context for the specified font.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

If successful, the client may embed the font.

For additional information on the exclusion list, see the Remarks section of [TTEnableEmbeddingForFacename](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h

Library	T2embed.lib
DLL	T2embed.dll

See also

[TTEnableEmbeddingForFacename](#)

[TTGetEmbeddedFontInfo](#)

[TTGetEmbeddingType](#)

[TTIsEmbeddingEnabled](#)

TTLoadEmbeddedFont function (t2embapi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

Reads an embedded font from the document stream and installs it. Also allows a client to further restrict embedding privileges of the font.

Syntax

```
LONG TTLoadEmbeddedFont(
    [out]      HANDLE      *phFontReference,
    [in]       ULONG       ulFlags,
    [out]      ULONG       *pulPrivStatus,
    [in]       ULONG       ulPrivils,
    [out]      ULONG       *pulStatus,
    [in]       READEMBEDPROC lpfnReadFromStream,
    [in]       LPVOID      lpvReadStream,
    [in, optional] LPWSTR   szWinFamilyName,
    [in, optional] LPSTR    szMacFamilyName,
    [in, optional] TTLOADINFO *pTTLoadInfo
);
```

Parameters

[out] phFontReference

A pointer to a handle that identifies the installed embedded font. This handle references an internal structure, not an Hfont.

[in] ulFlags

A flag specifying loading and installation options. Currently, this flag can be set to zero or the following value:

VALUE	MEANING
TTLOAD_PRIVATE	Load the font so that it is not enumerated to the user. If the font is not installable, it will remain private.

[out] pulPrivStatus

A pointer to flag indicating embedding privileges of the font. This flag is written upon completion of this function and can have one of the following values. This function returns the least restrictive license granted.

VALUE	MEANING
EMBED_PREVIEWPRINT	Preview and Print Embedding. The font may be embedded within documents, but must only be installed temporarily on the remote system. A document containing this type of font can only be opened as read-only. The application must not allow the user to edit the document. The document can only be viewed and/or printed.

EMBED_EDITABLE	Editable Embedding. The font may be embedded within documents, but must only be installed temporarily on the remote system. A document containing this type of font may be opened "read/write," with editing permitted.
EMBED_INSTALLABLE	Installable Embedding. The font may be embedded and permanently installed on the remote system. The user of the remote system acquires the identical rights, obligations, and licenses for that font as the original purchaser of the font, and is subject to the same end-user license agreement, copyright, design patent, and/or trademark as was the original purchaser.
EMBED_NOEMBEDDING	Restricted License Embedding. The font must not be modified, embedded, or exchanged in any manner without first obtaining permission of the legal owner.

[in] ulPrivs

A flag indicating a further restriction of embedding privileges, imposed by the client loading the font. This flag must have one of the following values.

VALUE	MEANING
LICENSE_PREVIEWPRINT	Preview and Print Embedding.
LICENSE_EDITABLE	Editable Embedding.
LICENSE_INSTALLABLE	Installable Embedding.
LICENSE_NOEMBEDDING	Restricted License Embedding.
LICENSE_DEFAULT	Use default embedding level.

[out] pulStatus

A pointer to a bitfield containing status information about the **TTLoadEmbeddedFont** request. This field is filled upon completion of this function and can have zero or more of the following values.

VALUE	MEANING
TTLOAD_FONT_SUBSETTED	The font loaded is a subset of the original font.
TTLOAD_FONT_IN_SYSSTARTUP	The font loaded was labeled as installable and has been added to the registry so it will be available upon startup.

[in] lpfnReadFromStream

A pointer to the client-defined callback function that reads the font structure from the document stream.

[in] *lpvReadStream*

A pointer to the stream (font structure).

[in, optional] *szWinFamilyName*

A pointer to the new 16-bit-character Unicode Microsoft Windows family name of the font. Set to **NULL** to use existing name. When changing the name of a font upon loading, you must supply both this parameter and the *szMacFamilyName* parameter.

[in, optional] *szMacFamilyName*

A pointer to the new 8-bit-character Macintosh family name of the font. Set to **NULL** to use existing name. When changing the name of a font upon loading, you must supply both this parameter and the *szWinFamilyName* parameter.

[in, optional] *pTTLoadInfo*

A pointer to a [TTLOADINFO](#) structure containing the URL from which the embedded font object has been obtained. If this value does not match one of those contained in the [TTEMBEDINFO](#) structure, the font will not load successfully.

Return value

If successful, returns [E_NONE](#).

If font loading is successful, a font indicated by *phFontReference* is created from the font structure with the names referenced in *szWinFamilyName* and *szMacFamilyName*. *pulPrivStatus* is set indicating the embedding privileges of the font; and *pulStatus* may be set indicating status information about the font loading operation.

Otherwise, returns an error code described in [Embedding Function Error Messages](#).

Remarks

To assist a client in determining whether an embedded font is already installed on the system, the font loading function will return an error message indicating a font with the same name exists on the system ([E_FONTNAMEALREADYEXISTS](#)), and if that font has the same checksum as the embedded font ([E_FONTRALREADYEXISTS](#)). The client then has two options:

1. Assume that the installed font is really the same as the embedded font and covers the same subsets.
2. Force the embedded font to be installed with a different name to avoid incompatibilities with the font already on the system.

To change the name of an embedded font prior to installing, the client must supply both the 8-bit-character and 16-bit-character name strings as parameters. The font name will be changed in the name table of the newly installed font. The new name is only available to the client and will not be enumerated to the user.

To use the existing name of the embedded font, the name string parameters need to be set to **NULL**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTDeleteEmbeddedFont](#)

[TTEMBEDINFO](#)

[TTGetEmbeddingType](#)

[TTGetNewFontName](#)

[TTLOADINFO](#)

TTLOADINFO structure (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The TTLOADINFO structure contains the URL from which the embedded font object has been obtained.

Syntax

```
typedef struct {
    unsigned short usStructSize;
    unsigned short usRefStrSize;
    unsigned short *pusRefStr;
} TTLOADINFO;
```

Members

`usStructSize`

Size, in bytes, of this structure. The client should set this value to `sizeof(TTLOADINFO)`.

`usRefStrSize`

Size, in wide characters, of `pusRefStr`, including the terminating null character.

`pusRefStr`

Pointer to the string containing the current URL.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	t2embapi.h

See also

[TTEMBEDINFO](#)

[TTLoadEmbeddedFont](#)

TTRunValidationTests function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Validates part or all glyph data of a wide-character (16-bit) font, in the size range specified.

Windows Vista and later: this function will always return E_API_NOTIMPL, and no processing is performed by this API.

Syntax

```
LONG TTRunValidationTests(
    [in] HDC                 hDC,
    [in] TTVALIDATIONTESTSPARAMS *pTestParam
);
```

Parameters

[in] hDC

Device context handle.

[in] pTestParam

Pointer to a [TTVALIDATIONTESTSPARAMS](#) structure specifying the parameters to test.

Return value

If successful and the glyph data is valid, returns E_NONE.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

This function was supported in Windows XP and earlier, but is no longer supported. In Windows Vista and later, this function will always return E_API_NOTIMPL, and no processing is performed by this API.

Effective font validation can be performed by a tool, such as Font Validator, that is capable of performing thorough validation of all parts of the font file. See the [Font Validator documentation](#) for more information.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	t2embapi.h

Library	T2embed.lib
DLL	T2embed.dll

See also

[TTRunValidationTestsEx](#)

[TTVALIDATIONTESTPARAMS](#)

TTRunValidationTestsEx function (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Validates part or all glyph data of a UCS-4 character (32-bit) font, in the size range specified.

Windows Vista and later: this function will always return E_API_NOTIMPL, and no processing is performed by this API.

Syntax

```
LONG TTRunValidationTestsEx(
    [in] HDC                 hDC,
    [in] TTVALIDATIONTESTSPARAMSEX *pTestParam
);
```

Parameters

[in] hDC

Device context handle.

[in] pTestParam

Pointer to a [TTVALIDATIONTESTSPARAMSEX](#) structure specifying the parameters to test.

Return value

If successful and the glyph data is valid, returns E_NONE.

Otherwise, returns an error code described in [Embedding-Function Error Messages](#).

Remarks

TTRunValidationTestsEx is a UCS-4 version of [TTRunValidationTests](#).

This function was supported in Windows XP and earlier, but is no longer supported. In Windows Vista and later, this function will always return E_API_NOTIMPL, and no processing is performed by this API.

Effective font validation can be performed by a tool, such as Font Validator, that is capable of performing thorough validation of all parts of the font file. Please see

<https://www.microsoft.com/typography/FontValidator.aspx> for information on the Font Validator tool.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	t2embapi.h
Library	T2embed.lib
DLL	T2embed.dll

See also

[TTRunValidationTests](#)

[TTVALIDATIONTESTPARAMSEX](#)

TTVALIDATIONTESTSPARAMS structure (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The TTVALIDATIONTESTSPARAMS structure contains parameters for testing a Microsoft OpenType font.

Syntax

```
typedef struct {
    unsigned long    ulStructSize;
    long             lTestFromSize;
    long             lTestToSize;
    unsigned long    ulCharSet;
    unsigned short   usReserved1;
    unsigned short   usCharCodeCount;
    unsigned short *pusCharCodeSet;
} TTVALIDATIONTESTSPARAMS;
```

Members

`ulStructSize`

Size, in bytes, of this structure. The client should set this value to `sizeof(TTVALIDATIONTESTSPARAMS)`.

`lTestFromSize`

First character point size to test. This value is the smallest font size (lower bound) of the font sizes to test.

`lTestToSize`

Last character point size to test. This value is the largest font size (upper bound) of the font sizes to test.

`ulCharSet`

Flag specifying the character set of the font to validate. This flag can have one of the following values.

VALUE	DESCRIPTION
CHARSET_UNICODE	Unicode character set, requiring 16-bit-character encoding.
CHARSET_SYMBOL	Symbol character set, requiring 16-bit-character encoding.

`usReserved1`

Currently not used.

`usCharCodeCount`

If zero, test over all glyphs.

`pusCharCodeSet`

Pointer to array of Unicode characters.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	t2embapi.h

See also

[TTRunValidationTests](#)

[TTVALIDATIONTESTSPARAMSEX](#)

TTVALIDATIONTESTSPARAMSEX structure (t2embapi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The TTVALIDATIONTESTSPARAMSEX structure contains parameters for testing a Microsoft OpenType font.

Syntax

```
typedef struct {
    unsigned long    ulStructSize;
    long             lTestFromSize;
    long             lTestToSize;
    unsigned long    ulCharSet;
    unsigned short   usReserved1;
    unsigned short   usCharCodeCount;
    unsigned long    *pulCharCodeSet;
} TTVALIDATIONTESTSPARAMSEX;
```

Members

`ulStructSize`

Size, in bytes, of this structure. The client should set this value to `sizeof(TTVALIDATIONTESTSPARAMSEX)`.

`lTestFromSize`

First character point size to test. This value is the smallest font size (lower bound) of the font sizes to test.

`lTestToSize`

Last character point size to test. This value is the largest font size (upper bound) to test.

`ulCharSet`

Flag specifying the character set of the font to validate. This flag can have one of the following values.

VALUE	DESCRIPTION
CHARSET_UNICODE	Unicode character set, requiring 16-bit character encoding.
CHARSET_SYMBOL	Symbol character set, requiring 16-bit character encoding.

`usReserved1`

Currently not used.

`usCharCodeCount`

If zero, test over all glyphs.

`pulCharCodeSet`

Pointer to array of UCS-4 characters.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	t2embapi.h

See also

[TTRunValidationTestsEx](#)

[TTVALIDATIONTESTSPARAMS](#)

tvout.h header

4/21/2022 • 2 minutes to read • [Edit Online](#)

This header is used by multiple technologies. For more information, see:

- [Display Devices Reference](#)
- [Windows GDI](#)

tvout.h contains the following programming interfaces:

Structures

VIDEOPARAMETERS

The video miniport driver receives a pointer to a VIDEOPARAMETERS structure in the InputBuffer member of a VIDEO_REQUEST_PACKET when the IOCTL request is IOCTL_VIDEO_HANDLE_VIDEOPARAMETERS.

windef.h header

4/21/2022 • 2 minutes to read • [Edit Online](#)

This header is used by multiple technologies. For more information, see:

- [Display Devices Reference](#)
- [High DPI](#)
- [Windows Accessibility Features](#)
- [Windows and Messages](#)
- [Windows GDI](#)

windef.h contains the following programming interfaces:

Structures

POINT

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

POINTL

The POINTL structure defines the x- and y-coordinates of a point.

POINTS

The POINTS structure defines the x- and y-coordinates of a point.

RECT

The RECT structure defines a rectangle by the coordinates of its upper-left and lower-right corners.

RECTL

The RECTL structure defines a rectangle by the coordinates of its upper-left and lower-right corners.

SIZE

The SIZE structure defines the width and height of a rectangle.

Enumerations

DPI_AWARENESS

Identifies the dots per inch (dpi) setting for a thread, process, or window.

DPI_HOSTING_BEHAVIOR

Identifies the DPI hosting behavior for a window. This behavior allows windows created in the thread to host child windows with a different DPI_AWARENESS_CONTEXT.

windowsx.h header

4/21/2022 • 14 minutes to read • [Edit Online](#)

This header is used by multiple technologies. For more information, see:

- [Developer Notes](#)
- [Windows and Messages](#)
- [Windows Controls](#)
- [Windows GDI](#)

windowsx.h contains the following programming interfaces:

Functions

[Button_Enable](#)

Enables or disables a button.

[Button_GetCheck](#)

Gets the check state of a radio button or check box. You can use this macro or send the BM_GETCHECK message explicitly.

[Button_GetState](#)

Retrieves the state of a button or check box. You can use this macro or send the BM_GETSTATE message explicitly.

[Button_GetText](#)

Gets the text of a button.

[Button_GetTextLength](#)

Gets the number of characters in the text of a button.

[Button_SetCheck](#)

Sets the check state of a radio button or check box. You can use this macro or send the BM_SETCHECK message explicitly.

[Button_SetState](#)

Sets the highlight state of a button. The highlight state indicates whether the button is highlighted as if the user had pushed it. You can use this macro or send the BM_SETSTATE message explicitly.

[Button_SetStyle](#)

Sets the style of a button. You can use this macro or send the BM_SETSTYLE message explicitly.

[Button_SetText](#)

Sets the text of a button.

[ComboBox_AddlItemData](#)

Adds item data to the list in a combo box at the specified location. You can use this macro or send the CB_ADDSTRING message explicitly.

[ComboBox_AddString](#)

Adds a string to a list in a combo box.

[ComboBox_DeleteString](#)

Deletes the item at the specified location in a list in a combo box. You can use this macro or send the CB_DELETESTRING message explicitly.

[ComboBox_Dir](#)

Adds names to the list displayed by a combo box.

[ComboBox_Enable](#)

Enables or disables a combo box control.

[ComboBox_FindlItemData](#)

Finds the first item in a combo box list that has the specified item data. You can use this macro or send the CB_FINDSTRING message explicitly.

[ComboBox_FindString](#)

Finds the first string in a combo box list that begins with the specified string. You can use this macro or send the CB_FINDSTRING message explicitly.

[ComboBox_FindStringExact](#)

Finds the first string in a combo box list that exactly matches the specified string, except that the search is not case sensitive. You can use this macro or send the CB_FINDSTRINGEXACT message explicitly.

[ComboBox_GetCount](#)

Gets the number of items in the list box of a combo box. You can use this macro or send the CB_GETCOUNT message explicitly.

[ComboBox_GetCurSel](#)

Gets the index of the currently selected item in a combo box. You can use this macro or send the CB_GETCURSEL message explicitly.

[ComboBox_GetDroppedControlRect](#)

Retrieves the screen coordinates of a combo box in its dropped-down state. You can use this macro or send the CB_GETDROPPEDCONTROLRECT message explicitly.

[ComboBox_GetDroppedState](#)

Ascertains whether the drop list in a combo box control is visible. You can use this macro or send the CB_GETDROPPEDSTATE message explicitly.

[ComboBox_GetExtendedUI](#)

Ascertainment whether a combo box is using the default user interface (UI) or the extended UI. You can use this macro or send the CB_GETEXTENDEDUI message explicitly.

[ComboBox_GetItemData](#)

Gets the application-defined value associated with the specified list item in a combo box. You can use this macro or send the CB_GETITEMDATA message explicitly.

[ComboBox_GetItemHeight](#)

Retrieves the height of list items in a combo box. You can use this macro or send the CB_GETITEMHEIGHT message explicitly.

[ComboBox_GetLBText](#)

Gets a string from a list in a combo box. You can use this macro or send the CB_GETLBTEXT message explicitly.

[ComboBox_GetLBTextLen](#)

Gets the length of a string in the list in a combo box. You can use this macro or send the CB_GETLBTEXTLEN message explicitly.

[ComboBox_GetText](#)

Retrieves the text from a combo box control.

[ComboBox_GetTextLength](#)

Gets the number of characters in the text of a combo box.

[ComboBox_InsertItemData](#)

Inserts item data in a list in a combo box at the specified location. You can use this macro or send the CB_INSERTSTRING message explicitly.

[ComboBox_InsertString](#)

Adds a string to a list in a combo box at the specified location. You can use this macro or send the CB_INSERTSTRING message explicitly.

[ComboBox_LimitText](#)

Limits the length of the text the user may type into the edit control of a combo box. You can use this macro or send the CB_LIMITTEXT message explicitly.

[ComboBox_ResetContent](#)

Removes all items from the list box and edit control of a combo box. You can use this macro or send the CB_RESETCONTENT message explicitly.

[ComboBox_SelectItemData](#)

Searches a list in a combo box for an item that has the specified item data. If a matching item is found, the item is selected. You can use this macro or send the CB_SELECTSTRING message explicitly.

[ComboBox_SelectString](#)

Searches a list in a combo box for an item that begins with the characters in a specified string. If a matching item is found, the item is selected. You can use this macro or send the CB_SELECTSTRING message explicitly.

[ComboBox_SetCurSel](#)

Sets the currently selected item in a combo box. You can use this macro or send the CB_SETCURSEL message explicitly.

[ComboBox_SetExtendedUI](#)

Selects either the default user interface (UI) or the extended UI for a combo box that has the CBS_DROPDOWN or CBS_DROPDOWNLIST style. You can use this macro or send the CB_SETEXTENDEDUI message explicitly.

[ComboBox_SetItemData](#)

Sets the application-defined value associated with the specified list item in a combo box. You can use this macro or send the CB_SETITEMDATA message explicitly.

[ComboBox_SetItemHeight](#)

Sets the height of list items or the selection field in a combo box. You can use this macro or send the CB_SETITEMHEIGHT message explicitly.

[ComboBox_SetText](#)

Sets the text of a combo box.

[ComboBox_ShowDropdown](#)

Shows or hides the list in a combo box. You can use this macro or send the CB_SHOWDROPDOWN message explicitly.

[DeleteFont](#)

The DeleteFont macro deletes a font object, freeing all system resources associated with the font object.

[Edit_CanUndo](#)

Determines whether there are any actions in the undo queue of an edit or rich edit control. You can use this macro or send the EM_CANUNDO message explicitly.

[Edit_EmptyUndoBuffer](#)

Resets the undo flag of an edit or rich edit control. The undo flag is set whenever an operation within the edit control can be undone. You can use this macro or send the EM_EMPTYUNDOBUFFER message explicitly.

[Edit_Enable](#)

Enables or disables an edit control.

[Edit_FmtLines](#)

Sets a flag that determines whether text retrieved from a multiline edit control includes soft line-break characters.

[Edit_GetFirstVisibleLine](#)

Gets the index of the uppermost visible line in a multiline edit or rich edit control. You can use this macro or send the EM_GETFIRSTVISIBLELINE message explicitly.

[Edit_GetHandle](#)

Gets a handle to the memory currently allocated for the text of a multiline edit control. You can use this macro or send the EM_GETHANDLE message explicitly.

[Edit_GetLine](#)

Retrieves a line of text from an edit or rich edit control. You can use this macro or send the EM_GETLINE message explicitly.

[Edit_GetLineCount](#)

Gets the number of lines in the text of an edit control. You can use this macro or send the EM_GETLINECOUNT message explicitly.

[Edit_GetModify](#)

Gets the state of an edit or rich edit control's modification flag. The flag indicates whether the contents of the control have been modified. You can use this macro or send the EM_GETMODIFY message explicitly.

[Edit_GetPasswordChar](#)

Gets the password character for an edit or rich edit control. You can use this macro or send the EM_GETPASSWORDCHAR message explicitly.

[Edit_GetRect](#)

Gets the formatting rectangle of an edit control. You can use this macro or send the EM_GETRECT message explicitly.

[Edit_GetSel](#)

Gets the starting and ending character positions of the current selection in an edit or rich edit control. You can use this macro or send the EM_GETSEL message explicitly.

[Edit_GetText](#)

Gets the text of an edit control.

[Edit_GetTextLength](#)

Gets the number of characters in the text of an edit control.

[Edit_GetWordBreakProc](#)

Retrieves the address of an edit or rich edit control's Wordwrap function. You can use this macro or send the EM_GETWORDBREAKPROC message explicitly.

[Edit_LimitText](#)

Limits the length of text that can be entered into an edit control. You can use this macro or send the EM_LIMITTEXT message explicitly.

[Edit_LineFromChar](#)

Gets the index of the line that contains the specified character index in a multiline edit or rich edit control. You can use this macro or send the EM_LINEFROMCHAR message explicitly.

[Edit_LineIndex](#)

Gets the character index of the first character of a specified line in a multiline edit or rich edit control. You can use this macro or send the EM_LINEINDEX message explicitly.

[Edit_LineLength](#)

Retrieves the length, in characters, of a line in an edit or rich edit control. You can use this macro or send the EM_LINELENGTH message explicitly.

[Edit_ReplaceSel](#)

Replaces the selected text in an edit control or a rich edit control with the specified text. You can use this macro or send the EM_REPLACESEL message explicitly.

[Edit_Scroll](#)

Scrolls the text vertically in a multiline edit or rich edit control. You can use this macro or send the EM_SCROLL message explicitly.

[Edit_ScrollCaret](#)

Scrolls the caret into view in an edit or rich edit control. You can use this macro or send the EM_SCROLLCARET message explicitly.

[Edit_SetHandle](#)

Sets the handle of the memory that will be used by a multiline edit control. You can use this macro or send the EM_SETHANDLE message explicitly.

[Edit_SetModify](#)

Sets or clears the modification flag for an edit control. The modification flag indicates whether the text within the edit control has been modified. You can use this macro or send the EM_SETMODIFY message explicitly.

[Edit_SetPasswordChar](#)

Sets or removes the password character for an edit or rich edit control. When a password character is set, that character is displayed in place of the characters typed by the user. You can use this macro or send the EM_SETPASSWORDCHAR message explicitly.

[Edit_SetReadOnly](#)

Sets or removes the read-only style (ES_READONLY) of an edit or rich edit control. You can use this macro or send the EM_SETreadonly message explicitly.

[Edit_SetRect](#)

Sets the formatting rectangle of an edit control. You can use this macro or send the EM_SETRECT message explicitly.

[Edit_SetRectNoPaint](#)

Sets the formatting rectangle of a multiline edit control. This macro is equivalent to Edit_SetRect, except that it does not redraw the edit control window. You can use this macro or send the EM_SETRECTNP message explicitly.

[Edit_SetSel](#)

Selects a range of characters in an edit or rich edit control. You can use this macro or send the EM_SETSEL message explicitly.

[Edit_SetTabStops](#)

Sets the tab stops in a multiline edit or rich edit control. When text is copied to the control, any tab character in the text causes space to be generated up to the next tab stop. You can use this macro or send the EM_SETTABSTOPS message explicitly.

[Edit_SetText](#)

Sets the text of an edit control.

[Edit_SetWordBreakProc](#)

Replaces an edit control's default Wordwrap function with an application-defined Wordwrap function. You can use this macro or send the EM_SETWORDBREAKPROC message explicitly.

[Edit_Undo](#)

Undoes the last operation in the undo queue of an edit or rich edit control. You can use this macro or send the EM_UNDO message explicitly.

[GET_X_LPARAM](#)

Retrieves the signed x-coordinate from the specified LPARAM value.

[GET_Y_LPARAM](#)

Retrieves the signed y-coordinate from the given LPARAM value.

[ListBox_AddItemData](#)

Adds item data to the list box at the specified location. You can use this macro or send the LB_ADDSTRING message explicitly.

[ListBox_AddString](#)

Adds a string to a list box.

[ListBox_DeleteString](#)

Deletes the item at the specified location in a list box. You can use this macro or send the LB_DELETESTRING message explicitly.

[ListBox_Dir](#)

Adds names to the list displayed by a list box.

[ListBox_Enable](#)

Enables or disables a list box control.

[ListBox_FindItemData](#)

Finds the first item in a list box that has the specified item data. You can use this macro or send the LB_FINDSTRING message explicitly.

[ListBox_FindString](#)

Finds the first string in a list box that begins with the specified string. You can use this macro or send the LB_FINDSTRING message explicitly.

[ListBox_FindStringExact](#)

Finds the first list box string that exactly matches the specified string, except that the search is not case sensitive. You can use this macro or send the LB_FINDSTRINGEXACT message explicitly.

[ListBox_GetCaretIndex](#)

Retrieves the index of the list box item that has the focus rectangle in a multiple-selection list box. The item may or may not be selected. You can use this macro or send the LB_GETCARETINDEX message explicitly.

[ListBox_GetCount](#)

Gets the number of items in a list box. You can use this macro or send the LB_GETCOUNT message explicitly.

[ListBox_GetCurSel](#)

Gets the index of the currently selected item in a single-selection list box. You can use this macro or send the LB_GETCURSEL message explicitly.

[ListBox_GetHorizontalExtent](#)

Gets the width that a list box can be scrolled horizontally (the scrollable width) if the list box has a horizontal scroll bar. You can use this macro or send the LB_GETHORIZONTALEXTENT message explicitly.

[ListBox_GetItemData](#)

Gets the application-defined value associated with the specified list box item. You can use this macro or send the LB_GETITEMDATA message explicitly.

[ListBox_GetItemHeight](#)

Retrieves the height of items in a list box.

[ListBox_GetItemRect](#)

Gets the dimensions of the rectangle that bounds a list box item as it is currently displayed in the list box. You can use this macro or send the LB_GETITEMRECT message explicitly.

[ListBox_GetSel](#)

Gets the selection state of an item. You can use this macro or send the LB_GETSEL message explicitly.

[ListBox_GetSelCount](#)

Gets the count of selected items in a multiple-selection list box. You can use this macro or send the LB_GETSELCOUNT message explicitly.

[ListBox_GetSelItems](#)

Gets the indexes of selected items in a multiple-selection list box. You can use this macro or send the LB_GETSELITEMS message explicitly.

[ListBox_GetText](#)

Gets a string from a list box. You can use this macro or send the LB_GETTEXT message explicitly.

[ListBox_GetTextLen](#)

Gets the length of a string in a list box. You can use this macro or send the LB_GETTEXTLEN message explicitly.

[ListBox_GetTopIndex](#)

Gets the index of the first visible item in a list box. You can use this macro or send the LB_GETTOPINDEX message explicitly.

[ListBox_InsertItemData](#)

Inserts item data to a list box at the specified location. You can use this macro or send the LB_INSERTSTRING message explicitly.

[ListBox_InsertString](#)

Adds a string to a list box at the specified location. You can use this macro or send the LB_INSERTSTRING message explicitly.

[ListBox_ResetContent](#)

Removes all items from a list box. You can use this macro or send the LB_RESETCONTENT message explicitly.

[ListBox_SelectItemData](#)

Searches a list box for an item that has the specified item data. If a matching item is found, the item is selected. You can use this macro or send the LB_SELECTSTRING message explicitly.

[ListBox_SelectString](#)

Searches a list box for an item that begins with the characters in a specified string. If a matching item is found, the item is selected. You can use this macro or send the LB_SELECTSTRING message explicitly.

[ListBox_SetItemRange](#)

Selects or deselects one or more consecutive items in a multiple-selection list box. You can use this macro or send the LB_SELITEMRANGE message explicitly.

[ListBox_SetCaretIndex](#)

Sets the focus rectangle to the item at the specified index in a multiple-selection list box. If the item is not visible, it is scrolled into view. You can use this macro or send the LB_SETCARETINDEX message explicitly.

[ListBox_SetColumnWidth](#)

Sets the width of all columns in a multiple-column list box. You can use this macro or send the LB_SETCOLUMNWIDTH message explicitly.

[ListBox_SetCurSel](#)

Sets the currently selected item in a single-selection list box. You can use this macro or send the LB_SETCURSEL message explicitly.

[ListBox_SetHorizontalExtent](#)

Set the width by which a list box can be scrolled horizontally (the scrollable width).

[ListBox_SetItemData](#)

Sets the application-defined value associated with the specified list box item. You can use this macro or send the LB_SETITEMDATA message explicitly.

[ListBox_SetItemHeight](#)

Sets the height of items in a list box.

[ListBox_SetSel](#)

Selects or deselects an item in a multiple-selection list box. You can use this macro or send the LB_SETSEL message explicitly.

[ListBox_SetTabStops](#)

Sets the tab-stop positions in a list box. You can use this macro or send the LB_SETTABSTOPS message explicitly.

[ListBox_SetTopIndex](#)

Ensures that the specified item in a list box is visible. You can use this macro or send the LB_SETTOPINDEX message explicitly.

[ScrollBar_Enable](#)

Enables or disables a scroll bar control.

[ScrollBar_GetPos](#)

Retrieves the position of the scroll box (thumb) in the specified scroll bar.

[ScrollBar_GetRange](#)

Gets the range of a scroll bar.

[ScrollBar_SetPos](#)

Sets the position of the scroll box (thumb) in the specified scroll bar and, if requested, redraws the scroll bar to reflect the new position of the scroll box.

[ScrollBar_SetRange](#)

Sets the range of a scroll bar.

[ScrollBar_Show](#)

Shows or hides a scroll bar control.

SelectFont

The SelectFont macro selects a font object into the specified device context (DC). The new font object replaces the previous font object.

Static_Enable

Enables or disables a static control.

Static_GetIcon

Retrieves a handle to the icon associated with a static control that has the SS_ICON style. You can use this macro or send the STM_GETICON message explicitly.

Static_GetText

Gets the text of a static control.

Static_GetTextLength

Gets the number of characters in the text of a static control.

Static_SetIcon

Sets the icon for a static control. You can use this macro or send the STM_SETICON message explicitly.

Static_SetText

Sets the text of a static control.

DeleteFont macro (windowsx.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DeleteFont** macro deletes a font object, freeing all system resources associated with the font object.

Syntax

```
void DeleteFont(  
    hfont  
)
```

Parameters

`hfont`

A handle to the font object.

Return value

None

Remarks

After the font object is deleted, the specified handle is no longer valid.

The **DeleteFont** macro is equivalent to calling [DeleteObject](#) as follows:

```
DeleteObject((HGDIOBJ)(HFONT)(hfont))
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	windowsx.h

See also

[DeleteObject](#)

[SelectFont](#)

SelectFont macro (windowsx.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SelectFont** macro selects a font object into the specified device context (DC). The new font object replaces the previous font object.

Syntax

```
void SelectFont(  
    hdc,  
    hfont  
)
```

Parameters

hdc

A handle to the DC.

hfont

A handle to the font object to be selected. The font object must have been created using either [CreateFont](#) or [CreateFontIndirect](#).

Return value

None

Remarks

After an application has finished drawing with the new font object, it should always replace a new font object with the original font object.

The **SelectFont** macro is equivalent to calling [SelectObject](#) as follows:

```
((HFONT) SelectObject((hdc), (HGDIOBJ)(HFONT)(hfont)))
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	windowsx.h
--------	------------

See also

[DeleteFont](#)

[SelectObject](#)

wingdi.h header

4/21/2022 • 53 minutes to read • [Edit Online](#)

This header is used by multiple technologies. For more information, see:

- [Data Exchange](#)
- [Direct3D 9 Graphics](#)
- [DirectShow](#)
- [Display Devices Reference](#)
- [Internationalization for Windows Applications](#)
- [OpenGL](#)
- [Tablet PC](#)
- [Windows Color System](#)
- [Windows GDI](#)

wingdi.h contains the following programming interfaces:

Functions

[AbortDoc](#)

The AbortDoc function stops the current print job and erases everything drawn since the last call to the StartDoc function.

[AbortPath](#)

The AbortPath function closes and discards any paths in the specified device context.

[AddFontMemResourceEx](#)

The AddFontMemResourceEx function adds the font resource from a memory image to the system.

[AddFontResourceA](#)

The AddFontResource function adds the font resource from the specified file to the system font table. The font can subsequently be used for text output by any application.

[AddFontResourceExA](#)

The AddFontResourceEx function adds the font resource from the specified file to the system. Fonts added with the AddFontResourceEx function can be marked as private and not enumerable.

[AddFontResourceExW](#)

The AddFontResourceEx function adds the font resource from the specified file to the system. Fonts added with the AddFontResourceEx function can be marked as private and not enumerable.

AddFontResourceW

The AddFontResource function adds the font resource from the specified file to the system font table. The font can subsequently be used for text output by any application.

AlphaBlend

The AlphaBlend function displays bitmaps that have transparent or semitransparent pixels.

AngleArc

The AngleArc function draws a line segment and an arc.

AnimatePalette

The AnimatePalette function replaces entries in the specified logical palette.

Arc

The Arc function draws an elliptical arc.

ArcTo

The ArcTo function draws an elliptical arc.

BeginPath

The BeginPath function opens a path bracket in the specified device context.

BitBlt

The BitBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

CancelDC

The CancelDC function cancels any pending operation on the specified device context (DC).

CheckColorsInGamut

The CheckColorsInGamut function determines whether a specified set of RGB triples lies in the output gamut of a specified device. The RGB triples are interpreted in the input logical color space.

ChoosePixelFormat

The ChoosePixelFormat function attempts to match an appropriate pixel format supported by a device context to a given pixel format specification.

Chord

The Chord function draws a chord (a region bounded by the intersection of an ellipse and a line segment, called a secant). The chord is outlined by using the current pen and filled by using the current brush.

[CloseEnhMetaFile](#)

The CloseEnhMetaFile function closes an enhanced-metafile device context and returns a handle that identifies an enhanced-format metafile.

[CloseFigure](#)

The CloseFigure function closes an open figure in a path.

[CloseMetaFile](#)

The CloseMetaFile function closes a metafile device context and returns a handle that identifies a Windows-format metafile.

[CMYK](#)

The CMYK macro creates a CMYK color value by combining the specified cyan, magenta, yellow, and black values.

[ColorCorrectPalette](#)

The ColorCorrectPalette function corrects the entries of a palette using the WCS 1.0 parameters in the specified device context.

[ColorMatchToTarget](#)

The ColorMatchToTarget function enables you to preview colors as they would appear on the target device.

[CombineRgn](#)

The CombineRgn function combines two regions and stores the result in a third region. The two regions are combined according to the specified mode.

[CombineTransform](#)

The CombineTransform function concatenates two world-space to page-space transformations.

[CopyEnhMetaFileA](#)

The CopyEnhMetaFile function copies the contents of an enhanced-format metafile to a specified file.

[CopyEnhMetaFileW](#)

The CopyEnhMetaFile function copies the contents of an enhanced-format metafile to a specified file.

[CopyMetaFileA](#)

The CopyMetaFile function copies the content of a Windows-format metafile to the specified file.

[CopyMetaFileW](#)

The CopyMetaFile function copies the content of a Windows-format metafile to the specified file.

[CreateBitmap](#)

The CreateBitmap function creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel).

[CreateBitmapIndirect](#)

The `CreateBitmapIndirect` function creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel).

[CreateBrushIndirect](#)

The `CreateBrushIndirect` function creates a logical brush that has the specified style, color, and pattern.

[CreateColorSpaceA](#)

The `CreateColorSpace` function creates a logical color space.

[CreateColorSpaceW](#)

The `CreateColorSpace` function creates a logical color space.

[CreateCompatibleBitmap](#)

The `CreateCompatibleBitmap` function creates a bitmap compatible with the device that is associated with the specified device context.

[CreateCompatibleDC](#)

The `CreateCompatibleDC` function creates a memory device context (DC) compatible with the specified device.

[CreateDCA](#)

The `CreateDC` function creates a device context (DC) for a device using the specified name.

[CreateDCW](#)

The `CreateDC` function creates a device context (DC) for a device using the specified name.

[CreateDIBitmap](#)

The `CreateDIBitmap` function creates a compatible bitmap (DDB) from a DIB and, optionally, sets the bitmap bits.

[CreateDIBPatternBrush](#)

The `CreateDIBPatternBrush` function creates a logical brush that has the pattern specified by the specified device-independent bitmap (DIB).

[CreateDIBPatternBrushPt](#)

The `CreateDIBPatternBrushPt` function creates a logical brush that has the pattern specified by the device-independent bitmap (DIB).

[CreateDIBSection](#)

The `CreateDIBSection` function creates a DIB that applications can write to directly.

[CreateDiscardableBitmap](#)

The `CreateDiscardableBitmap` function creates a discardable bitmap that is compatible with the specified device.

[CreateEllipticRgn](#)

The CreateEllipticRgn function creates an elliptical region.

[CreateEllipticRgnIndirect](#)

The CreateEllipticRgnIndirect function creates an elliptical region.

[CreateEnhMetaFileA](#)

The CreateEnhMetaFile function creates a device context for an enhanced-format metafile. This device context can be used to store a device-independent picture.

[CreateEnhMetaFileW](#)

The CreateEnhMetaFile function creates a device context for an enhanced-format metafile. This device context can be used to store a device-independent picture.

[CreateFontA](#)

The CreateFont function creates a logical font with the specified characteristics. The logical font can subsequently be selected as the font for any device.

[CreateFontIndirectA](#)

The CreateFontIndirect function creates a logical font that has the specified characteristics. The font can subsequently be selected as the current font for any device context.

[CreateFontIndirectExA](#)

The CreateFontIndirectEx function specifies a logical font that has the characteristics in the specified structure. The font can subsequently be selected as the current font for any device context.

[CreateFontIndirectExW](#)

The CreateFontIndirectEx function specifies a logical font that has the characteristics in the specified structure. The font can subsequently be selected as the current font for any device context.

[CreateFontIndirectW](#)

The CreateFontIndirect function creates a logical font that has the specified characteristics. The font can subsequently be selected as the current font for any device context.

[CreateFontW](#)

The CreateFont function creates a logical font with the specified characteristics. The logical font can subsequently be selected as the font for any device.

[CreateHalftonePalette](#)

The CreateHalftonePalette function creates a halftone palette for the specified device context (DC).

[CreateHatchBrush](#)

The CreateHatchBrush function creates a logical brush that has the specified hatch pattern and color.

[CreateIC](#)

The CreateIC function creates an information context for the specified device.

[CreateICW](#)

The CreateIC function creates an information context for the specified device.

[CreateMetaFileA](#)

The CreateMetaFile function creates a device context for a Windows-format metafile.

[CreateMetaFileW](#)

The CreateMetaFile function creates a device context for a Windows-format metafile.

[CreatePalette](#)

The CreatePalette function creates a logical palette.

[CreatePatternBrush](#)

The CreatePatternBrush function creates a logical brush with the specified bitmap pattern. The bitmap can be a DIB section bitmap, which is created by the CreateDIBSection function, or it can be a device-dependent bitmap.

[CreatePen](#)

The CreatePen function creates a logical pen that has the specified style, width, and color. The pen can subsequently be selected into a device context and used to draw lines and curves.

[CreatePenIndirect](#)

The CreatePenIndirect function creates a logical cosmetic pen that has the style, width, and color specified in a structure.

[CreatePolygonRgn](#)

The CreatePolygonRgn function creates a polygonal region.

[CreatePolyPolygonRgn](#)

The CreatePolyPolygonRgn function creates a region consisting of a series of polygons. The polygons can overlap.

[CreateRectRgn](#)

The CreateRectRgn function creates a rectangular region.

[CreateRectRgnIndirect](#)

The CreateRectRgnIndirect function creates a rectangular region.

[CreateRoundRectRgn](#)

The CreateRoundRectRgn function creates a rectangular region with rounded corners.

[CreateScalableFontResourceA](#)

The CreateScalableFontResource function creates a font resource file for a scalable font.

[CreateScalableFontResourceW](#)

The CreateScalableFontResource function creates a font resource file for a scalable font.

[CreateSolidBrush](#)

The CreateSolidBrush function creates a logical brush that has the specified solid color.

[DeleteColorSpace](#)

The DeleteColorSpace function removes and destroys a specified color space.

[DeleteDC](#)

The DeleteDC function deletes the specified device context (DC).

[DeleteEnhMetaFile](#)

The DeleteEnhMetaFile function deletes an enhanced-format metafile or an enhanced-format metafile handle.

[DeleteMetaFile](#)

The DeleteMetaFile function deletes a Windows-format metafile or Windows-format metafile handle.

[DeleteObject](#)

The DeleteObject function deletes a logical pen, brush, font, bitmap, region, or palette, freeing all system resources associated with the object. After the object is deleted, the specified handle is no longer valid.

[DescribePixelFormat](#)

The DescribePixelFormat function obtains information about the pixel format identified by iPixelFormat of the device associated with hdc. The function sets the members of the PIXELFORMATDESCRIPTOR structure pointed to by ppfd with that pixel format data.

[DeviceCapabilitiesA](#)

The DeviceCapabilities function retrieves the capabilities of a printer driver.

[DeviceCapabilitiesW](#)

The DeviceCapabilities function retrieves the capabilities of a printer driver.

[DPtoLP](#)

The DPtoLP function converts device coordinates into logical coordinates. The conversion depends on the mapping mode of the device context, the settings of the origins and extents for the window and viewport, and the world transformation.

[DrawEscape](#)

The DrawEscape function provides drawing capabilities of the specified video display that are not directly available through the graphics device interface (GDI).

[Ellipse](#)

The Ellipse function draws an ellipse. The center of the ellipse is the center of the specified bounding rectangle. The ellipse is outlined by using the current pen and is filled by using the current brush.

[EndDoc](#)

The EndDoc function ends a print job.

[EndPage](#)

The EndPage function notifies the device that the application has finished writing to a page. This function is typically used to direct the device driver to advance to a new page.

[EndPath](#)

The EndPath function closes a path bracket and selects the path defined by the bracket into the specified device context.

[EnumEnhMetaFile](#)

The EnumEnhMetaFile function enumerates the records within an enhanced-format metafile by retrieving each record and passing it to the specified callback function.

[EnumFontFamiliesA](#)

The EnumFontFamilies function enumerates the fonts in a specified font family that are available on a specified device.

[EnumFontFamiliesExA](#)

The EnumFontFamiliesEx function enumerates all uniquely-named fonts in the system that match the font characteristics specified by the LOGFONT structure. EnumFontFamiliesEx enumerates fonts based on typeface name, character set, or both.

[EnumFontFamiliesExW](#)

The EnumFontFamiliesEx function enumerates all uniquely-named fonts in the system that match the font characteristics specified by the LOGFONT structure. EnumFontFamiliesEx enumerates fonts based on typeface name, character set, or both.

[EnumFontFamiliesW](#)

The EnumFontFamilies function enumerates the fonts in a specified font family that are available on a specified device.

[EnumFontsA](#)

The EnumFonts function enumerates the fonts available on a specified device.

[EnumFontsW](#)

The EnumFonts function enumerates the fonts available on a specified device.

[EnumICMProfilesA](#)

The EnumICMProfiles function enumerates the different output color profiles that the system supports for a given device context.

[EnumICMProfilesW](#)

The `EnumICMProfiles` function enumerates the different output color profiles that the system supports for a given device context.

[EnumMetaFile](#)

The `EnumMetaFile` function enumerates the records within a Windows-format metafile by retrieving each record and passing it to the specified callback function.

[EnumObjects](#)

The `EnumObjects` function enumerates the pens or brushes available for the specified device context (DC).

[EqualRgn](#)

The `EqualRgn` function checks the two specified regions to determine whether they are identical. The function considers two regions identical if they are equal in size and shape.

[Escape](#)

Enables an application to access the system-defined device capabilities that are not available through GDI.

[ExcludeClipRect](#)

The `ExcludeClipRect` function creates a new clipping region that consists of the existing clipping region minus the specified rectangle.

[ExtCreatePen](#)

The `ExtCreatePen` function creates a logical cosmetic or geometric pen that has the specified style, width, and brush attributes.

[ExtCreateRegion](#)

The `ExtCreateRegion` function creates a region from the specified region and transformation data.

[ExtEscape](#)

The `ExtEscape` function enables an application to access device capabilities that are not available through GDI.

[ExtFloodFill](#)

The `ExtFloodFill` function fills an area of the display surface with the current brush.

[ExtSelectClipRgn](#)

The `ExtSelectClipRgn` function combines the specified region with the current clipping region using the specified mode.

[ExtTextOutA](#)

The `ExtTextOut` function draws text using the currently selected font, background color, and text color. You can optionally provide dimensions to be used for clipping, opaquing, or both.

[ExtTextOutW](#)

The ExtTextOut function draws text using the currently selected font, background color, and text color. You can optionally provide dimensions to be used for clipping, opaquing, or both.

[FillPath](#)

The FillPath function closes any open figures in the current path and fills the path's interior by using the current brush and polygon-filling mode.

[FillRgn](#)

The FillRgn function fills a region by using the specified brush.

[FlattenPath](#)

The FlattenPath function transforms any curves in the path that is selected into the current device context (DC), turning each curve into a sequence of lines.

[FloodFill](#)

The FloodFill function fills an area of the display surface with the current brush. The area is assumed to be bounded as specified by the color parameter.

[FrameRgn](#)

The FrameRgn function draws a border around the specified region by using the specified brush.

[GdiAlphaBlend](#)

The GdiAlphaBlend function displays bitmaps that have transparent or semitransparent pixels.

[GdiComment](#)

The GdiComment function copies a comment from a buffer into a specified enhanced-format metafile.

[GdiFlush](#)

The GdiFlush function flushes the calling thread's current batch.

[GdiGetBatchLimit](#)

The GdiGetBatchLimit function returns the maximum number of function calls that can be accumulated in the calling thread's current batch. The system flushes the current batch whenever this limit is exceeded.

[GdiGradientFill](#)

The GdiGradientFill function fills rectangle and triangle structures.

[GdiSetBatchLimit](#)

The GdiSetBatchLimit function sets the maximum number of function calls that can be accumulated in the calling thread's current batch. The system flushes the current batch whenever this limit is exceeded.

[GdiTransparentBlt](#)

The GdiTransparentBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

[GetArcDirection](#)

The GetArcDirection function retrieves the current arc direction for the specified device context. Arc and rectangle functions use the arc direction.

[GetAspectRatioFilterEx](#)

The GetAspectRatioFilterEx function retrieves the setting for the current aspect-ratio filter.

[GetBitmapBits](#)

The GetBitmapBits function copies the bitmap bits of a specified device-dependent bitmap into a buffer.

[GetBitmapDimensionEx](#)

The GetBitmapDimensionEx function retrieves the dimensions of a compatible bitmap. The retrieved dimensions must have been set by the SetBitmapDimensionEx function.

[GetBkColor](#)

The GetBkColor function returns the current background color for the specified device context.

[GetBkMode](#)

The GetBkMode function returns the current background mix mode for a specified device context. The background mix mode of a device context affects text, hatched brushes, and pen styles that are not solid lines.

[GetBoundsRect](#)

The GetBoundsRect function obtains the current accumulated bounding rectangle for a specified device context.

[GetBrushOrgEx](#)

The GetBrushOrgEx function retrieves the current brush origin for the specified device context. This function replaces the GetBrushOrg function.

[GetValue](#)

The GetValue macro retrieves an intensity value for the blue component of a red, green, blue (RGB) value.

[GetCharABCWidthsA](#)

The GetCharABCWidths function retrieves the widths, in logical units, of consecutive characters in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.

[GetCharABCWidthsFloatA](#)

The GetCharABCWidthsFloat function retrieves the widths, in logical units, of consecutive characters in a specified range from the current font.

[GetCharABCWidthsFloatW](#)

The GetCharABCWidthsFloat function retrieves the widths, in logical units, of consecutive characters in a specified range from the current font.

[GetCharABCWidthsI](#)

The GetCharABCWidthsI function retrieves the widths, in logical units, of consecutive glyph indices in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.

[GetCharABCWidthsW](#)

The GetCharABCWidths function retrieves the widths, in logical units, of consecutive characters in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.

[GetCharacterPlacementA](#)

The GetCharacterPlacement function retrieves information about a character string, such as character widths, caret positioning, ordering within the string, and glyph rendering.

[GetCharacterPlacementW](#)

The GetCharacterPlacement function retrieves information about a character string, such as character widths, caret positioning, ordering within the string, and glyph rendering.

[GetCharWidth32A](#)

The GetCharWidth32 function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

[GetCharWidth32W](#)

The GetCharWidth32 function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

[GetCharWidthA](#)

The GetCharWidth function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

[GetCharWidthFloatA](#)

The GetCharWidthFloat function retrieves the fractional widths of consecutive characters in a specified range from the current font.

[GetCharWidthFloatW](#)

The GetCharWidthFloat function retrieves the fractional widths of consecutive characters in a specified range from the current font.

[GetCharWidthI](#)

The GetCharWidthI function retrieves the widths, in logical coordinates, of consecutive glyph indices in a specified range from the current font.

[GetCharWidthW](#)

The GetCharWidth function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

[GetClipBox](#)

The GetClipBox function retrieves the dimensions of the tightest bounding rectangle that can be drawn around the current visible area on the device.

[GetClipRgn](#)

The GetClipRgn function retrieves a handle identifying the current application-defined clipping region for the specified device context.

[GetColorAdjustment](#)

The GetColorAdjustment function retrieves the color adjustment values for the specified device context (DC).

[GetColorSpace](#)

The GetColorSpace function retrieves the handle to the input color space from a specified device context.

[GetCurrentObject](#)

The GetCurrentObject function retrieves a handle to an object of the specified type that has been selected into the specified device context (DC).

[GetCurrentPositionEx](#)

The GetCurrentPositionEx function retrieves the current position in logical coordinates.

[GetCValue](#)

The GetCValue macro retrieves the cyan color value from a CMYK color value.

[GetDCBrushColor](#)

The GetDCBrushColor function retrieves the current brush color for the specified device context (DC).

[GetDCOrgEx](#)

The GetDCOrgEx function retrieves the final translation origin for a specified device context (DC).

[GetDCPenColor](#)

The GetDCPenColor function retrieves the current pen color for the specified device context (DC).

[GetDeviceCaps](#)

The GetDeviceCaps function retrieves device-specific information for the specified device.

[GetDeviceGammaRamp](#)

The GetDeviceGammaRamp function gets the gamma ramp on direct color display boards having drivers that support downloadable gamma ramps in hardware.

[GetDIBColorTable](#)

The GetDIBColorTable function retrieves RGB (red, green, blue) color values from a range of entries in the color table of the DIB section bitmap that is currently selected into a specified device context.

[GetDIBits](#)

The GetDIBits function retrieves the bits of the specified compatible bitmap and copies them into a buffer as a DIB using the specified format.

[GetEnhMetaFileA](#)

The GetEnhMetaFile function creates a handle that identifies the enhanced-format metafile stored in the specified file.

[GetEnhMetaFileBits](#)

The GetEnhMetaFileBits function retrieves the contents of the specified enhanced-format metafile and copies them into a buffer.

[GetEnhMetaFileDescriptionA](#)

The GetEnhMetaFileDescription function retrieves an optional text description from an enhanced-format metafile and copies the string to the specified buffer.

[GetEnhMetaFileDescriptionW](#)

The GetEnhMetaFileDescription function retrieves an optional text description from an enhanced-format metafile and copies the string to the specified buffer.

[GetEnhMetaFileHeader](#)

The GetEnhMetaFileHeader function retrieves the record containing the header for the specified enhanced-format metafile.

[GetEnhMetaFilePaletteEntries](#)

The GetEnhMetaFilePaletteEntries function retrieves optional palette entries from the specified enhanced metafile.

[GetEnhMetaFilePixelFormat](#)

The GetEnhMetaFilePixelFormat function retrieves pixel format information for an enhanced metafile.

[GetEnhMetaFileW](#)

The GetEnhMetaFile function creates a handle that identifies the enhanced-format metafile stored in the specified file.

[GetFontData](#)

The GetFontData function retrieves font metric data for a TrueType font.

[GetFontLanguageInfo](#)

The GetFontLanguageInfo function returns information about the currently selected font for the specified display context. Applications typically use this information and the GetCharacterPlacement function to prepare a character string for display.

[GetFontUnicodeRanges](#)

The GetFontUnicodeRanges function returns information about which Unicode characters are supported by a font. The information is returned as a GLYPHSET structure.

[GetGlyphIndicesA](#)

The GetGlyphIndices function translates a string into an array of glyph indices. The function can be used to determine whether a glyph exists in a font.

[GetGlyphIndicesW](#)

The GetGlyphIndices function translates a string into an array of glyph indices. The function can be used to determine whether a glyph exists in a font.

[GetGlyphOutlineA](#)

The GetGlyphOutline function retrieves the outline or bitmap for a character in the TrueType font that is selected into the specified device context.

[GetGlyphOutlineW](#)

The GetGlyphOutline function retrieves the outline or bitmap for a character in the TrueType font that is selected into the specified device context.

[GetGraphicsMode](#)

The GetGraphicsMode function retrieves the current graphics mode for the specified device context.

[GetGValue](#)

The GetGValue macro retrieves an intensity value for the green component of a red, green, blue (RGB) value.

[GetICMProfileA](#)

The GetICMProfile function retrieves the file name of the current output color profile for a specified device context.

[GetICMProfileW](#)

The GetICMProfile function retrieves the file name of the current output color profile for a specified device context.

[GetKerningPairsA](#)

The GetKerningPairs function retrieves the character-kerning pairs for the currently selected font for the specified device context.

[GetKerningPairsW](#)

The GetKerningPairs function retrieves the character-kerning pairs for the currently selected font for the specified device context.

[GetKValue](#)

The GetKValue macro retrieves the black color value from a CMYK color value.

[GetLayout](#)

The GetLayout function returns the layout of a device context (DC).

[GetLogColorSpaceA](#)

The GetLogColorSpace function retrieves the color space definition identified by a specified handle.

[GetLogColorSpaceW](#)

The GetLogColorSpace function retrieves the color space definition identified by a specified handle.

[GetMapMode](#)

The GetMapMode function retrieves the current mapping mode.

[GetMetaFileA](#)

The GetMetaFile function creates a handle that identifies the metafile stored in the specified file.

[GetMetaFileBitsEx](#)

The GetMetaFileBitsEx function retrieves the contents of a Windows-format metafile and copies them into the specified buffer.

[GetMetaFileW](#)

The GetMetaFile function creates a handle that identifies the metafile stored in the specified file.

[GetMetaRgn](#)

The GetMetaRgn function retrieves the current metaregion for the specified device context.

[GetMiterLimit](#)

The GetMiterLimit function retrieves the miter limit for the specified device context.

[GetMValue](#)

The GetMValue macro retrieves the magenta color value from a CMYK color value.

[GetNearestColor](#)

The GetNearestColor function retrieves a color value identifying a color from the system palette that will be displayed when the specified color value is used.

[GetNearestPaletteIndex](#)

The GetNearestPaletteIndex function retrieves the index for the entry in the specified logical palette most closely matching a specified color value.

[GetObject](#)

The GetObject function retrieves information for the specified graphics object.

[GetObjectA](#)

The GetObject function retrieves information for the specified graphics object.

[GetObjectType](#)

The GetObjectType retrieves the type of the specified object.

[GetObjectW](#)

The GetObject function retrieves information for the specified graphics object.

[GetOutlineTextMetricsA](#)

The GetOutlineTextMetrics function retrieves text metrics for TrueType fonts.

[GetOutlineTextMetricsW](#)

The GetOutlineTextMetrics function retrieves text metrics for TrueType fonts.

[GetPaletteEntries](#)

The GetPaletteEntries function retrieves a specified range of palette entries from the given logical palette.

[GetPath](#)

The GetPath function retrieves the coordinates defining the endpoints of lines and the control points of curves found in the path that is selected into the specified device context.

[GetPixel](#)

The GetPixel function retrieves the red, green, blue (RGB) color value of the pixel at the specified coordinates.

[GetPixelFormat](#)

The GetPixelFormat function obtains the index of the currently selected pixel format of the specified device context.

[GetPolyFillMode](#)

The GetPolyFillMode function retrieves the current polygon fill mode.

[GetRandomRgn](#)

The GetRandomRgn function copies the system clipping region of a specified device context to a specific region.

[GetRasterizerCaps](#)

The GetRasterizerCaps function returns flags indicating whether TrueType fonts are installed in the system.

[GetRegionData](#)

The GetRegionData function fills the specified buffer with data describing a region. This data includes the dimensions of the rectangles that make up the region.

[GetRgnBox](#)

The GetRgnBox function retrieves the bounding rectangle of the specified region.

[GetROP2](#)

The GetROP2 function retrieves the foreground mix mode of the specified device context. The mix mode specifies how the pen or interior color and the color already on the screen are combined to yield a new color.

[GetValue](#)

The GetValue macro retrieves an intensity value for the red component of a red, green, blue (RGB) value.

[GetStockObject](#)

The GetStockObject function retrieves a handle to one of the stock pens, brushes, fonts, or palettes.

[GetStretchBltMode](#)

The GetStretchBltMode function retrieves the current stretching mode. The stretching mode defines how color data is added to or removed from bitmaps that are stretched or compressed when the StretchBlt function is called.

[GetSystemPaletteEntries](#)

The GetSystemPaletteEntries function retrieves a range of palette entries from the system palette that is associated with the specified device context (DC).

[GetSystemPaletteUse](#)

The GetSystemPaletteUse function retrieves the current state of the system (physical) palette for the specified device context (DC).

[GetTextAlign](#)

The GetTextAlign function retrieves the text-alignment setting for the specified device context.

[GetTextCharacterExtra](#)

The GetTextCharacterExtra function retrieves the current intercharacter spacing for the specified device context.

[GetTextCharset](#)

Retrieves a character set identifier for the font that is currently selected into a specified device context.

[GetTextCharsetInfo](#)

Retrieves information about the character set of the font that is currently selected into a specified device context.

[GetTextColor](#)

The GetTextColor function retrieves the current text color for the specified device context.

[GetTextExtentExPointA](#)

The GetTextExtentExPoint function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.

[GetTextExtentExPointI](#)

The `GetTextExtentExPointI` function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.

[GetTextExtentExPointW](#)

The `GetTextExtentExPoint` function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.

[GetTextExtentPoint32A](#)

The `GetTextExtentPoint32` function computes the width and height of the specified string of text.

[GetTextExtentPoint32W](#)

The `GetTextExtentPoint32` function computes the width and height of the specified string of text.

[GetTextExtentPointA](#)

The `GetTextExtentPoint` function computes the width and height of the specified string of text.

[GetTextExtentPointI](#)

The `GetTextExtentPointI` function computes the width and height of the specified array of glyph indices.

[GetTextExtentPointW](#)

The `GetTextExtentPoint` function computes the width and height of the specified string of text.

[GetTextFaceA](#)

The `GetTextFace` function retrieves the typeface name of the font that is selected into the specified device context.

[GetTextFaceW](#)

The `GetTextFace` function retrieves the typeface name of the font that is selected into the specified device context.

[GetTextMetrics](#)

The `GetTextMetrics` function fills the specified buffer with the metrics for the currently selected font.

[GetTextMetricsA](#)

The `GetTextMetrics` function fills the specified buffer with the metrics for the currently selected font.

[GetTextMetricsW](#)

The `GetTextMetrics` function fills the specified buffer with the metrics for the currently selected font.

[GetViewportExtEx](#)

The `GetViewportExtEx` function retrieves the x-extent and y-extent of the current viewport for the specified device context.

[GetViewportOrgEx](#)

The GetViewportOrgEx function retrieves the x-coordinates and y-coordinates of the viewport origin for the specified device context.

[GetWindowExtEx](#)

This function retrieves the x-extent and y-extent of the window for the specified device context.

[GetWindowOrgEx](#)

The GetWindowOrgEx function retrieves the x-coordinates and y-coordinates of the window origin for the specified device context.

[GetWinMetaFileBits](#)

The GetWinMetaFileBits function converts the enhanced-format records from a metafile into Windows-format records and stores the converted records in the specified buffer.

[GetWorldTransform](#)

The GetWorldTransform function retrieves the current world-space to page-space transformation.

[GetYValue](#)

The GetYValue macro retrieves the yellow color value from a CMYK color value.

[GradientFill](#)

The GradientFill function fills rectangle and triangle structures.

[IntersectClipRect](#)

The IntersectClipRect function creates a new clipping region from the intersection of the current clipping region and the specified rectangle.

[InvertRgn](#)

The InvertRgn function inverts the colors in the specified region.

[LineDDA](#)

The LineDDA function determines which pixels should be highlighted for a line defined by the specified starting and ending points.

[LineTo](#)

The LineTo function draws a line from the current position up to, but not including, the specified point.

[LPtoDP](#)

The LPtoDP function converts logical coordinates into device coordinates. The conversion depends on the mapping mode of the device context, the settings of the origins and extents for the window and viewport, and the world transformation.

[MAKEPOINTS](#)

The MAKEPOINTS macro converts a value that contains the x- and y-coordinates of a point into a POINTS structure.

[MAKEROP4](#)

The MAKEROP4 macro creates a quaternary raster operation code for use with the MaskBlt function.

[MaskBlt](#)

The MaskBlt function combines the color data for the source and destination bitmaps using the specified mask and raster operation.

[ModifyWorldTransform](#)

The ModifyWorldTransform function changes the world transformation for a device context using the specified mode.

[MoveToEx](#)

The MoveToEx function updates the current position to the specified point and optionally returns the previous position.

[OffsetClipRgn](#)

The OffsetClipRgn function moves the clipping region of a device context by the specified offsets.

[OffsetRgn](#)

The OffsetRgn function moves a region by the specified offsets.

[OffsetViewportOrgEx](#)

The OffsetViewportOrgEx function modifies the viewport origin for a device context using the specified horizontal and vertical offsets.

[OffsetWindowOrgEx](#)

The OffsetWindowOrgEx function modifies the window origin for a device context using the specified horizontal and vertical offsets.

[PaintRgn](#)

The PaintRgn function paints the specified region by using the brush currently selected into the device context.

[PALETTEINDEX](#)

The PALETTEINDEX macro accepts an index to a logical-color palette entry and returns a palette-entry specifier consisting of a COLORREF value that specifies the color associated with the given index.

[PALETTERGB](#)

The PALETTERGB macro accepts three values that represent the relative intensities of red, green, and blue and returns a palette-relative red, green, blue (RGB) specifier consisting of 2 in the high-order byte and an RGB value in the three low-order bytes. An application using a color palette can pass this specifier, instead of an explicit RGB value, to functions that expect a color.

[PatBlt](#)

The PatBlt function paints the specified rectangle using the brush that is currently selected into the specified device context. The brush color and the surface color or colors are combined by using the specified raster operation.

[PathToRegion](#)

The PathToRegion function creates a region from the path that is selected into the specified device context. The resulting region uses device coordinates.

[Pie](#)

The Pie function draws a pie-shaped wedge bounded by the intersection of an ellipse and two radials. The pie is outlined by using the current pen and filled by using the current brush.

[PlayEnhMetaFile](#)

The PlayEnhMetaFile function displays the picture stored in the specified enhanced-format metafile.

[PlayEnhMetaFileRecord](#)

The PlayEnhMetaFileRecord function plays an enhanced-metafile record by executing the graphics device interface (GDI) functions identified by the record.

[PlayMetaFile](#)

The PlayMetaFile function displays the picture stored in the given Windows-format metafile on the specified device.

[PlayMetaFileRecord](#)

The PlayMetaFileRecord function plays a Windows-format metafile record by executing the graphics device interface (GDI) function contained within that record.

[PlgBlt](#)

The PlgBlt function performs a bit-block transfer of the bits of color data from the specified rectangle in the source device context to the specified parallelogram in the destination device context.

[PolyBezier](#)

The PolyBezier function draws one or more Bézier curves.

[PolyBezierTo](#)

The PolyBezierTo function draws one or more Bézier curves.

[PolyDraw](#)

The PolyDraw function draws a set of line segments and Bézier curves.

[Polygon](#)

The Polygon function draws a polygon consisting of two or more vertices connected by straight lines. The polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode.

Polyline

The Polyline function draws a series of line segments by connecting the points in the specified array.

PolylineTo

The PolylineTo function draws one or more straight lines.

PolyPolygon

The PolyPolygon function draws a series of closed polygons. Each polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode. The polygons drawn by this function can overlap.

PolyPolyline

The PolyPolyline function draws multiple series of connected line segments.

PolyTextOutA

The PolyTextOut function draws several strings using the font and text colors currently selected in the specified device context.

PolyTextOutW

The PolyTextOut function draws several strings using the font and text colors currently selected in the specified device context.

PtInRegion

The PtInRegion function determines whether the specified point is inside the specified region.

PtVisible

The PtVisible function determines whether the specified point is within the clipping region of a device context.

RealizePalette

The RealizePalette function maps palette entries from the current logical palette to the system palette.

Rectangle

The Rectangle function draws a rectangle. The rectangle is outlined by using the current pen and filled by using the current brush.

RectInRegion

The RectInRegion function determines whether any part of the specified rectangle is within the boundaries of a region.

RectVisible

The RectVisible function determines whether any part of the specified rectangle lies within the clipping region of a device context.

RemoveFontMemResourceEx

The RemoveFontMemResourceEx function removes the fonts added from a memory image file.

[RemoveFontResourceA](#)

The RemoveFontResource function removes the fonts in the specified file from the system font table.

[RemoveFontResourceExA](#)

The RemoveFontResourceEx function removes the fonts in the specified file from the system font table.

[RemoveFontResourceExW](#)

The RemoveFontResourceEx function removes the fonts in the specified file from the system font table.

[RemoveFontResourceW](#)

The RemoveFontResource function removes the fonts in the specified file from the system font table.

[ResetDCA](#)

The ResetDC function updates the specified printer or plotter device context (DC) using the specified information.

[ResetDCW](#)

The ResetDC function updates the specified printer or plotter device context (DC) using the specified information.

[ResizePalette](#)

The ResizePalette function increases or decreases the size of a logical palette based on the specified value.

[RestoreDC](#)

The RestoreDC function restores a device context (DC) to the specified state. The DC is restored by popping state information off a stack created by earlier calls to the SaveDC function.

[RGB](#)

The RGB macro selects a red, green, blue (RGB) color based on the arguments supplied and the color capabilities of the output device.

[RoundRect](#)

The RoundRect function draws a rectangle with rounded corners. The rectangle is outlined by using the current pen and filled by using the current brush.

[SaveDC](#)

The SaveDC function saves the current state of the specified device context (DC) by copying data describing selected objects and graphic modes (such as the bitmap, brush, palette, font, pen, region, drawing mode, and mapping mode) to a context stack.

[ScaleViewportExtEx](#)

The ScaleViewportExtEx function modifies the viewport for a device context using the ratios formed by the specified multiplicands and divisors.

[ScaleWindowExtEx](#)

The ScaleWindowExtEx function modifies the window for a device context using the ratios formed by the specified multiplicands and divisors.

[SelectClipPath](#)

The SelectClipPath function selects the current path as a clipping region for a device context, combining the new region with any existing clipping region using the specified mode.

[SelectClipRgn](#)

The SelectClipRgn function selects a region as the current clipping region for the specified device context.

[SelectObject](#)

The SelectObject function selects an object into the specified device context (DC). The new object replaces the previous object of the same type.

[SelectPalette](#)

The SelectPalette function selects the specified logical palette into a device context.

[SetAbortProc](#)

The SetAbortProc function sets the application-defined abort function that allows a print job to be canceled during spooling.

[SetArcDirection](#)

The SetArcDirection sets the drawing direction to be used for arc and rectangle functions.

[SetBitmapBits](#)

The SetBitmapBits function sets the bits of color data for a bitmap to the specified values.

[SetBitmapDimensionEx](#)

The SetBitmapDimensionEx function assigns preferred dimensions to a bitmap. These dimensions can be used by applications; however, they are not used by the system.

[SetBkColor](#)

The SetBkColor function sets the current background color to the specified color value, or to the nearest physical color if the device cannot represent the specified color value.

[SetBkMode](#)

The SetBkMode function sets the background mix mode of the specified device context. The background mix mode is used with text, hatched brushes, and pen styles that are not solid lines.

[SetBoundsRect](#)

The SetBoundsRect function controls the accumulation of bounding rectangle information for the specified device context.

[SetBrushOrgEx](#)

The SetBrushOrgEx function sets the brush origin that GDI assigns to the next brush an application selects into the specified device context.

[SetColorAdjustment](#)

The SetColorAdjustment function sets the color adjustment values for a device context (DC) using the specified values.

[SetColorSpace](#)

The SetColorSpace function defines the input color space for a given device context.

[SetDCBrushColor](#)

The SetDCBrushColor function sets the current device context (DC) brush color to the specified color value. If the device cannot represent the specified color value, the color is set to the nearest physical color.

[SetDCPenColor](#)

The SetDCPenColor function sets the current device context (DC) pen color to the specified color value. If the device cannot represent the specified color value, the color is set to the nearest physical color.

[SetDeviceGammaRamp](#)

The SetDeviceGammaRamp function sets the gamma ramp on direct color display boards having drivers that support downloadable gamma ramps in hardware.

[SetDIBColorTable](#)

The SetDIBColorTable function sets RGB (red, green, blue) color values in a range of entries in the color table of the DIB that is currently selected into a specified device context.

[SetDIBits](#)

The SetDIBits function sets the pixels in a compatible bitmap (DDB) using the color data found in the specified DIB.

[SetDIBitsToDevice](#)

The SetDIBitsToDevice function sets the pixels in the specified rectangle on the device that is associated with the destination device context using color data from a DIB, JPEG, or PNG image.

[SetEnhMetaFileBits](#)

The SetEnhMetaFileBits function creates a memory-based enhanced-format metafile from the specified data.

[SetGraphicsMode](#)

The SetGraphicsMode function sets the graphics mode for the specified device context.

[SetICMMode](#)

The SetICMMode function causes Image Color Management to be enabled, disabled, or queried on a given device context (DC).

[SetICMProfileA](#)

The SetICMProfile function sets a specified color profile as the output profile for a specified device context (DC).

[SetICMProfileW](#)

The SetICMProfile function sets a specified color profile as the output profile for a specified device context (DC).

[SetLayout](#)

The SetLayout function changes the layout of a device context (DC).

[SetMapMode](#)

The SetMapMode function sets the mapping mode of the specified device context. The mapping mode defines the unit of measure used to transform page-space units into device-space units, and also defines the orientation of the device's x and y axes.

[SetMapperFlags](#)

The SetMapperFlags function alters the algorithm the font mapper uses when it maps logical fonts to physical fonts.

[SetMetaFileBitsEx](#)

The SetMetaFileBitsEx function creates a memory-based Windows-format metafile from the supplied data.

[SetMetaRgn](#)

The SetMetaRgn function intersects the current clipping region for the specified device context with the current metaregion and saves the combined region as the new metaregion for the specified device context.

[SetMiterLimit](#)

The SetMiterLimit function sets the limit for the length of miter joins for the specified device context.

[SetPaletteEntries](#)

The SetPaletteEntries function sets RGB (red, green, blue) color values and flags in a range of entries in a logical palette.

[SetPixel](#)

The SetPixel function sets the pixel at the specified coordinates to the specified color.

[SetPixelFormat](#)

The SetPixelFormat function sets the pixel format of the specified device context to the format specified by the iPixelFormat index.

[SetPixelV](#)

The SetPixelV function sets the pixel at the specified coordinates to the closest approximation of the specified color. The point must be in the clipping region and the visible part of the device surface.

[SetPolyFillMode](#)

The SetPolyFillMode function sets the polygon fill mode for functions that fill polygons.

[SetRectRgn](#)

The SetRectRgn function converts a region into a rectangular region with the specified coordinates.

[SetROP2](#)

The SetROP2 function sets the current foreground mix mode.

[SetStretchBltMode](#)

The SetStretchBltMode function sets the bitmap stretching mode in the specified device context.

[SetSystemPaletteUse](#)

The SetSystemPaletteUse function allows an application to specify whether the system palette contains 2 or 20 static colors.

[SetTextAlign](#)

The SetTextAlign function sets the text-alignment flags for the specified device context.

[SetTextCharacterExtra](#)

The SetTextCharacterExtra function sets the intercharacter spacing. Intercharacter spacing is added to each character, including break characters, when the system writes a line of text.

[SetTextColor](#)

The SetTextColor function sets the text color for the specified device context to the specified color.

[SetTextJustification](#)

The SetTextJustification function specifies the amount of space the system should add to the break characters in a string of text. The space is added when an application calls the TextOut or ExtTextOut functions.

[SetViewportExtEx](#)

Sets the horizontal and vertical extents of the viewport for a device context by using the specified values.

[SetViewportOrgEx](#)

The SetViewportOrgEx function specifies which device point maps to the window origin (0,0).

[SetWindowExtEx](#)

The SetWindowExtEx function sets the horizontal and vertical extents of the window for a device context by using the specified values.

[SetWindowOrgEx](#)

The SetWindowOrgEx function specifies which window point maps to the viewport origin (0,0).

[SetWinMetaFileBits](#)

The SetWinMetaFileBits function converts a metafile from the older Windows format to the new enhanced format and stores the new metafile in memory.

[SetWorldTransform](#)

The SetWorldTransform function sets a two-dimensional linear transformation between world space and page space for the specified device context. This transformation can be used to scale, rotate, shear, or translate graphics output.

[StartDocA](#)

The StartDoc function starts a print job.

[StartDocW](#)

The StartDoc function starts a print job.

[StartPage](#)

The StartPage function prepares the printer driver to accept data.

[StretchBlt](#)

The StretchBlt function copies a bitmap from a source rectangle into a destination rectangle, stretching or compressing the bitmap to fit the dimensions of the destination rectangle, if necessary.

[StretchDIBits](#)

The StretchDIBits function copies the color data for a rectangle of pixels in a DIB, JPEG, or PNG image to the specified destination rectangle.

[StrokeAndFillPath](#)

The StrokeAndFillPath function closes any open figures in a path, strokes the outline of the path by using the current pen, and fills its interior by using the current brush.

[StrokePath](#)

The StrokePath function renders the specified path by using the current pen.

[SwapBuffers](#)

The SwapBuffers function exchanges the front and back buffers if the current pixel format for the window referenced by the specified device context includes a back buffer.

[TextOutA](#)

The TextOut function writes a character string at the specified location, using the currently selected font, background color, and text color.

[TextOutW](#)

The TextOut function writes a character string at the specified location, using the currently selected font, background color, and text color.

[TranslateCharsetInfo](#)

Translates character set information and sets all members of a destination structure to appropriate values.

[TransparentBlt](#)

The TransparentBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

[UnrealizeObject](#)

The UnrealizeObject function resets the origin of a brush or resets a logical palette.

[UpdateColors](#)

The UpdateColors function updates the client area of the specified device context by remapping the current colors in the client area to the currently realized logical palette.

[UpdateICMRegKeyA](#)

The UpdateICMRegKey function manages color profiles and Color Management Modules in the system.

[UpdateICMRegKeyW](#)

The UpdateICMRegKey function manages color profiles and Color Management Modules in the system.

[wglCopyContext](#)

The wglCopyContext function copies selected groups of rendering states from one OpenGL rendering context to another.

[wglCreateContext](#)

The wglCreateContext function creates a new OpenGL rendering context, which is suitable for drawing on the device referenced by hdc. The rendering context has the same pixel format as the device context.

[wglCreateLayerContext](#)

The wglCreateLayerContext function creates a new OpenGL rendering context for drawing to a specified layer plane on a device context.

[wglDeleteContext](#)

The wglDeleteContext function deletes a specified OpenGL rendering context.

[wglDescribeLayerPlane](#)

The wglDescribeLayerPlane function obtains information about the layer planes of a given pixel format.

[wglGetCurrentContext](#)

The wglGetCurrentContext function obtains a handle to the current OpenGL rendering context of the calling thread.

[wglGetCurrentDC](#)

The wglGetCurrentDC function obtains a handle to the device context that is associated with the current OpenGL rendering context of the calling thread.

[wglGetLayerPaletteEntries](#)

Retrieves the palette entries from a given color-index layer plane for a specified device context.

wglGetProcAddress

The wglGetProcAddress function returns the address of an OpenGL extension function for use with the current OpenGL rendering context.

wglGetCurrent

The wglGetCurrent function makes a specified OpenGL rendering context the calling thread's current rendering context.

wglRealizeLayerPalette

The wglRealizeLayerPalette function maps palette entries from a given color-index layer plane into the physical palette or initializes the palette of an RGBA layer plane.

wglSetLayerPaletteEntries

Sets the palette entries in a given color-index layer plane for a specified device context.

wglShareLists

The wglShareLists function enables multiple OpenGL rendering contexts to share a single display-list space.

wglSwapLayerBuffers

The wglSwapLayerBuffers function swaps the front and back buffers in the overlay, underlay, and main planes of the window referenced by a specified device context.

wglUseFontBitmapsA

The wglUseFontBitmaps function creates a set of bitmap display lists for use in the current OpenGL rendering context.

wglUseFontBitmapsW

The wglUseFontBitmaps function creates a set of bitmap display lists for use in the current OpenGL rendering context.

wglUseFontOutlinesA

The wglUseFontOutlines function creates a set of display lists, one for each glyph of the currently selected outline font of a device context, for use with the current rendering context.

wglUseFontOutlinesW

The wglUseFontOutlines function creates a set of display lists, one for each glyph of the currently selected outline font of a device context, for use with the current rendering context.

WidenPath

The WidenPath function redefines the current path as the area that would be painted if the path were stroked using the pen currently selected into the given device context.

Callback functions

[ABORTPROC](#)

The AbortProc function is an application-defined callback function used with the SetAbortProc function.

[ENHMFENUMPROC](#)

The EnhMetaFileProc function is an application-defined callback function used with the EnumEnhMetaFile function.

[GOBJENUMPROC](#)

The EnumObjectsProc function is an application-defined callback function used with the EnumObjects function.

[ICMENUMPROCA](#)

The EnumICMPProfilesProcCallback callback is an application-defined callback function that processes color profile data from EnumICMPProfiles .

[ICMENUMPROCW](#)

The EnumICMPProfilesProcCallback callback is an application-defined callback function that processes color profile data from EnumICMPProfiles .

[LINEDDAPROC](#)

The LineDDAProc function is an application-defined callback function used with the LineDDA function.

[MFENUMPROC](#)

The EnumMetaFileProc function is an application-defined callback function that processes Windows-format metafile records.

Structures

[ABC](#)

The ABC structure contains the width of a character in a TrueType font.

[ABCFLOAT](#)

The ABCFLOAT structure contains the A, B, and C widths of a font character.

[AXESLISTA](#)

The AXESLIST structure contains information on all the axes of a multiple master font.

[AXESLISTW](#)

The AXESLIST structure contains information on all the axes of a multiple master font.

[AXISINFOA](#)

The AXISINFO structure contains information about an axis of a multiple master font.

[AXISINFO](#)

The AXISINFO structure contains information about an axis of a multiple master font.

[BITMAP](#)

The BITMAP structure defines the type, width, height, color format, and bit values of a bitmap.

[BITMAPCOREHEADER](#)

The BITMAPCOREHEADER structure contains information about the dimensions and color format of a DIB.

[BITMAPCOREINFO](#)

The BITMAPCOREINFO structure defines the dimensions and color information for a DIB.

[BITMAPFILEHEADER](#)

The BITMAPFILEHEADER structure contains information about the type, size, and layout of a file that contains a DIB.

[BITMAPINFO](#)

The BITMAPINFO structure defines the dimensions and color information for a DIB.

[BITMAPINFOHEADER](#)

The BITMAPINFOHEADER structure contains information about the dimensions and color format of a device-independent bitmap (DIB).

[BITMAPV4HEADER](#)

The BITMAPV4HEADER structure is the bitmap information header file. It is an extended version of the BITMAPINFOHEADER structure. Applications can use the BITMAPV5HEADER structure for added functionality.

[BITMAPV5HEADER](#)

The BITMAPV5HEADER structure is the bitmap information header file. It is an extended version of the BITMAPINFOHEADER structure.

[BLENDFUNCTION](#)

The BLENDFUNCTION structure controls blending by specifying the blending functions for source and destination bitmaps.

[CHARSETINFO](#)

Contains information about a character set.

[CIEXYZ](#)

The CIEXYZ structure contains the x,y, and z coordinates of a specific color in a specified color space.

[CIEXYZTRIPLE](#)

The CIEXYZTRIPLE structure contains the x,y, and z coordinates of the three colors that correspond to the red, green, and blue endpoints for a specified logical color space.

COLORADJUSTMENT

The COLORADJUSTMENT structure defines the color adjustment values used by the StretchBlt and StretchDIBits functions when the stretch mode is HALFTONE. You can set the color adjustment values by calling the SetColorAdjustment function.

DESIGNVECTOR

The DESIGNVECTOR structure is used by an application to specify values for the axes of a multiple master font.

DEVMODEA

The DEVMODE data structure contains information about the initialization and environment of a printer or a display device.

DEVMODEW

The DEVMODEW structure is used for specifying characteristics of display and print devices in the Unicode (wide) character set.

DIBSECTION

The DIBSECTION structure contains information about a DIB created by calling the CreateDIBSection function.

DISPLAY_DEVICEA

The DISPLAY_DEVICE structure receives information about the display device specified by the iDevNum parameter of the EnumDisplayDevices function.

DISPLAY_DEVICEW

The DISPLAY_DEVICE structure receives information about the display device specified by the iDevNum parameter of the EnumDisplayDevices function.

DISPLAYCONFIG_2DREGION

The DISPLAYCONFIG_2DREGION structure represents a point or an offset in a two-dimensional space.

DISPLAYCONFIG_ADAPTER_NAME

The DISPLAYCONFIG_ADAPTER_NAME structure contains information about the display adapter.

DISPLAYCONFIG_DESKTOP_IMAGE_INFO

The DISPLAYCONFIG_DESKTOP_IMAGE_INFO structure contains information about the image displayed on the desktop.

DISPLAYCONFIG_DEVICE_INFO_HEADER

The DISPLAYCONFIG_DEVICE_INFO_HEADER structure contains display information about the device.

DISPLAYCONFIG_MODE_INFO

The DISPLAYCONFIG_MODE_INFO structure contains either source mode or target mode information.

DISPLAYCONFIG_PATH_INFO

The DISPLAYCONFIG_PATH_INFO structure is used to describe a single path from a target to a source.

[DISPLAYCONFIG_PATH_SOURCE_INFO](#)

The DISPLAYCONFIG_PATH_SOURCE_INFO structure contains source information for a single path.

[DISPLAYCONFIG_PATH_TARGET_INFO](#)

The DISPLAYCONFIG_PATH_TARGET_INFO structure contains target information for a single path.

[DISPLAYCONFIG_RATIONAL](#)

The DISPLAYCONFIG_RATIONAL structure describes a fractional value that represents vertical and horizontal frequencies of a video mode (that is, vertical sync and horizontal sync).

[DISPLAYCONFIG_SDR_WHITE_LEVEL](#)

[DISPLAYCONFIG_SET_TARGET_PERSISTENCE](#)

The DISPLAYCONFIG_SET_TARGET_PERSISTENCE structure contains information about setting the display.

[DISPLAYCONFIG_SOURCE_DEVICE_NAME](#)

The DISPLAYCONFIG_SOURCE_DEVICE_NAME structure contains the GDI device name for the source or view.

[DISPLAYCONFIG_SOURCE_MODE](#)

The DISPLAYCONFIG_SOURCE_MODE structure represents a point or an offset in a two-dimensional space.

[DISPLAYCONFIG_SUPPORT_VIRTUAL_RESOLUTION](#)

The DISPLAYCONFIG_SUPPORT_VIRTUAL_RESOLUTION structure contains information on the state of virtual resolution support for the monitor.

[DISPLAYCONFIG_TARGET_BASE_TYPE](#)

Specifies base output technology info for a given target ID.

[DISPLAYCONFIG_TARGET_DEVICE_NAME](#)

The DISPLAYCONFIG_TARGET_DEVICE_NAME structure contains information about the target.

[DISPLAYCONFIG_TARGET_DEVICE_NAME_FLAGS](#)

The DISPLAYCONFIG_TARGET_DEVICE_NAME_FLAGS structure contains information about a target device.

[DISPLAYCONFIG_TARGET_MODE](#)

The DISPLAYCONFIG_TARGET_MODE structure describes a display path target mode.

[DISPLAYCONFIG_TARGET_PREFERRED_MODE](#)

The DISPLAYCONFIG_TARGET_PREFERRED_MODE structure contains information about the preferred mode of a display.

[DISPLAYCONFIG_VIDEO_SIGNAL_INFO](#)

The DISPLAYCONFIG_VIDEO_SIGNAL_INFO structure contains information about the video signal for a display.

[DOCINFOA](#)

The DOCINFO structure contains the input and output file names and other information used by the StartDoc function.

[DOCINFOW](#)

The DOCINFO structure contains the input and output file names and other information used by the StartDoc function.

[DRAWPATRECT](#)

The DRAWPATRECT structure defines a rectangle to be created.

[EMR](#)

The EMR structure provides the base structure for all enhanced metafile records. An enhanced metafile record contains the parameters for a specific GDI function used to create part of a picture in an enhanced format metafile.

[EMRABORTPATH](#)

Contains data for the AbortPath, BeginPath, EndPath, CloseFigure, FlattenPath, WidenPath, SetMetaRgn, SaveDC, and RealizePalette enhanced metafile records.

[EMRALPHABLEND](#)

The EMRALPHABLEND structure contains members for the AlphaBlend enhanced metafile record.

[EMRANGLEARC](#)

The EMRANGLEARC structure contains members for the AngleArc enhanced metafile record.

[EMRARC](#)

The EMRARC, EMRARCTO, EMRCHORD, and EMRPIE structures contain members for the Arc, ArcTo, Chord, and Pie enhanced metafile records.

[EMRBITBLT](#)

The EMRBITBLT structure contains members for the BitBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

[EMRCOLORCORRECTPALETTE](#)

The EMRCOLORCORRECTPALETTE structure contains members for the ColorCorrectPalette enhanced metafile record.

[EMRCOLORMATCHTOTARGET](#)

The EMRCOLORMATCHTOTARGET structure contains members for the ColorMatchToTarget enhanced metafile record.

[EMRCREATEBRUSHINDIRECT](#)

The EMRCREATEBRUSHINDIRECT structure contains members for the CreateBrushIndirect enhanced metafile record.

[EMRCREATECOLORSPACE](#)

The EMRCREATECOLORSPACE structure contains members for the CreateColorSpace enhanced metafile record.

[EMRCREATECOLORSPACEW](#)

The EMRCREATECOLORSPACEW structure contains members for the CreateColorSpace enhanced metafile record. It differs from EMRCREATECOLORSPACE in that it has a Unicode logical color space and also has an optional array containing raw source profile data.

[EMRCREATEDIBPATTERNBRUSHPT](#)

The EMRCREATEDIBPATTERNBRUSHPT structure contains members for the CreateDIBPatternBrushPt enhanced metafile record. The BITMAPINFO structure is followed by the bitmap bits that form a packed device-independent bitmap (DIB).

[EMRCREATEMONOBRUSH](#)

The EMRCREATEMONOBRUSH structure contains members for the CreatePatternBrush (when passed a monochrome bitmap) or CreateDIBPatternBrush (when passed a monochrome DIB) enhanced metafile records.

[EMRCREATEPALETTE](#)

The EMRCREATEPALETTE structure contains members for the CreatePalette enhanced metafile record.

[EMRCREATEPEN](#)

The EMRCREATEPEN structure contains members for the CreatePen enhanced metafile record.

[EMRELLIPSE](#)

The EMRELLIPSE and EMRRECTANGLE structures contain members for the Ellipse and Rectangle enhanced metafile records.

[EMREOF](#)

The EMREOF structure contains data for the enhanced metafile record that indicates the end of the metafile.

[EMREXCLUDECLIPRECT](#)

The EMREXCLUDECLIPRECT and EMRINTERSECTCLIPRECT structures contain members for the ExcludeClipRect and IntersectClipRect enhanced metafile records.

[EMREXTCREATEFONTINDIRECTW](#)

The EMREXTCREATEFONTINDIRECTW structure contains members for the CreateFontIndirect enhanced metafile record.

[EMREXTCREATEPEN](#)

The EMREXTCREATEPEN structure contains members for the ExtCreatePen enhanced metafile record. If the record contains a BITMAPINFO structure, it is followed by the bitmap bits that form a packed device-independent bitmap (DIB).

[EMREXTFLOODFILL](#)

The EMREXTFLOODFILL structure contains members for the ExtFloodFill enhanced metafile record.

[EMREXTSELECTCLIPRGN](#)

The EMREXTSELECTCLIPRGN structure contains members for the ExtSelectClipRgn enhanced metafile record.

[EMREXTTEXTOUTA](#)

The EMREXTTEXTOUTA and EMREXTTEXTOUTW structures contain members for the ExtTextOut, TextOut, or DrawText enhanced metafile records.

[EMRFILLPATH](#)

The EMRFILLPATH,♦EMRSTROKEANDFILLPATH,♦ and EMRSTROKEPATH structures contain members for the FillPath, StrokeAndFillPath, and StrokePath enhanced metafile records.

[EMRFILLRGN](#)

The EMRFILLRGN structure contains members for the FillRgn enhanced metafile record.

[EMRFORMAT](#)

The EMRFORMAT structure contains information that identifies graphics data in an enhanced metafile. A GDICOMMENT_MULTIFORMATS enhanced metafile public comment contains an array of EMRFORMAT structures.

[EMRFRAMERGN](#)

The EMRFRAMERGN structure contains members for the FrameRgn enhanced metafile record.

[EMRGDICOMMENT](#)

The EMRGDICOMMENT structure contains application-specific data.

[EMRGLSBOUNDEDRECORD](#)

The EMRGLSBOUNDEDRECORD structure contains members for an enhanced metafile record generated by OpenGL functions. It contains data for OpenGL functions with information in pixel units that must be scaled when playing the metafile.

[EMRGLSRECORD](#)

The EMRGLSRECORD structure contains members for an enhanced metafile record generated by OpenGL functions. It contains data for OpenGL functions that scale automatically to the OpenGL viewport.

[EMRGRADIENTFILL](#)

The EMRGRADIENTFILL structure contains members for the GradientFill enhanced metafile record.

[EMRINVERTRGN](#)

The EMRINVERTRGN and EMRPAINTRGN structures contain members for the InvertRgn and PaintRgn enhanced metafile records.

[EMRLINETO](#)

The EMRLINETO and EMRMOVETOEX structures contains members for the LineTo and MoveToEx enhanced metafile records.

[EMRMASKBLT](#)

The EMRMASKBLT structure contains members for the MaskBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

[EMRMODIFYWORLDTRANSFORM](#)

The EMRMODIFYWORLDTRANSFORM structure contains members for the ModifyWorldTransform enhanced metafile record.

[EMROFFSETCLIPRGN](#)

The EMROFFSETCLIPRGN structure contains members for the OffsetClipRgn enhanced metafile record.

[EMRPIXELFORMAT](#)

The EMRPIXELFORMAT structure contains the members for the SetPixelFormat enhanced metafile record. The pixel format information in ENHMETAHEADER refers to this structure.

[EMRPLGBT](#)

The EMRPLGBT structure contains members for the PlgBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

[EMRPOLYDRAW](#)

The EMRPOLYDRAW structure contains members for the PolyDraw enhanced metafile record.

[EMRPOLYDRAW16](#)

The EMRPOLYDRAW16 structure contains members for the PolyDraw enhanced metafile record.

[EMRPOLYLINE](#)

The EMRPOLYLINE, EMRPOLYBEZIER, EMRPOLYGON, EMRPOLYBEZIERTO, and EMRPOLYLINETO structures contain members for the Polyline, PolyBezier, Polygon, PolyBezierTo, and PolylineTo enhanced metafile records.

[EMRPOLYLINE16](#)

The EMRPOLYLINE16, EMRPOLYBEZIER16, EMRPOLYGON16, EMRPOLYBEZIERTO16, and EMRPOLYLINETO16 structures contain members for the Polyline, PolyBezier, Polygon, PolyBezierTo, and PolylineTo enhanced metafile records.

[EMRPOLYPOLYLINE](#)

The EMRPOLYPOLYLINE and EMRPOLYPOLYGON structures contain members for the PolyPolyline and PolyPolygon enhanced metafile records.

[EMRPOLYPOLYLINE16](#)

The EMRPOLYPOLYLINE16 and EMRPOLYPOLYGON16 structures contain members for the PolyPolyline and PolyPolygon enhanced metafile records.

[EMRPOLYTEXTOUTA](#)

The EMRPOLYTEXTOUTA and EMRPOLYTEXTOUTW structures contain members for the PolyTextOut enhanced metafile record.

[EMRRESIZEPALETTE](#)

The EMRRESIZEPALETTE structure contains members for the ResizePalette enhanced metafile record.

[EMRRESTOREDC](#)

The EMRRESTOREDC structure contains members for the RestoreDC enhanced metafile record.

[EMRROUNDRECT](#)

The EMRROUNDRECT structure contains members for the RoundRect enhanced metafile record.

[EMRSCALEVIEWPORTEXTEX](#)

The EMRSCALEVIEWPORTEXTEX and EMRSCALEWINDOWEXTEX structures contain members for the ScaleViewportExtEx and ScaleWindowExtEx enhanced metafile records.

[EMRSELECTCLIPPATH](#)

Contains parameters for the SelectClipPath, SetBkMode, SetMapMode, SetPolyFillMode, SetROP2, SetStretchBltMode, SetTextAlign, SetICMMode , and SetLayout enhanced metafile records.

[EMRSELECTOBJECT](#)

The EMRSELECTOBJECT and EMRDELETEOBJECT structures contain members for the SelectObject and DeleteObject enhanced metafile records.

[EMRSELECTPALETTE](#)

The EMRSELECTPALETTE structure contains members for the SelectPalette enhanced metafile record. Note that the bForceBackground parameter in SelectPalette is always recorded as TRUE, which causes the palette to be realized as a background palette.

[EMRSETARCDIRECTION](#)

The EMRSETARCDIRECTION structure contains members for the SetArcDirection enhanced metafile record.

[EMRSETBKCOLOR](#)

The EMRSETBKCOLOR and EMRSETTEXTCOLOR structures contain members for the SetBkColor and SetTextColor enhanced metafile records.

[EMRSETCOLORADJUSTMENT](#)

The EMRSETCOLORADJUSTMENT structure contains members for the SetColorAdjustment enhanced metafile record.

[EMRSETCOLORSPACE](#)

The EMRSETCOLORSPACE, EMRSELECTCOLORSPACE, and EMRDELETECOLORSPACE structures contain members for the SetColorSpace and DeleteColorSpace enhanced metafile records.

[EMRSETDIBITSTODEVICE](#)

The EMRSETDIBITSTODEVICE structure contains members for the SetDIBitsToDevice enhanced metafile record.

[EMRSETICMPROFILE](#)

The EMRSETICMPROFILE structure contains members for the SetICMProfile enhanced metafile record.

[EMRSETMAPPERFLAGS](#)

The EMRSETMAPPERFLAGS structure contains members for the SetMapperFlags enhanced metafile record.

[EMRSETMITERLIMIT](#)

The EMRSETMITERLIMIT structure contains members for the SetMiterLimit enhanced metafile record.

[EMRSETPALETTEENTRIES](#)

The EMRSETPALETTEENTRIES structure contains members for the SetPaletteEntries enhanced metafile record.

[EMRSETPIXELV](#)

The EMRSETPIXELV structure contains members for the SetPixelV enhanced metafile record. When an enhanced metafile is created, calls to SetPixel are also recorded in this record.

[EMRSETVIEWPORTEXTEX](#)

The EMRSETVIEWPORTEXTEX and EMRSETWINDOWEXTEX structures contain members for the SetViewportExtEx and SetWindowExtEx enhanced metafile records.

[EMRSETVIEWPORTORGEX](#)

The EMRSETVIEWPORTORGEX, EMRSETWINDOWORGEX, and EMRSETBRUSHORGEX structures contain members for the SetViewportOrgEx, SetWindowOrgEx, and SetBrushOrgEx enhanced metafile records.

[EMRSETWORLDTRANSFORM](#)

The EMRSETWORLDTRANSFORM structure contains members for the SetWorldTransform enhanced metafile record.

[EMRSTRETCHBLT](#)

The EMRSTRETCHBLT structure contains members for the StretchBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

[EMRSTRETCHDIBITS](#)

The EMRSTRETCHDIBITS structure contains members for the StretchDIBits enhanced metafile record.

[EMRTEXT](#)

The EMRTEXT structure contains members for text output.

[EMRTRANSPARENTBLT](#)

The EMRTRANSPARENTBLT structure contains members for the TransparentBlt enhanced metafile record.

[ENHMETAHEADER](#)

The ENHMETAHEADER structure contains enhanced-metafile data such as the dimensions of the picture stored in the enhanced metafile, the count of records in the enhanced metafile, the resolution of the device on which the picture was created, and so on. This structure is always the first record in an enhanced metafile.

[ENHMETARECORD](#)

The ENHMETARECORD structure contains data that describes a graphics device interface (GDI) function used to create part of a picture in an enhanced-format metafile.

[ENUMLOGFONTA](#)

The ENUMLOGFONT structure defines the attributes of a font, the complete name of a font, and the style of a font.

[ENUMLOGFONTEXA](#)

The ENUMLOGFONTEX structure contains information about an enumerated font.

[ENUMLOGFONTEXDVA](#)

The ENUMLOGFONTEXDV structure contains the information used to create a font.

[ENUMLOGFONTEXDVW](#)

The ENUMLOGFONTEXDV structure contains the information used to create a font.

[ENUMLOGFONTEXW](#)

The ENUMLOGFONTEX structure contains information about an enumerated font.

[ENUMLOGFONTW](#)

The ENUMLOGFONT structure defines the attributes of a font, the complete name of a font, and the style of a font.

[ENUMTEXTMETRICA](#)

The ENUMTEXTMETRIC structure contains information about a physical font.

[ENUMTEXTMETRICW](#)

The ENUMTEXTMETRIC structure contains information about a physical font.

[EXTLOGFONTA](#)

The EXTLOGFONT structure defines the attributes of a font.

[EXTLOGFONTW](#)

The EXTLOGFONT structure defines the attributes of a font.

[EXTLOGOPEN](#)

The EXTLOGOPEN structure defines the pen style, width, and brush attributes for an extended pen.

[FIXED](#)

The FIXED structure contains the integral and fractional parts of a fixed-point real number.

[FONTSIGNATURE](#)

Contains information identifying the code pages and Unicode subranges for which a given font provides glyphs.

[GCP_RESULTS](#)

The GCP_RESULTS structure contains information about characters in a string. This structure receives the results of the GetCharacterPlacement function. For some languages, the first element in the arrays may contain more, language-dependent information.

[GCP_RESULTSW](#)

The GCP_RESULTS structure contains information about characters in a string. This structure receives the results of the GetCharacterPlacement function. For some languages, the first element in the arrays may contain more, language-dependent information.

[GLYPHMETRICS](#)

The GLYPHMETRICS structure contains information about the placement and orientation of a glyph in a character cell.

[GLYPHMETRICSFLOAT](#)

The GLYPHMETRICSFLOAT structure contains information about the placement and orientation of a glyph in a character cell.

[GLYPHSET](#)

The GLYPHSET structure contains information about a range of Unicode code points.

[GRADIENT_RECT](#)

The GRADIENT_RECT structure specifies the index of two vertices in the pVertex array in the GradientFill function. These two vertices form the upper-left and lower-right boundaries of a rectangle.

[GRADIENT_TRIANGLE](#)

The GRADIENT_TRIANGLE structure specifies the index of three vertices in the pVertex array in the GradientFill function. These three vertices form one triangle.

[HANDLETABLE](#)

The HANDLETABLE structure is an array of handles, each of which identifies a graphics device interface (GDI) object.

[KERNINGPAIR](#)

The KERNINGPAIR structure defines a kerning pair.

[LAYERPLANEDESCRIPTOR](#)

The LAYERPLANEDESCRIPTOR structure describes the pixel format of a drawing surface.

[LOCALESIGNATURE](#)

Contains extended font signature information, including two code page bitfields (CPBs) that define the default and supported character sets and code pages. This structure is typically used to represent the relationships between font coverage and locales.

[LOGBRUSH](#)

The LOGBRUSH structure defines the style, color, and pattern of a physical brush. It is used by the CreateBrushIndirect and ExtCreatePen functions.

[LOGBRUSH32](#)

The LOGBRUSH32 structure defines the style, color, and pattern of a physical brush.

[LOGCOLORSPACEA](#)

The LOGCOLORSPACE structure contains information that defines a logical color space.

[LOGCOLORSPACEW](#)

The LOGCOLORSPACE structure contains information that defines a logical color space.

[LOGFONTA](#)

The LOGFONT structure defines the attributes of a font.

[LOGFONTW](#)

The LOGFONT structure defines the attributes of a font.

[LOGPALETTE](#)

The LOGPALETTE structure defines a logical palette.

[LOGPEN](#)

The LOGPEN structure defines the style, width, and color of a pen. The CreatePenIndirect function uses the LOGPEN structure.

[MAT2](#)

The MAT2 structure contains the values for a transformation matrix used by the GetGlyphOutline function.

[METAFILEPICT](#)

Defines the metafile picture format used for exchanging metafile data through the clipboard.

[METAHEADER](#)

The METAHEADER structure contains information about a Windows-format metafile.

[METARECORD](#)

The METARECORD structure contains a Windows-format metafile record.

[NEWTEXTMETRICA](#)

The NEWTEXTMETRIC structure contains data that describes a physical font.

[NEWTEXTMETRICEXA](#)

The NEWTEXTMETRICEX structure contains information about a physical font.

[NEWTEXTMETRICEXW](#)

The NEWTEXTMETRICEX structure contains information about a physical font.

[NEWTEXTMETRICW](#)

The NEWTEXTMETRIC structure contains data that describes a physical font.

[OUTLINETEXTMETRICA](#)

The OUTLINETEXTMETRIC structure contains metrics describing a TrueType font.

[OUTLINETEXTMETRICW](#)

The OUTLINETEXTMETRIC structure contains metrics describing a TrueType font.

[PALETTEENTRY](#)

Specifies the color and usage of an entry in a logical palette.

[PANOSE](#)

The PANOSE structure describes the PANOSE font-classification values for a TrueType font. These characteristics are then used to associate the font with other fonts of similar appearance but different names.

[PIXELFORMATDESCRIPTOR](#)

The PIXELFORMATDESCRIPTOR structure describes the pixel format of a drawing surface.

[POINTFLOAT](#)

The POINTFLOAT structure contains the x and y coordinates of a point.

[POINTFX](#)

The POINTFX structure contains the coordinates of points that describe the outline of a character in a TrueType font.

[POLYTEXTA](#)

The POLYTEXT structure describes how the PolyTextOut function should draw a string of text.

[POLYTEXTW](#)

The POLYTEXT structure describes how the PolyTextOut function should draw a string of text.

[PSFEATURE_CUSTPAPER](#)

The PSFEATURE_CUSTPAPER structure contains information about a custom paper size for a PostScript driver. This structure is used with the GET_PS_FEATURESETTING printer escape function.

[PSFEATURE_OUTPUT](#)

The PSFEATURE_OUTPUT structure contains information about PostScript driver output options. This structure is used with the GET_PS_FEATURESETTING printer escape function.

[PSINJECTDATA](#)

The PSINJECTDATA structure is a header for the input buffer used with the POSTSCRIPT_INJECTION printer escape function.

[RASTERIZER_STATUS](#)

The RASTERIZER_STATUS structure contains information about whether TrueType is installed. This structure is filled when an application calls the GetRasterizerCaps function.

[RGBQUAD](#)

The RGBQUAD structure describes a color consisting of relative intensities of red, green, and blue.

[RGBTRIPLE](#)

The RGBTRIPLE structure describes a color consisting of relative intensities of red, green, and blue. The bmcColors member of the BITMAPCOREINFO structure consists of an array of RGBTRIPLE structures.

[RGNDATA](#)

The RGNDATA structure contains a header and an array of rectangles that compose a region. The rectangles are sorted top to bottom, left to right. They do not overlap.

[RGNDATAHEADER](#)

The RGNDATAHEADER structure describes the data returned by the GetRegionData function.

[TEXTMETRICA](#)

The TEXTMETRIC structure contains basic information about a physical font. All sizes are specified in logical units; that is, they depend on the current mapping mode of the display context.

[TEXTMETRICW](#)

The TEXTMETRIC structure contains basic information about a physical font. All sizes are specified in logical units; that is, they depend on the current mapping mode of the display context.

[TRIVERTEX](#)

The TRIVERTEX structure contains color information and position information.

[TTPOLYCURVE](#)

The TTPOLYCURVE structure contains information about a curve in the outline of a TrueType character.

[TTPOLYGONHEADER](#)

The TTPOLYGONHEADER structure specifies the starting position and type of a contour in a TrueType character outline.

[WCRANGE](#)

The WCRANGE structure specifies a range of Unicode characters.

[XFORM](#)

The XFORM structure specifies a world-space to page-space transformation.

Enumerations

[DISPLAYCONFIG_DEVICE_INFO_TYPE](#)

The DISPLAYCONFIG_DEVICE_INFO_TYPE enumeration specifies the type of display device info to configure or obtain through the DisplayConfigSetDeviceInfo or DisplayConfigGetDeviceInfo function.

[DISPLAYCONFIG_MODE_INFO_TYPE](#)

The DISPLAYCONFIG_MODE_INFO_TYPE enumeration specifies that the information that is contained within the DISPLAYCONFIG_MODE_INFO structure is either source or target mode.

[DISPLAYCONFIG_PIXELFORMAT](#)

The DISPLAYCONFIG_PIXELFORMAT enumeration specifies pixel format in various bits per pixel (BPP) values.

[DISPLAYCONFIG_ROTATION](#)

The DISPLAYCONFIG_ROTATION enumeration specifies the clockwise rotation of the display.

[DISPLAYCONFIG_SCALING](#)

The DISPLAYCONFIG_SCALING enumeration specifies the scaling transformation applied to content displayed on a video present network (VidPN) present path.

[DISPLAYCONFIG_SCANLINE_ORDERING](#)

The DISPLAYCONFIG_SCANLINE_ORDERING enumeration specifies the method that the display uses to create an image on a screen.

[DISPLAYCONFIG_TOPOLOGY_ID](#)

The DISPLAYCONFIG_TOPOLOGY_ID enumeration specifies the type of display topology.

[DISPLAYCONFIG_VIDEO_OUTPUT_TECHNOLOGY](#)

The DISPLAYCONFIG_VIDEO_OUTPUT_TECHNOLOGY enumeration specifies the target's connector type.

ABC structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The ABC structure contains the width of a character in a TrueType font.

Syntax

```
typedef struct _ABC {  
    int abcA;  
    UINT abcB;  
    int abcC;  
} ABC, *PABC, *NPABC, *LPABC;
```

Members

abcA

The A spacing of the character. The A spacing is the distance to add to the current position before drawing the character glyph.

abcB

The B spacing of the character. The B spacing is the width of the drawn portion of the character glyph.

abcC

The C spacing of the character. The C spacing is the distance to add to the current position to provide white space to the right of the character glyph.

Remarks

The total width of a character is the summation of the A, B, and C spaces. Either the A or the C space can be negative to indicate underhangs or overhangs.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

ABCFLOAT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The ABCFLOAT structure contains the A, B, and C widths of a font character.

Syntax

```
typedef struct _ABCFLOAT {
    FLOAT abcfa;
    FLOAT abcfb;
    FLOAT abcc;
} ABCFLOAT, *PABCFLOAT, *NPABCFLOAT, *LPABCFLOAT;
```

Members

`abcfa`

The A spacing of the character. The A spacing is the distance to add to the current position before drawing the character glyph.

`abcfb`

The B spacing of the character. The B spacing is the width of the drawn portion of the character glyph.

`abcc`

The C spacing of the character. The C spacing is the distance to add to the current position to provide white space to the right of the character glyph.

Remarks

The A, B, and C widths are measured along the base line of the font.

The character increment (total width) of a character is the sum of the A, B, and C spaces. Either the A or the C space can be negative to indicate underhangs or overhangs.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

`GetCharABCWidthsFloat`

AbortPath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **AbortPath** function closes and discards any paths in the specified device context.

Syntax

```
BOOL AbortPath(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

Handle to the device context from which a path will be discarded.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

If there is an open path bracket in the given device context, the path bracket is closed and the path is discarded.

If there is a closed path in the device context, the path is discarded.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[EndPath](#)

[Path Functions](#)

[Paths Overview](#)

AddFontMemResourceEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **AddFontMemResourceEx** function adds the font resource from a memory image to the system.

Syntax

```
HANDLE AddFontMemResourceEx(
    [in] PVOID pFileView,
    [in] DWORD cjSize,
    [in] PVOID pvResrvd,
    [in] DWORD *pNumFonts
);
```

Parameters

[in] `pFileView`

A pointer to a font resource.

[in] `cjSize`

The number of bytes in the font resource that is pointed to by `pbFont`.

[in] `pvResrvd`

Reserved. Must be 0.

[in] `pNumFonts`

A pointer to a variable that specifies the number of fonts installed.

Return value

If the function succeeds, the return value specifies the handle to the font added. This handle uniquely identifies the fonts that were installed on the system. If the function fails, the return value is zero. No extended error information is available.

Remarks

This function allows an application to get a font that is embedded in a document or a webpage. A font that is added by **AddFontMemResourceEx** is always private to the process that made the call and is not enumerable.

A memory image can contain more than one font. When this function succeeds, `pcFonts` is a pointer to a **DWORD** whose value is the number of fonts added to the system as a result of this call. For example, this number could be 2 for the vertical and horizontal faces of an Asian font.

When the function succeeds, the caller of this function can free the memory pointed to by `pbFont` because the system has made its own copy of the memory. To remove the fonts that were installed, call [RemoveFontMemResourceEx](#). However, when the process goes away, the system will unload the fonts even if the process did not call [RemoveFontMemResource](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DESIGNVECTOR](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[RemoveFontMemResourceEx](#)

[SendMessage](#)

AddFontResourceA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **AddFontResource** function adds the font resource from the specified file to the system font table. The font can subsequently be used for text output by any application.

To mark a font as private or not enumerable, use the [AddFontResourceEx](#) function.

Syntax

```
int AddFontResourceA(
    [in] LPCSTR unnamedParam1
);
```

Parameters

[in] `unnamedParam1`

A pointer to a null-terminated character string that contains a valid font file name. This parameter can specify any of the following files.

FILE EXTENSION	MEANING
.fon	Font resource file.
.fnt	Raw bitmap font file.
.ttf	Raw TrueType file.
.ttc	East Asian Windows: TrueType font collection.
.fot	TrueType resource file.
.otf	PostScript OpenType font.
.mmmm	Multiple master Type1 font resource file. It must be used with .pfm and .pfb files.
.pfb	Type 1 font bits file. It is used with a .pfm file.

.pfm	Type 1 font metrics file. It is used with a .pfb file.
------	--

To add a font whose information comes from several resource files, have */pszFileName* point to a string with the file names separated by a "|" --for example, abcxxxx.pfm | abcxxxx.pfb.

Return value

If the function succeeds, the return value specifies the number of fonts added.

If the function fails, the return value is zero. No extended error information is available.

Remarks

Any application that adds or removes fonts from the system font table should notify other windows of the change by sending a [WM_FONTCHANGE](#) message to all top-level windows in the operating system. The application should send this message by calling the [SendMessage](#) function and setting the *hwnd* parameter to [HWND_BROADCAST](#).

When an application no longer needs a font resource that it loaded by calling the [AddFontResource](#) function, it must remove that resource by calling the [RemoveFontResource](#) function.

This function installs the font only for the current session. When the system restarts, the font will not be present. To have the font installed even after restarting the system, the font must be listed in the registry.

A font listed in the registry and installed to a location other than the %windir%\fonts\ folder cannot be modified, deleted, or replaced as long as it is loaded in any session. In order to change one of these fonts, it must first be removed by calling [RemoveFontResource](#), removed from the font registry ([HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts](#)), and the system restarted. After restarting the system, the font will no longer be loaded and can be changed.

NOTE

The wingdi.h header defines AddFontResource as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[AddFontResourceEx](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[RemoveFontResource](#)

[SendMessage](#)

AddFontResourceExA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **AddFontResourceEx** function adds the font resource from the specified file to the system. Fonts added with the **AddFontResourceEx** function can be marked as private and not enumerable.

Syntax

```
int AddFontResourceExA(
    [in] LPCSTR name,
    [in] DWORD   f1,
    [in] PVOID   res
);
```

Parameters

[in] name

A pointer to a null-terminated character string that contains a valid font file name. This parameter can specify any of the following files.

FILE EXTENSION	MEANING
.fon	Font resource file.
.fnt	Raw bitmap font file.
.ttf	Raw TrueType file.
.ttc	East Asian Windows: TrueType font collection.
.fot	TrueType resource file.
.otf	PostScript OpenType font.
.mmmm	multiple master Type1 font resource file. It must be used with .pfm and .pfb files.
.pfb	Type 1 font bits file. It is used with a .pfm file.

.pfm	Type 1 font metrics file. It is used with a .pfb file.
------	--

To add a font whose information comes from several resource files, point *lpszFileName* to a string with the file names separated by a | --for example, abcxxxx.pfm | abcxxxx.pfb.

[in] *f1*

The characteristics of the font to be added to the system. This parameter can be one of the following values.

VALUE	MEANING
FR_PRIVATE	Specifies that only the process that called the AddFontResourceEx function can use this font. When the font name matches a public font, the private font will be chosen. When the process terminates, the system will remove all fonts installed by the process with the AddFontResourceEx function.
FR_NOT_ENUM	Specifies that no process, including the process that called the AddFontResourceEx function, can enumerate this font.

[in] *res*

Reserved. Must be zero.

Return value

If the function succeeds, the return value specifies the number of fonts added.

If the function fails, the return value is zero. No extended error information is available.

Remarks

This function allows a process to use fonts without allowing other processes access to the fonts.

When an application no longer needs a font resource it loaded by calling the **AddFontResourceEx** function, it must remove the resource by calling the **RemoveFontResourceEx** function.

This function installs the font only for the current session. When the system restarts, the font will not be present. To have the font installed even after restarting the system, the font must be listed in the registry.

A font listed in the registry and installed to a location other than the %windir%\fonts\ folder cannot be modified, deleted, or replaced as long as it is loaded in any session. In order to change one of these fonts, it must first be removed by calling **RemoveFontResource**, removed from the font registry (**HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts**), and the system restarted. After restarting the system, the font will no longer be loaded and can be changed.

NOTE

The wingdi.h header defines **AddFontResourceEx** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[RemoveFontResourceEx](#)

[SendMessage](#)

AddFontResourceExW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **AddFontResourceEx** function adds the font resource from the specified file to the system. Fonts added with the **AddFontResourceEx** function can be marked as private and not enumerable.

Syntax

```
int AddFontResourceExW(
    [in] LPCWSTR name,
    [in] DWORD    f1,
    [in] PVOID    res
);
```

Parameters

[in] name

A pointer to a null-terminated character string that contains a valid font file name. This parameter can specify any of the following files.

FILE EXTENSION	MEANING
.fon	Font resource file.
.fnt	Raw bitmap font file.
.ttf	Raw TrueType file.
.ttc	East Asian Windows: TrueType font collection.
.fot	TrueType resource file.
.otf	PostScript OpenType font.
.mmmm	multiple master Type1 font resource file. It must be used with .pfm and .pfb files.
.pfb	Type 1 font bits file. It is used with a .pfm file.

.pfm	Type 1 font metrics file. It is used with a .pfb file.
------	--

To add a font whose information comes from several resource files, point *lpszFileName* to a string with the file names separated by a | --for example, abcxxxx.pfm | abcxxxx.pfb.

[in] *f1*

The characteristics of the font to be added to the system. This parameter can be one of the following values.

VALUE	MEANING
FR_PRIVATE	Specifies that only the process that called the AddFontResourceEx function can use this font. When the font name matches a public font, the private font will be chosen. When the process terminates, the system will remove all fonts installed by the process with the AddFontResourceEx function.
FR_NOT_ENUM	Specifies that no process, including the process that called the AddFontResourceEx function, can enumerate this font.

[in] *res*

Reserved. Must be zero.

Return value

If the function succeeds, the return value specifies the number of fonts added.

If the function fails, the return value is zero. No extended error information is available.

Remarks

This function allows a process to use fonts without allowing other processes access to the fonts.

When an application no longer needs a font resource it loaded by calling the **AddFontResourceEx** function, it must remove the resource by calling the **RemoveFontResourceEx** function.

This function installs the font only for the current session. When the system restarts, the font will not be present. To have the font installed even after restarting the system, the font must be listed in the registry.

A font listed in the registry and installed to a location other than the %windir%\fonts\ folder cannot be modified, deleted, or replaced as long as it is loaded in any session. In order to change one of these fonts, it must first be removed by calling **RemoveFontResource**, removed from the font registry (**HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts**), and the system restarted. After restarting the system, the font will no longer be loaded and can be changed.

NOTE

The wingdi.h header defines **AddFontResourceEx** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[RemoveFontResourceEx](#)

[SendMessage](#)

AddFontResourceW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **AddFontResource** function adds the font resource from the specified file to the system font table. The font can subsequently be used for text output by any application.

To mark a font as private or not enumerable, use the [AddFontResourceEx](#) function.

Syntax

```
int AddFontResourceW(
    [in] LPCWSTR unnamedParam1
);
```

Parameters

[in] unnamedParam1

A pointer to a null-terminated character string that contains a valid font file name. This parameter can specify any of the following files.

FILE EXTENSION	MEANING
.fon	Font resource file.
.fnt	Raw bitmap font file.
.ttf	Raw TrueType file.
.ttc	East Asian Windows: TrueType font collection.
.fot	TrueType resource file.
.otf	PostScript OpenType font.
.mmmm	Multiple master Type1 font resource file. It must be used with .pfm and .pfb files.
.pfb	Type 1 font bits file. It is used with a .pfm file.

.pfm	Type 1 font metrics file. It is used with a .pfb file.
------	--

To add a font whose information comes from several resource files, have */pszFileName* point to a string with the file names separated by a "|" --for example, abcxxxx.pfm | abcxxxx.pfb.

Return value

If the function succeeds, the return value specifies the number of fonts added.

If the function fails, the return value is zero. No extended error information is available.

Remarks

Any application that adds or removes fonts from the system font table should notify other windows of the change by sending a [WM_FONTCHANGE](#) message to all top-level windows in the operating system. The application should send this message by calling the [SendMessage](#) function and setting the *hwnd* parameter to [HWND_BROADCAST](#).

When an application no longer needs a font resource that it loaded by calling the [AddFontResource](#) function, it must remove that resource by calling the [RemoveFontResource](#) function.

This function installs the font only for the current session. When the system restarts, the font will not be present. To have the font installed even after restarting the system, the font must be listed in the registry.

A font listed in the registry and installed to a location other than the %windir%\fonts\ folder cannot be modified, deleted, or replaced as long as it is loaded in any session. In order to change one of these fonts, it must first be removed by calling [RemoveFontResource](#), removed from the font registry ([HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts](#)), and the system restarted. After restarting the system, the font will no longer be loaded and can be changed.

NOTE

The wingdi.h header defines AddFontResource as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[AddFontResourceEx](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[RemoveFontResource](#)

[SendMessage](#)

AlphaBlend function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **AlphaBlend** function displays bitmaps that have transparent or semitransparent pixels.

Syntax

```
BOOL AlphaBlend(
    [in] HDC         hdcDest,
    [in] int          xoriginDest,
    [in] int          yoriginDest,
    [in] int          wDest,
    [in] int          hDest,
    [in] HDC         hdcSrc,
    [in] int          xoriginSrc,
    [in] int          yoriginSrc,
    [in] int          wSrc,
    [in] int          hSrc,
    [in] BLENDFUNCTION ftn
);
```

Parameters

[in] `hdcDest`

A handle to the destination device context.

[in] `xoriginDest`

The x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] `yoriginDest`

The y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] `wDest`

The width, in logical units, of the destination rectangle.

[in] `hDest`

The height, in logical units, of the destination rectangle.

[in] `hdcSrc`

A handle to the source device context.

[in] `xoriginSrc`

The x-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] `yoriginSrc`

The y-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] `wSrc`

The width, in logical units, of the source rectangle.

[in] hSrc

The height, in logical units, of the source rectangle.

[in] ftn

The alpha-blending function for source and destination bitmaps, a global alpha value to be applied to the entire source bitmap, and format information for the source bitmap. The source and destination blend functions are currently limited to AC_SRC_OVER. See the [BLENDFUNCTION](#) and [EMRALPHABLEND](#) structures.

Return value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE.

Remarks

If the source rectangle and destination rectangle are not the same size, the source bitmap is stretched to match the destination rectangle. If the [SetStretchBltMode](#) function is used, the *iStretchMode* value is automatically converted to COLORONCOLOR for this function (that is, BLACKONWHITE, WHITEONBLACK, and HALFTONE are changed to COLORONCOLOR).

The destination coordinates are transformed by using the transformation currently specified for the destination device context. The source coordinates are transformed by using the transformation currently specified for the source device context.

An error occurs (and the function returns FALSE) if the source device context identifies an enhanced metafile device context.

If destination and source bitmaps do not have the same color format, **AlphaBlend** converts the source bitmap to match the destination bitmap.

AlphaBlend does not support mirroring. If either the width or height of the source or destination is negative, this call will fail.

When rendering to a printer, first call [GetDeviceCaps](#) with SHADEBLENDCAPS to determine if the printer supports blending with **AlphaBlend**. Note that, for a display DC, all blending operations are supported and these flags represent whether the operations are accelerated.

If the source and destination are the same surface, that is, they are both the screen or the same memory bitmap and the source and destination rectangles overlap, an error occurs and the function returns FALSE.

The source rectangle must lie completely within the source surface, otherwise an error occurs and the function returns FALSE.

AlphaBlend fails if the width or height of the source or destination is negative.

The **SourceConstantAlpha** member of [BLENDFUNCTION](#) specifies an alpha transparency value to be used on the entire source bitmap. The **SourceConstantAlpha** value is combined with any per-pixel alpha values. If **SourceConstantAlpha** is 0, it is assumed that the image is transparent. Set the **SourceConstantAlpha** value to 255 (which indicates that the image is opaque) when you only want to use per-pixel alpha values.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Msimg32.lib
DLL	Msimg32.dll

See also

[BLENDFUNCTION](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[EMRALPHABLEND](#)

[GetDeviceCaps](#)

[SetStretchBltMode](#)

AngleArc function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **AngleArc** function draws a line segment and an arc. The line segment is drawn from the current position to the beginning of the arc. The arc is drawn along the perimeter of a circle with the given radius and center. The length of the arc is defined by the given start and sweep angles.

Syntax

```
BOOL AngleArc(
    [in] HDC    hdc,
    [in] int     x,
    [in] int     y,
    [in] DWORD   r,
    [in] FLOAT   StartAngle,
    [in] FLOAT   SweepAngle
);
```

Parameters

[in] `hdc`

Handle to a device context.

[in] `x`

Specifies the x-coordinate, in logical units, of the center of the circle.

[in] `y`

Specifies the y-coordinate, in logical units, of the center of the circle.

[in] `r`

Specifies the radius, in logical units, of the circle. This value must be positive.

[in] `StartAngle`

Specifies the start angle, in degrees, relative to the x-axis.

[in] `SweepAngle`

Specifies the sweep angle, in degrees, relative to the starting angle.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **AngleArc** function moves the current position to the ending point of the arc.

The arc drawn by this function may appear to be elliptical, depending on the current transformation and

mapping mode. Before drawing the arc, **AngleArc** draws the line segment from the current position to the beginning of the arc.

The arc is drawn by constructing an imaginary circle around the specified center point with the specified radius. The starting point of the arc is determined by measuring counterclockwise from the x-axis of the circle by the number of degrees in the start angle. The ending point is similarly located by measuring counterclockwise from the starting point by the number of degrees in the sweep angle.

If the sweep angle is greater than 360 degrees, the arc is swept multiple times.

This function draws lines by using the current pen. The figure is not filled.

Examples

For an example, see [Drawing a Pie Chart](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Arc](#)

[ArcTo](#)

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

[MoveToEx](#)

AnimatePalette function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **AnimatePalette** function replaces entries in the specified logical palette.

Syntax

```
BOOL AnimatePalette(
    [in] HPALETTE      hPal,
    [in] UINT          iStartIndex,
    [in] UINT          cEntries,
    [in] const PALETTEENTRY *ppe
);
```

Parameters

[in] `hPal`

A handle to the logical palette.

[in] `iStartIndex`

The first logical palette entry to be replaced.

[in] `cEntries`

The number of entries to be replaced.

[in] `ppe`

A pointer to the first member in an array of **PALETTEENTRY** structures used to replace the current entries.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

An application can determine whether a device supports palette operations by calling the [GetDeviceCaps](#) function and specifying the RASTERCAPS constant.

The **AnimatePalette** function only changes entries with the PC_RESERVED flag set in the corresponding `palPalEntry` member of the [LOGPALETTE](#) structure.

If the given palette is associated with the active window, the colors in the palette are replaced immediately.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[CreatePalette](#)

[GetDeviceCaps](#)

[LOGPALETTE](#)

[PALETTEENTRY](#)

Arc function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **Arc** function draws an elliptical arc.

Syntax

```
BOOL Arc(
    [in] HDC hdc,
    [in] int x1,
    [in] int y1,
    [in] int x2,
    [in] int y2,
    [in] int x3,
    [in] int y3,
    [in] int x4,
    [in] int y4
);
```

Parameters

[in] `hdc`

A handle to the device context where drawing takes place.

[in] `x1`

The x-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

[in] `y1`

The y-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

[in] `x2`

The x-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

[in] `y2`

The y-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

[in] `x3`

The x-coordinate, in logical units, of the ending point of the radial line defining the starting point of the arc.

[in] `y3`

The y-coordinate, in logical units, of the ending point of the radial line defining the starting point of the arc.

[in] `x4`

The x-coordinate, in logical units, of the ending point of the radial line defining the ending point of the arc.

[in] `y4`

The y-coordinate, in logical units, of the ending point of the radial line defining the ending point of the arc.

Return value

If the arc is drawn, the return value is nonzero.

If the arc is not drawn, the return value is zero.

Remarks

The points (*nLeftRect*, *nTopRect*) and (*nRightRect*, *nBottomRect*) specify the bounding rectangle. An ellipse formed by the specified bounding rectangle defines the curve of the arc. The arc extends in the current drawing direction from the point where it intersects the radial from the center of the bounding rectangle to the (*nXStartArc*, *nYStartArc*) point. The arc ends where it intersects the radial from the center of the bounding rectangle to the (*nXEndArc*, *nYEndArc*) point. If the starting point and ending point are the same, a complete ellipse is drawn.

The arc is drawn using the current pen; it is not filled.

The current position is neither used nor updated by **Arc**.

Use the [GetArcDirection](#) and [SetArcDirection](#) functions to get and set the current drawing direction for a device context. The default drawing direction is counterclockwise.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AngleArc](#)

[ArcTo](#)

[Chord](#)

[Ellipse](#)

[GetArcDirection](#)

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

[Pie](#)

[SetArcDirection](#)

ArcTo function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ArcTo** function draws an elliptical arc.

Syntax

```
BOOL ArcTo(
    [in] HDC hdc,
    [in] int left,
    [in] int top,
    [in] int right,
    [in] int bottom,
    [in] int xr1,
    [in] int yr1,
    [in] int xr2,
    [in] int yr2
);
```

Parameters

[in] `hdc`

A handle to the device context where drawing takes place.

[in] `left`

The x-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

[in] `top`

The y-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

[in] `right`

The x-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

[in] `bottom`

The y-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

[in] `xr1`

The x-coordinate, in logical units, of the endpoint of the radial defining the starting point of the arc.

[in] `yr1`

The y-coordinate, in logical units, of the endpoint of the radial defining the starting point of the arc.

[in] `xr2`

The x-coordinate, in logical units, of the endpoint of the radial defining the ending point of the arc.

[in] `yr2`

The y-coordinate, in logical units, of the endpoint of the radial defining the ending point of the arc.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

[ArcTo](#) is similar to the [Arc](#) function, except that the current position is updated.

The points (*nLeftRect*, *nTopRect*) and (*nRightRect*, *nBottomRect*) specify the bounding rectangle. An ellipse formed by the specified bounding rectangle defines the curve of the arc. The arc extends counterclockwise from the point where it intersects the radial line from the center of the bounding rectangle to the (*nXRadial1*, *nYRadial1*) point. The arc ends where it intersects the radial line from the center of the bounding rectangle to the (*nXRadial2*, *nYRadial2*) point. If the starting point and ending point are the same, a complete ellipse is drawn.

A line is drawn from the current position to the starting point of the arc. If no error occurs, the current position is set to the ending point of the arc.

The arc is drawn using the current pen; it is not filled.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AngleArc](#)

[Arc](#)

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

[SetArcDirection](#)

AXESLISTA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The AXESLIST structure contains information on all the axes of a multiple master font.

Syntax

```
typedef struct tagAXESLISTA {
    DWORD     axlReserved;
    DWORD     axlNumAxes;
    AXISINFOA axlAxisInfo[MM_MAX_NUMAXES];
} AXESLISTA, *PAXESLISTA, *LPAXESLISTA;
```

Members

axlReserved

Reserved. Must be STAMP_AXESLIST.

axlNumAxes

Number of axes for a specified multiple master font.

axlAxisInfo

An array of [AXISINFO](#) structures. Each **AXISINFO** structure contains information on an axis of a specified multiple master font. This corresponds to the **dvValues** array in the [DESIGNVECTOR](#) structure.

Remarks

The PostScript Open Type Font does not support multiple master functionality.

The information on the axes of a multiple master font are specified by the [AXISINFO](#) structures. The **axlNumAxes** member specifies the actual size of **axlAxisInfo**, while **MM_MAX_NUMAXES**, which equals 16, is the maximum allowed size of **axlAxisInfo**.

NOTE

The wingdi.h header defines AXESLIST as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Header	wingdi.h (include Windows.h)
---------------	------------------------------

See also

[AXISINFO](#)

[DESIGNVECTOR](#)

[ENUMTEXTMETRIC](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

AXESLISTW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The AXESLIST structure contains information on all the axes of a multiple master font.

Syntax

```
typedef struct tagAXESLISTW {  
    DWORD     axlReserved;  
    DWORD     axlNumAxes;  
    AXISINFO  axlAxisInfo[MM_MAX_NUMAXES];  
} AXESLISTW, *PAXESLISTW, *LPAXESLISTW;
```

Members

axlReserved

Reserved. Must be STAMP_AXESLIST.

axlNumAxes

Number of axes for a specified multiple master font.

axlAxisInfo

An array of [AXISINFO](#) structures. Each **AXISINFO** structure contains information on an axis of a specified multiple master font. This corresponds to the **dvValues** array in the [DESIGNVECTOR](#) structure.

Remarks

The PostScript Open Type Font does not support multiple master functionality.

The information on the axes of a multiple master font are specified by the [AXISINFO](#) structures. The **axlNumAxes** member specifies the actual size of **axlAxisInfo**, while **MM_MAX_NUMAXES**, which equals 16, is the maximum allowed size of **axlAxisInfo**.

NOTE

The wingdi.h header defines AXESLIST as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Header	wingdi.h (include Windows.h)
---------------	------------------------------

See also

[AXISINFO](#)

[DESIGNVECTOR](#)

[ENUMTEXTMETRIC](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

AXISINFOA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The AXISINFO structure contains information about an axis of a multiple master font.

Syntax

```
typedef struct tagAXISINFOA {  
    LONG axMinValue;  
    LONG axMaxValue;  
    BYTE axAxisName[MM_MAX_AXES_NAMELEN];  
} AXISINFOA, *PAXISINFOA, *LPAxisInfoA;
```

Members

`axMinValue`

The minimum value for this axis.

`axMaxValue`

The maximum value for this axis.

`axAxisName`

The name of the axis, specified as an array of characters.

Remarks

The AXISINFO structure contains the name of an axis in a multiple master font and also the minimum and maximum possible values for the axis. The length of the name is MM_MAX_AXES_NAMELEN, which equals 16. An application queries these values before setting its desired values in the DESIGNVECTOR array.

The PostScript Open Type Font does not support multiple master functionality.

For the ANSI version of this structure, `axAxisName` must be an array of bytes.

NOTE

The wingdi.h header defines AXISINFO as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Header	wingdi.h (include Windows.h)
---------------	------------------------------

See also

[AXESLIST](#)

[DESIGNVECTOR](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

AXISINFO structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The AXISINFO structure contains information about an axis of a multiple master font.

Syntax

```
typedef struct tagAXISINFO {
    LONG axMinValue;
    LONG axMaxValue;
    WCHAR axAxisName[MM_MAX_AXES_NAMELEN];
} AXISINFO, *PAXISINFO, *LPAxisInfo;
```

Members

`axMinValue`

The minimum value for this axis.

`axMaxValue`

The maximum value for this axis.

`axAxisName`

The name of the axis, specified as an array of characters.

Remarks

The AXISINFO structure contains the name of an axis in a multiple master font and also the minimum and maximum possible values for the axis. The length of the name is MM_MAX_AXES_NAMELEN, which equals 16. An application queries these values before setting its desired values in the DESIGNVECTOR array.

The PostScript Open Type Font does not support multiple master functionality.

For the ANSI version of this structure, `axAxisName` must be an array of bytes.

NOTE

The wingdi.h header defines AXISINFO as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Header	wingdi.h (include Windows.h)
---------------	------------------------------

See also

[AXESLIST](#)

[DESIGNVECTOR](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

BeginPath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **BeginPath** function opens a path bracket in the specified device context.

Syntax

```
BOOL BeginPath(  
    [in] HDC hdc  
>;
```

Parameters

`[in] hdc`

A handle to the device context.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

After a path bracket is open, an application can begin calling GDI drawing functions to define the points that lie in the path. An application can close an open path bracket by calling the [EndPath](#) function.

When an application calls **BeginPath** for a device context, any previous paths are discarded from that device context. The following list shows which drawing functions can be used.

- [AngleArc](#)
- [Arc](#)
- [ArcTo](#)
- [Chord](#)
- [CloseFigure](#)
- [Ellipse](#)
- [ExtTextOut](#)
- [LineTo](#)
- [MoveToEx](#)
- [Pie](#)
- [PolyBezier](#)
- [PolyBezierTo](#)
- [PolyDraw](#)
- [Polygon](#)
- [Polyline](#)
- [PolylineTo](#)
- [PolyPolygon](#)

- [PolyPolyline](#)
- [Rectangle](#)
- [RoundRect](#)
- [TextOut](#)

Examples

For an example, see [Using Paths](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EndPath](#)

[FillPath](#)

[Path Functions](#)

[PathToRegion](#)

[Paths Overview](#)

[SelectClipPath](#)

[StrokeAndFillPath](#)

[StrokePath](#)

[WidenPath](#)

BitBlt function (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **BitBlt** function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

Syntax

```
BOOL BitBlt(
    [in] HDC    hdc,
    [in] int     x,
    [in] int     y,
    [in] int     cx,
    [in] int     cy,
    [in] HDC    hdcSrc,
    [in] int     x1,
    [in] int     y1,
    [in] DWORD   rop
);
```

Parameters

[in] hdc

A handle to the destination device context.

[in] x

The x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] y

The y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] cx

The width, in logical units, of the source and destination rectangles.

[in] cy

The height, in logical units, of the source and the destination rectangles.

[in] hdcSrc

A handle to the source device context.

[in] x1

The x-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] y1

The y-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] rop

A raster-operation code. These codes define how the color data for the source rectangle is to be combined with

the color data for the destination rectangle to achieve the final color.

The following list shows some common raster operation codes.

VALUE	MEANING
BLACKNESS	Fills the destination rectangle using the color associated with index 0 in the physical palette. (This color is black for the default physical palette.)
CAPTUREBLT	Includes any windows that are layered on top of your window in the resulting image. By default, the image only contains your window. Note that this generally cannot be used for printing device contexts.
DSTINVERT	Inverts the destination rectangle.
MERGECOPY	Merges the colors of the source rectangle with the brush currently selected in <i>hdcDest</i> , by using the Boolean AND operator.
MERGEPAINT	Merges the colors of the inverted source rectangle with the colors of the destination rectangle by using the Boolean OR operator.
NOMIRRORBITMAP	Prevents the bitmap from being mirrored.
NOTSRCCOPY	Copies the inverted source rectangle to the destination.
NOTSRCERASE	Combines the colors of the source and destination rectangles by using the Boolean OR operator and then inverts the resultant color.
PATCOPY	Copies the brush currently selected in <i>hdcDest</i> , into the destination bitmap.
PATINVERT	Combines the colors of the brush currently selected in <i>hdcDest</i> , with the colors of the destination rectangle by using the Boolean XOR operator.
PATPAINT	Combines the colors of the brush currently selected in <i>hdcDest</i> , with the colors of the inverted source rectangle by using the Boolean OR operator. The result of this operation is combined with the colors of the destination rectangle by using the Boolean OR operator.
SRCAND	Combines the colors of the source and destination rectangles by using the Boolean AND operator.
SRCCOPY	Copies the source rectangle directly to the destination rectangle.

SRCERASE	Combines the inverted colors of the destination rectangle with the colors of the source rectangle by using the Boolean AND operator.
SRCINVERT	Combines the colors of the source and destination rectangles by using the Boolean XOR operator.
SRCPAINT	Combines the colors of the source and destination rectangles by using the Boolean OR operator.
WHITENESS	Fills the destination rectangle using the color associated with index 1 in the physical palette. (This color is white for the default physical palette.)

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

Remarks

BitBlt only does clipping on the destination DC.

If a rotation or shear transformation is in effect in the source device context, **BitBlt** returns an error. If other transformations exist in the source device context (and a matching transformation is not in effect in the destination device context), the rectangle in the destination device context is stretched, compressed, or rotated, as necessary.

If the color formats of the source and destination device contexts do not match, the **BitBlt** function converts the source color format to match the destination format.

When an enhanced metafile is being recorded, an error occurs if the source device context identifies an enhanced-metafile device context.

Not all devices support the **BitBlt** function. For more information, see the RC_BITBLT raster capability entry in the [GetDeviceCaps](#) function as well as the following functions: [MaskBlt](#), [PngBlt](#), and [StretchBlt](#).

BitBlt returns an error if the source and destination device contexts represent different devices. To transfer data between DCs for different devices, convert the memory bitmap to a DIB by calling [GetDIBits](#). To display the DIB to the second device, call [SetDIBits](#) or [StretchDIBits](#).

ICM: No color management is performed when blits occur.

Examples

The following code example demonstrates the use of **BitBlt**.

```

if (!BitBlt(hdcMemDC,
    0, 0,
    rcClient.right - rcClient.left, rcClient.bottom - rcClient.top,
    hdcWindow,
    0, 0,
    SRCCOPY))
{
    MessageBox(hWnd, L"BitBlt has failed", L"Failed", MB_OK);
    goto done;
}

```

To see this example in context, see [Capturing an Image](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[GetDIBits](#)

[GetDeviceCaps](#)

[MaskBlt](#)

[PlgBlt](#)

[SetDIBits](#)

[StretchBlt](#)

[StretchDIBits](#)

BITMAP structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **BITMAP** structure defines the type, width, height, color format, and bit values of a bitmap.

Syntax

```
typedef struct tagBITMAP {  
    LONG    bmType;  
    LONG    bmWidth;  
    LONG    bmHeight;  
    LONG    bmWidthBytes;  
    WORD    bmPlanes;  
    WORD    bmBitsPixel;  
    LPVOID  bmBits;  
} BITMAP, *PBITMAP, *NPBITMAP, *LPBITMAP;
```

Members

bmType

The bitmap type. This member must be zero.

bmWidth

The width, in pixels, of the bitmap. The width must be greater than zero.

bmHeight

The height, in pixels, of the bitmap. The height must be greater than zero.

bmWidthBytes

The number of bytes in each scan line. This value must be divisible by 2, because the system assumes that the bit values of a bitmap form an array that is word aligned.

bmPlanes

The count of color planes.

bmBitsPixel

The number of bits required to indicate the color of a pixel.

bmBits

A pointer to the location of the bit values for the bitmap. The **bmBits** member must be a pointer to an array of character (1-byte) values.

Remarks

The bitmap formats currently used are monochrome and color. The monochrome bitmap uses a one-bit, one-plane format. Each scan is a multiple of 16 bits.

Scans are organized as follows for a monochrome bitmap of height n :

```
Scan 0
Scan 1
.
.
.
Scan n-2
Scan n-1
```

The pixels on a monochrome device are either black or white. If the corresponding bit in the bitmap is 1, the pixel is set to the foreground color; if the corresponding bit in the bitmap is zero, the pixel is set to the background color.

All devices that have the RC_BITBLT device capability support bitmaps. For more information, see [GetDeviceCaps](#).

Each device has a unique color format. To transfer a bitmap from one device to another, use the [GetDIBits](#) and [SetDIBits](#) functions.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Bitmap Structures](#)

[Bitmaps Overview](#)

[CreateBitmapIndirect](#)

[GetDIBits](#)

[GetDeviceCaps](#)

[GetObject](#)

[SetDIBits](#)

BITMAPCOREHEADER structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **BITMAPCOREHEADER** structure contains information about the dimensions and color format of a DIB.

Syntax

```
typedef struct tagBITMAPCOREHEADER {  
    DWORD bcSize;  
    WORD  bcWidth;  
    WORD  bcHeight;  
    WORD  bcPlanes;  
    WORD  bcBitCount;  
} BITMAPCOREHEADER, *LPBITMAPCOREHEADER, *PBITMAPCOREHEADER;
```

Members

bcSize

The number of bytes required by the structure.

bcWidth

The width of the bitmap, in pixels.

bcHeight

The height of the bitmap, in pixels.

bcPlanes

The number of planes for the target device. This value must be 1.

bcBitCount

The number of bits-per-pixel. This value must be 1, 4, 8, or 24.

Remarks

The [BITMAPCOREINFO](#) structure combines the **BITMAPCOREHEADER** structure and a color table to provide a complete definition of the dimensions and colors of a DIB. For more information about specifying a DIB, see [BITMAPCOREINFO](#).

An application should use the information stored in the **bcSize** member to locate the color table in a [BITMAPCOREINFO](#) structure, using a method such as the following:

```
pColor = ((LPBYTE) pBitmapCoreInfo +  
          (WORD) (pBitmapCoreInfo -> bcSize))
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPCOREINFO](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

BITMAPCOREINFO structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **BITMAPCOREINFO** structure defines the dimensions and color information for a DIB.

Syntax

```
typedef struct tagBITMAPCOREINFO {
    BITMAPCOREHEADER bmciHeader;
    RGBTRIPLE        bmciColors[1];
} BITMAPCOREINFO, *LPBITMAPCOREINFO, *PBITMAPCOREINFO;
```

Members

bmciHeader

A **BITMAPCOREHEADER** structure that contains information about the dimensions and color format of a DIB.

bmciColors

Specifies an array of **RGBTRIPLE** structures that define the colors in the bitmap.

Remarks

A DIB consists of two parts: a **BITMAPCOREINFO** structure describing the dimensions and colors of the bitmap, and an array of bytes defining the pixels of the bitmap. The bits in the array are packed together, but each scan line must be padded with zeros to end on a **LONG** boundary. The origin of the bitmap is the lower-left corner.

The **bcBitCount** member of the **BITMAPCOREHEADER** structure determines the number of bits that define each pixel and the maximum number of colors in the bitmap. This member can be one of the following values.

VALUE	MEANING
1	The bitmap is monochrome, and the bmciColors member contains two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the bmciColors table; if the bit is set, the pixel has the color of the second entry in the table.
4	The bitmap has a maximum of 16 colors, and the bmciColors member contains up to 16 entries. Each pixel in the bitmap is represented by a 4-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the sixteenth table entry.
8	The bitmap has a maximum of 256 colors, and the bmciColors member contains up to 256 entries. In this case, each byte in the array represents a single pixel.

The bitmap has a maximum of 2 (24) colors, and the **bmciColors** member is **NULL**. Each three-byte triplet in the bitmap array represents the relative intensities of blue, green, and red, respectively, for a pixel.

The colors in the **bmciColors** table should appear in order of importance.

Alternatively, for functions that use DIBs, the **bmciColors** member can be an array of 16-bit unsigned integers that specify indexes into the currently realized logical palette, instead of explicit RGB values. In this case, an application using the bitmap must call the DIB functions ([CreateDIBitmap](#), [CreateDIBPatternBrush](#), and [CreateDIBSection](#)) with the *iUsage* parameter set to **DIB_PAL_COLORS**.

Note

The **bmciColors** member should not contain palette indexes if the bitmap is to be stored in a file or transferred to another application. Unless the application has exclusive use and control of the bitmap, the bitmap color table should contain explicit RGB values.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPCOREHEADER](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

[CreateDIBPatternBrush](#)

[CreateDIBSection](#)

[CreateDIBitmap](#)

[RGBTRIPLE](#)

BITMAPFILEHEADER structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **BITMAPFILEHEADER** structure contains information about the type, size, and layout of a file that contains a DIB.

Syntax

```
typedef struct tagBITMAPFILEHEADER {  
    WORD  bfType;  
    DWORD bfSize;  
    WORD  bfReserved1;  
    WORD  bfReserved2;  
    DWORD bfOffBits;  
} BITMAPFILEHEADER, *LPBITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

Members

bfType

The file type; must be BM.

bfSize

The size, in bytes, of the bitmap file.

bfReserved1

Reserved; must be zero.

bfReserved2

Reserved; must be zero.

bfOffBits

The offset, in bytes, from the beginning of the **BITMAPFILEHEADER** structure to the bitmap bits.

Remarks

A [BITMAPINFO](#) or [BITMAPCOREINFO](#) structure immediately follows the **BITMAPFILEHEADER** structure in the DIB file. For more information, see [Bitmap Storage](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPCOREINFO](#)

[BITMAPINFO](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

BITMAPINFO structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **BITMAPINFO** structure defines the dimensions and color information for a DIB.

Syntax

```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD        bmiColors[1];
} BITMAPINFO, *LPBITMAPINFO, *PBITMAPINFO;
```

Members

bmiHeader

A [BITMAPINFOHEADER](#) structure that contains information about the dimensions of color format.

bmiColors

The **bmiColors** member contains one of the following:

- An array of [RGBQUAD](#). The elements of the array that make up the color table.
- An array of 16-bit unsigned integers that specifies indexes into the currently realized logical palette. This use of **bmiColors** is allowed for functions that use DIBs. When **bmiColors** elements contain indexes to a realized logical palette, they must also call the following bitmap functions: [CreateDIBitmap](#), [CreateDIBPatternBrush](#)

CreateDIBSection

The *iUsage* parameter of [CreateDIBSection](#) must be set to **DIB_PAL_COLORS**.

The number of entries in the array depends on the values of the **biBitCount** and **biClrUsed** members of the [BITMAPINFOHEADER](#) structure.

The colors in the **bmiColors** table appear in order of importance. For more information, see the Remarks section.

Remarks

A DIB consists of two distinct parts: a **BITMAPINFO** structure describing the dimensions and colors of the bitmap, and an array of bytes defining the pixels of the bitmap. The bits in the array are packed together, but each scan line must be padded with zeros to end on a **LONG** data-type boundary. If the height of the bitmap is positive, the bitmap is a bottom-up DIB and its origin is the lower-left corner. If the height is negative, the bitmap is a top-down DIB and its origin is the upper left corner.

A bitmap is packed when the bitmap array immediately follows the **BITMAPINFO** header. Packed bitmaps are referenced by a single pointer. For packed bitmaps, the **biClrUsed** member must be set to an even number when using the **DIB_PAL_COLORS** mode so that the DIB bitmap array starts on a **DWORD** boundary.

Note

The **bmiColors** member should not contain palette indexes if the bitmap is to be stored in a file or transferred to another application.

Unless the application has exclusive use and control of the bitmap, the bitmap color table should contain explicit RGB values.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFOHEADER](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

[CreateDIBPatternBrush](#)

[CreateDIBSection](#)

[CreateDIBitmap](#)

[RGBQUAD](#)

BITMAPV4HEADER structure (wingdi.h)

4/21/2022 • 7 minutes to read • [Edit Online](#)

The **BITMAPV4HEADER** structure is the bitmap information header file. It is an extended version of the **BITMAPINFOHEADER** structure.

Applications can use the **BITMAPV5HEADER** structure for added functionality.

Syntax

```
typedef struct {
    DWORD      bV4Size;
    LONG       bV4Width;
    LONG       bV4Height;
    WORD       bV4Planes;
    WORD       bV4BitCount;
    DWORD      bV4V4Compression;
    DWORD      bV4SizeImage;
    LONG       bV4XPelsPerMeter;
    LONG       bV4YPelsPerMeter;
    DWORD      bV4ClrUsed;
    DWORD      bV4ClrImportant;
    DWORD      bV4RedMask;
    DWORD      bV4GreenMask;
    DWORD      bV4BlueMask;
    DWORD      bV4AlphaMask;
    DWORD      bV4CSType;
    CIEXYZTRIPLE bV4Endpoints;
    DWORD      bV4GammaRed;
    DWORD      bV4GammaGreen;
    DWORD      bV4GammaBlue;
} BITMAPV4HEADER, *LPBITMAPV4HEADER, *PBITMAPV4HEADER;
```

Members

bV4Size

The number of bytes required by the structure. Applications should use this member to determine which bitmap information header structure is being used.

bV4Width

The width of the bitmap, in pixels.

If **bV4Compression** is **BI_JPEG** or **BI_PNG**, **bV4Width** specifies the width of the JPEG or PNG image in pixels.

bV4Height

The height of the bitmap, in pixels. If **bV4Height** is positive, the bitmap is a bottom-up DIB and its origin is the lower-left corner. If **bV4Height** is negative, the bitmap is a top-down DIB and its origin is the upper-left corner.

If **bV4Height** is negative, indicating a top-down DIB, **bV4Compression** must be either **BI_RGB** or **BI_BITFIELDS**. Top-down DIBs cannot be compressed.

If **bV4Compression** is **BI_JPEG** or **BI_PNG**, **bV4Height** specifies the height of the JPEG or PNG image in pixels.

bV4Planes

The number of planes for the target device. This value must be set to 1.

bV4BitCount

The number of bits-per-pixel. The **bV4BitCount** member of the **BITMAPV4HEADER** structure determines the number of bits that define each pixel and the maximum number of colors in the bitmap. This member must be one of the following values.

VALUE	MEANING
0	The number of bits-per-pixel is specified or is implied by the JPEG or PNG file format.
1	The bitmap is monochrome, and the bmiColors member of BITMAPINFO contains two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the bmiColors table; if the bit is set, the pixel has the color of the second entry in the table.
4	The bitmap has a maximum of 16 colors, and the bmiColors member of BITMAPINFO contains up to 16 entries. Each pixel in the bitmap is represented by a 4-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the sixteenth table entry.
8	The bitmap has a maximum of 256 colors, and the bmiColors member of BITMAPINFO contains up to 256 entries. In this case, each byte in the array represents a single pixel.
16	The bitmap has a maximum of 2^{16} colors. If the bV4Compression member of the BITMAPV4HEADER structure is BI_RGB , the bmiColors member of BITMAPINFO is NULL . Each WORD in the bitmap array represents a single pixel. The relative intensities of red, green, and blue are represented with five bits for each color component. The value for blue is in the least significant five bits, followed by five bits each for green and red, respectively. The most significant bit is not used. The bmiColors color table is used for optimizing colors used on palette-based devices, and must contain the number of entries specified by the bV4ClrUsed member of the BITMAPV4HEADER . If the bV4Compression member of the BITMAPV4HEADER is BI_BITFIELDS , the bmiColors member contains three DWORD color masks that specify the red, green, and blue components of each pixel. Each WORD in the bitmap array represents a single pixel.
24	The bitmap has a maximum of 2^{24} colors, and the bmiColors member of BITMAPINFO is NULL . Each 3-byte triplet in the bitmap array represents the relative intensities of blue, green, and red for a pixel. The bmiColors color table is used for optimizing colors used on palette-based devices, and must contain the number of entries specified by the bV4ClrUsed member of the BITMAPV4HEADER .

The bitmap has a maximum of 2^{32} colors. If the **bV4Compression** member of the **BITMAPV4HEADER** is **BI_RGB**, the **bmiColors** member of **BITMAPINFO** is **NULL**. Each **DWORD** in the bitmap array represents the relative intensities of blue, green, and red for a pixel. The value for blue is in the least significant 8 bits, followed by 8 bits each for green and red. The high byte in each **DWORD** is not used. The **bmiColors** color table is used for optimizing colors used on palette-based devices, and must contain the number of entries specified by the **bV4ClrUsed** member of the **BITMAPV4HEADER**. If the **bV4Compression** member of the **BITMAPV4HEADER** is **BI_BITFIELDS**, the **bmiColors** member contains three **DWORD** color masks that specify the red, green, and blue components of each pixel. Each **DWORD** in the bitmap array represents a single pixel.

bV4V4Compression

The type of compression for a compressed bottom-up bitmap (top-down DIBs cannot be compressed). This member can be one of the following values.

VALUE	DESCRIPTION
BI_RGB	An uncompressed format.
BI_RLE8	A run-length encoded (RLE) format for bitmaps with 8 bpp. The compression format is a 2-byte format consisting of a count byte followed by a byte containing a color index. For more information, see Bitmap Compression .
BI_RLE4	An RLE format for bitmaps with 4 bpp. The compression format is a 2-byte format consisting of a count byte followed by two word-length color indexes. For more information, see Bitmap Compression .
BI_BITFIELDS	Specifies that the bitmap is not compressed. The members bV4RedMask , bV4GreenMask , and bV4BlueMask specify the red, green, and blue components for each pixel. This is valid when used with 16- and 32-bpp bitmaps.
BI_JPEG	Specifies that the image is compressed using the JPEG file interchange format. JPEG compression trades off compression against loss; it can achieve a compression ratio of 20:1 with little noticeable loss.
BI_PNG	Specifies that the image is compressed using the PNG file interchange format.

bV4SizeImage

The size, in bytes, of the image. This may be set to zero for **BI_RGB** bitmaps.

If **bV4Compression** is **BI_JPEG** or **BI_PNG**, **bV4SizeImage** is the size of the JPEG or PNG image buffer.

bV4XPelsPerMeter

The horizontal resolution, in pixels-per-meter, of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device.

bV4YPelsPerMeter

The vertical resolution, in pixels-per-meter, of the target device for the bitmap.

bV4ClrUsed

The number of color indexes in the color table that are actually used by the bitmap. If this value is zero, the bitmap uses the maximum number of colors corresponding to the value of the **bV4BitCount** member for the compression mode specified by **bV4Compression**.

If **bV4ClrUsed** is nonzero and the **bV4BitCount** member is less than 16, the **bV4ClrUsed** member specifies the actual number of colors the graphics engine or device driver accesses. If **bV4BitCount** is 16 or greater, the **bV4ClrUsed** member specifies the size of the color table used to optimize performance of the system color palettes. If **bV4BitCount** equals 16 or 32, the optimal color palette starts immediately following the **BITMAPV4HEADER**.

bV4ClrImportant

The number of color indexes that are required for displaying the bitmap. If this value is zero, all colors are important.

bV4RedMask

Color mask that specifies the red component of each pixel, valid only if **bV4Compression** is set to **BI_BITFIELDS**.

bV4GreenMask

Color mask that specifies the green component of each pixel, valid only if **bV4Compression** is set to **BI_BITFIELDS**.

bV4BlueMask

Color mask that specifies the blue component of each pixel, valid only if **bV4Compression** is set to **BI_BITFIELDS**.

bV4AlphaMask

Color mask that specifies the alpha component of each pixel.

bV4CSType

The color space of the DIB. The following table lists the value for **bV4CSType**.

VALUE	MEANING
LCS_CALIBRATED_RGB	This value indicates that endpoints and gamma values are given in the appropriate fields.

See the [LOGCOLORSPACE](#) structure for information that defines a logical color space.

bV4Endpoints

A [CIEXYZTRIPLE](#) structure that specifies the x, y, and z coordinates of the three colors that correspond to the red, green, and blue endpoints for the logical color space associated with the bitmap. This member is ignored unless the **bV4CSType** member specifies **LCS_CALIBRATED_RGB**.

Note A *color space* is a model for representing color numerically in terms of three or more coordinates. For example, the RGB color space represents colors in terms of the red, green, and blue coordinates.

bV4GammaRed

Tone response curve for red. This member is ignored unless color values are calibrated RGB values and **bV4CSType** is set to **LCS_CALIBRATED_RGB**. Specify in unsigned fixed 16.16 format. The upper 16 bits are the unsigned integer value. The lower 16 bits are the fractional part.

bV4GammaGreen

Tone response curve for green. Used if **bV4CSType** is set to **LCS_CALIBRATED_RGB**. Specify in unsigned fixed 16.16 format. The upper 16 bits are the unsigned integer value. The lower 16 bits are the fractional part.

bV4GammaBlue

Tone response curve for blue. Used if **bV4CSType** is set to **LCS_CALIBRATED_RGB**. Specify in unsigned fixed 16.16 format. The upper 16 bits are the unsigned integer value. The lower 16 bits are the fractional part.

Remarks

The **BITMAPV4HEADER** structure is extended to allow a JPEG or PNG image to be passed as the source image to [StretchDIBits](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[BITMAPINFOHEADER](#)

[BITMAPV5HEADER](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

[CIEXYZTRIPLE](#)

[CreateDIBitmap](#)

[LOGCOLORSPACE](#)

[StretchDIBits](#)

BITMAPV5HEADER structure (wingdi.h)

4/21/2022 • 10 minutes to read • [Edit Online](#)

The **BITMAPV5HEADER** structure is the bitmap information header file. It is an extended version of the [BITMAPINFOHEADER](#) structure.

Syntax

```
typedef struct {
    DWORD      bV5Size;
    LONG       bV5Width;
    LONG       bV5Height;
    WORD       bV5Planes;
    WORD       bV5BitCount;
    DWORD      bV5Compression;
    DWORD      bV5SizeImage;
    LONG       bV5XPelsPerMeter;
    LONG       bV5YPelsPerMeter;
    DWORD      bV5ClrUsed;
    DWORD      bV5ClrImportant;
    DWORD      bV5RedMask;
    DWORD      bV5GreenMask;
    DWORD      bV5BlueMask;
    DWORD      bV5AlphaMask;
    DWORD      bV5CSType;
    CIEXYZTRIPLE bV5Endpoints;
    DWORD      bV5GammaRed;
    DWORD      bV5GammaGreen;
    DWORD      bV5GammaBlue;
    DWORD      bV5Intent;
    DWORD      bV5ProfileData;
    DWORD      bV5ProfileSize;
    DWORD      bV5Reserved;
} BITMAPV5HEADER, *LPBITMAPV5HEADER, *PBITMAPV5HEADER;
```

Members

bV5Size

The number of bytes required by the structure. Applications should use this member to determine which bitmap information header structure is being used.

bV5Width

The width of the bitmap, in pixels.

If **bV5Compression** is **BI_JPEG** or **BI_PNG**, the **bV5Width** member specifies the width of the decompressed JPEG or PNG image in pixels.

bV5Height

The height of the bitmap, in pixels. If the value of **bV5Height** is positive, the bitmap is a bottom-up DIB and its origin is the lower-left corner. If **bV5Height** value is negative, the bitmap is a top-down DIB and its origin is the upper-left corner.

If **bV5Height** is negative, indicating a top-down DIB, **bV5Compression** must be either **BI_RGB** or **BI_BITFIELDS**.

Top-down DIBs cannot be compressed.

If **bV5Compression** is **BI_JPEG** or **BI_PNG**, the **bV5Height** member specifies the height of the decompressed JPEG or PNG image in pixels.

bV5Planes

The number of planes for the target device. This value must be set to 1.

bV5BitCount

The number of bits that define each pixel and the maximum number of colors in the bitmap.

This member can be one of the following values.

VALUE	MEANING
0	The number of bits per pixel is specified or is implied by the JPEG or PNG file format.
1	The bitmap is monochrome, and the bmiColors member of BITMAPINFO contains two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the bmiColors color table. If the bit is set, the pixel has the color of the second entry in the table.
4	The bitmap has a maximum of 16 colors, and the bmiColors member of BITMAPINFO contains up to 16 entries. Each pixel in the bitmap is represented by a 4-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the sixteenth table entry.
8	The bitmap has a maximum of 256 colors, and the bmiColors member of BITMAPINFO contains up to 256 entries. In this case, each byte in the array represents a single pixel.
16	<p>The bitmap has a maximum of 2^{16} colors. If the bV5Compression member of the BITMAPV5HEADER structure is BI_RGB, the bmiColors member of BITMAPINFO is NULL. Each WORD in the bitmap array represents a single pixel. The relative intensities of red, green, and blue are represented with five bits for each color component. The value for blue is in the least significant five bits, followed by five bits each for green and red. The most significant bit is not used. The bmiColors color table is used for optimizing colors used on palette-based devices, and must contain the number of entries specified by the bV5ClrUsed member of the BITMAPV5HEADER. If the bV5Compression member of the BITMAPV5HEADER is BI_BITFIELDS, the bmiColors member contains three DWORD color masks that specify the red, green, and blue components, respectively, of each pixel. Each WORD in the bitmap array represents a single pixel.</p> <p>When the bV5Compression member is BI_BITFIELDS, bits set in each DWORD mask must be contiguous and should not overlap the bits of another mask. All the bits in the pixel do not need to be used.</p>

24	The bitmap has a maximum of 2^{24} colors, and the bmiColors member of BITMAPINFO is NULL . Each 3-byte triplet in the bitmap array represents the relative intensities of blue, green, and red, respectively, for a pixel. The bmiColors color table is used for optimizing colors used on palette-based devices, and must contain the number of entries specified by the bV5ClrUsed member of the BITMAPV5HEADER structure.
32	The bitmap has a maximum of 2^{32} colors. If the bV5Compression member of the BITMAPV5HEADER is BI_RGB , the bmiColors member of BITMAPINFO is NULL . Each DWORD in the bitmap array represents the relative intensities of blue, green, and red for a pixel. The value for blue is in the least significant 8 bits, followed by 8 bits each for green and red. The high byte in each DWORD is not used. The bmiColors color table is used for optimizing colors used on palette-based devices, and must contain the number of entries specified by the bV5ClrUsed member of the BITMAPV5HEADER . If the bV5Compression member of the BITMAPV5HEADER is BI_BITFIELDS , the bmiColors member contains three DWORD color masks that specify the red, green, and blue components of each pixel. Each DWORD in the bitmap array represents a single pixel.

bV5Compression

Specifies that the bitmap is not compressed. The **bV5RedMask**, **bV5GreenMask**, and **bV5BlueMask** members specify the red, green, and blue components of each pixel. This is valid when used with 16- and 32-bpp bitmaps. This member can be one of the following values.

VALUE	MEANING
BI_RGB	An uncompressed format.
BI_RLE8	A run-length encoded (RLE) format for bitmaps with 8 bpp. The compression format is a two-byte format consisting of a count byte followed by a byte containing a color index. If bV5Compression is BI_RGB and the bV5BitCount member is 16, 24, or 32, the bitmap array specifies the actual intensities of blue, green, and red rather than using color table indexes. For more information, see Bitmap Compression .
BI_RLE4	An RLE format for bitmaps with 4 bpp. The compression format is a two-byte format consisting of a count byte followed by two word-length color indexes. For more information, see Bitmap Compression .
BI_BITFIELDS	Specifies that the bitmap is not compressed and that the color masks for the red, green, and blue components of each pixel are specified in the bV5RedMask , bV5GreenMask , and bV5BlueMask members. This is valid when used with 16- and 32-bpp bitmaps.
BI_JPEG	Specifies that the image is compressed using the JPEG file Interchange Format. JPEG compression trades off compression against loss; it can achieve a compression ratio of 20:1 with little noticeable loss.

BI_PNG	Specifies that the image is compressed using the PNG file Interchange Format.
---------------	---

bV5SizeImage

The size, in bytes, of the image. This may be set to zero for BI_RGB bitmaps.

If **bV5Compression** is BI_JPEG or BI_PNG, **bV5SizeImage** is the size of the JPEG or PNG image buffer.

bV5XPelsPerMeter

The horizontal resolution, in pixels-per-meter, of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device.

bV5YPelsPerMeter

The vertical resolution, in pixels-per-meter, of the target device for the bitmap.

bV5ClrUsed

The number of color indexes in the color table that are actually used by the bitmap. If this value is zero, the bitmap uses the maximum number of colors corresponding to the value of the **bV5BitCount** member for the compression mode specified by **bV5Compression**.

If **bV5ClrUsed** is nonzero and **bV5BitCount** is less than 16, the **bV5ClrUsed** member specifies the actual number of colors the graphics engine or device driver accesses. If **bV5BitCount** is 16 or greater, the **bV5ClrUsed** member specifies the size of the color table used to optimize performance of the system color palettes. If **bV5BitCount** equals 16 or 32, the optimal color palette starts immediately following the **BITMAPV5HEADER**. If **bV5ClrUsed** is nonzero, the color table is used on palettized devices, and **bV5ClrUsed** specifies the number of entries.

bV5ClrImportant

The number of color indexes that are required for displaying the bitmap. If this value is zero, all colors are required.

bV5RedMask

Color mask that specifies the red component of each pixel, valid only if **bV5Compression** is set to BI_BITFIELDS.

bV5GreenMask

Color mask that specifies the green component of each pixel, valid only if **bV5Compression** is set to BI_BITFIELDS.

bV5BlueMask

Color mask that specifies the blue component of each pixel, valid only if **bV5Compression** is set to BI_BITFIELDS.

bV5AlphaMask

Color mask that specifies the alpha component of each pixel.

bV5CSType

The color space of the DIB.

The following table specifies the values for **bV5CSType**.

VALUE	MEANING
LCS_CALIBRATED_RGB	This value implies that endpoints and gamma values are given in the appropriate fields.
LCS_sRGB	Specifies that the bitmap is in sRGB color space.
LCS_WINDOWS_COLOR_SPACE	This value indicates that the bitmap is in the system default color space, sRGB.
PROFILE_LINKED	This value indicates that bV5ProfileData points to the file name of the profile to use (gamma and endpoints values are ignored).
PROFILE_EMBEDDED	This value indicates that bV5ProfileData points to a memory buffer that contains the profile to be used (gamma and endpoints values are ignored).

See the [LOGCOLORSPACE](#) structure for information that defines a logical color space.

bV5Endpoints

A [CIEXYZTRIPLE](#) structure that specifies the x, y, and z coordinates of the three colors that correspond to the red, green, and blue endpoints for the logical color space associated with the bitmap. This member is ignored unless the **bV5CSType** member specifies LCS_CALIBRATED_RGB.

bV5GammaRed

Toned response curve for red. Used if **bV5CSType** is set to LCS_CALIBRATED_RGB. Specify in unsigned fixed 16.16 format. The upper 16 bits are the unsigned integer value. The lower 16 bits are the fractional part.

bV5GammaGreen

Toned response curve for green. Used if **bV5CSType** is set to LCS_CALIBRATED_RGB. Specify in unsigned fixed 16.16 format. The upper 16 bits are the unsigned integer value. The lower 16 bits are the fractional part.

bV5GammaBlue

Toned response curve for blue. Used if **bV5CSType** is set to LCS_CALIBRATED_RGB. Specify in unsigned fixed 16.16 format. The upper 16 bits are the unsigned integer value. The lower 16 bits are the fractional part.

bV5Intent

Rendering intent for bitmap. This can be one of the following values.

VALUE	INTENT	ICC NAME	MEANING
LCS_GM_ABS_COLORIMETRIC	Match	Absolute Colorimetric	Maintains the white point. Matches the colors to their nearest color in the destination gamut.
LCS_GM_BUSINESS	Graphic	Saturation	Maintains saturation. Used for business charts and other situations in which undithered colors are required.

LCS_GM_GRAPHICS	Proof	Relative Colorimetric	Maintains colorimetric match. Used for graphic designs and named colors.
LCS_GM_IMAGES	Picture	Perceptual	Maintains contrast. Used for photographs and natural images.

bV5ProfileData

The offset, in bytes, from the beginning of the **BITMAPV5HEADER** structure to the start of the profile data. If the profile is embedded, profile data is the actual profile, and it is linked. (The profile data is the null-terminated file name of the profile.) This cannot be a Unicode string. It must be composed exclusively of characters from the Windows character set (code page 1252). These profile members are ignored unless the **bV5CSType** member specifies PROFILE_LINKED or PROFILE_EMBEDDED.

bV5ProfileSize

Size, in bytes, of embedded profile data.

bV5Reserved

This member has been reserved. Its value should be set to zero.

Remarks

If **bV5Height** is negative, indicating a top-down DIB, **bV5Compression** must be either BI_RGB or BI_BITFIELDS. Top-down DIBs cannot be compressed.

The Independent Color Management interface (ICM) 2.0 allows International Color Consortium (ICC) color profiles to be linked or embedded in DIBs (DIBs). See [Using Structures](#) for more information.

When a DIB is loaded into memory, the profile data (if present) should follow the color table, and the **bV5ProfileData** should provide the offset of the profile data from the beginning of the **BITMAPV5HEADER** structure. The value stored in **bV5ProfileData** will be different from the value returned by the **sizeof** operator given the **BITMAPV5HEADER** argument, because **bV5ProfileData** is the offset in bytes from the beginning of the **BITMAPV5HEADER** structure to the start of the profile data. (Bitmap bits do not follow the color table in memory). Applications should modify the **bV5ProfileData** member after loading the DIB into memory.

For packed DIBs, the profile data should follow the bitmap bits similar to the file format. The **bV5ProfileData** member should still give the offset of the profile data from the beginning of the **BITMAPV5HEADER**.

Applications should access the profile data only when **bV5Size** equals the size of the **BITMAPV5HEADER** and **bV5CSType** equals PROFILE_EMBEDDED or PROFILE_LINKED.

If a profile is linked, the path of the profile can be any fully qualified name (including a network path) that can be opened using the [CreateFile](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Header	wingdi.h (include Windows.h)
---------------	------------------------------

See also

[BITMAPINFO](#)

[BITMAPINFOHEADER](#)

[BITMAPV4HEADER](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

[CIEXYZTRIPLE](#)

[CreateDIBitmap](#)

[LOGCOLORSPACE](#)

[StretchDIBits](#)

BLENDFUNCTION structure (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **BLENDFUNCTION** structure controls blending by specifying the blending functions for source and destination bitmaps.

Syntax

```
typedef struct _BLENDFUNCTION {
    BYTE BlendOp;
    BYTE BlendFlags;
    BYTE SourceConstantAlpha;
    BYTE AlphaFormat;
} BLENDFUNCTION, *PBLENDFUNCTION;
```

Members

BlendOp

The source blend operation. Currently, the only source and destination blend operation that has been defined is AC_SRC_OVER. For details, see the following Remarks section.

BlendFlags

Must be zero.

SourceConstantAlpha

Specifies an alpha transparency value to be used on the entire source bitmap. The **SourceConstantAlpha** value is combined with any per-pixel alpha values in the source bitmap. If you set **SourceConstantAlpha** to 0, it is assumed that your image is transparent. Set the **SourceConstantAlpha** value to 255 (opaque) when you only want to use per-pixel alpha values.

AlphaFormat

This member controls the way the source and destination bitmaps are interpreted. **AlphaFormat** has the following value.

VALUE	MEANING
AC_SRC_ALPHA	This flag is set when the bitmap has an Alpha channel (that is, per-pixel alpha). Note that the APIs use premultiplied alpha, which means that the red, green and blue channel values in the bitmap must be premultiplied with the alpha channel value. For example, if the alpha channel value is x, the red, green and blue channels must be multiplied by x and divided by 0xff prior to the call.

Remarks

When the **AlphaFormat** member is AC_SRC_ALPHA, the source bitmap must be 32 bpp. If it is not, the [AlphaBlend](#) function will fail.

When the **BlendOp** member is AC_SRC_OVER, the source bitmap is placed over the destination bitmap based on the alpha values of the source pixels.

If the source bitmap has no per-pixel alpha value (that is, AC_SRC_ALPHA is not set), the **SourceConstantAlpha** value determines the blend of the source and destination bitmaps, as shown in the following table. Note that SCA is used for **SourceConstantAlpha** here. Also, SCA is divided by 255 because it has a value that ranges from 0 to 255.

Dst.Red	= Src.Red * (SCA/255.0)	+ Dst.Red * (1.0 - (SCA/255.0))
Dst.Green	= Src.Green * (SCA/255.0)	+ Dst.Green * (1.0 - (SCA/255.0))
Dst.Blue	= Src.Blue * (SCA/255.0)	+ Dst.Blue * (1.0 - (SCA/255.0))

If the destination bitmap has an alpha channel, then the blend is as follows.

Dst.Alpha	= Src.Alpha * (SCA/255.0)	+ Dst.Alpha * (1.0 - (SCA/255.0))
-----------	---------------------------	-----------------------------------

If the source bitmap does not use **SourceConstantAlpha** (that is, it equals 0xFF), the per-pixel alpha determines the blend of the source and destination bitmaps, as shown in the following table.

Dst.Red	= Src.Red	+ (1 - Src.Alpha) * Dst.Red
Dst.Green	= Src.Green	+ (1 - Src.Alpha) * Dst.Green
Dst.Blue	= Src.Blue	+ (1 - Src.Alpha) * Dst.Blue

If the destination bitmap has an alpha channel, then the blend is as follows.

Dest.alpha	= Src.Alpha	+ (1 - SrcAlpha) * Dst.Alpha
------------	-------------	------------------------------

If the source has both the **SourceConstantAlpha** (that is, it is not 0xFF) and per-pixel alpha, the source is pre-multiplied by the **SourceConstantAlpha** and then the blend is based on the per-pixel alpha. The following tables show this. Note that **SourceConstantAlpha** is divided by 255 because it has a value that ranges from 0 to 255.

Src.Red	= Src.Red	* SourceConstantAlpha / 255.0;
Src.Green	= Src.Green	* SourceConstantAlpha / 255.0;
Src.Blue	= Src.Blue	* SourceConstantAlpha / 255.0;
Src.Alpha	= Src.Alpha	* SourceConstantAlpha / 255.0;
Dst.Red	= Src.Red	+ (1 - Src.Alpha) * Dst.Red
Dst.Green	= Src.Green	+ (1 - Src.Alpha) * Dst.Green

Dst.Blue	= Src.Blue	+ (1 - Src.Alpha) * Dst.Blue
Dst.Alpha	= Src.Alpha	+ (1 - Src.Alpha) * Dst.Alpha

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[AlphaBlend](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

CancelDC function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CancelDC** function cancels any pending operation on the specified device context (DC).

Syntax

```
BOOL CancelDC(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

A handle to the DC.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **CancelDC** function is used by multithreaded applications to cancel lengthy drawing operations. If thread A initiates a lengthy drawing operation, thread B may cancel that operation by calling this function.

If an operation is canceled, the affected thread returns an error and the result of its drawing operation is undefined. The results are also undefined if no drawing operation was in progress when the function was called.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateThread](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetCurrentThread](#)

Chord function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **Chord** function draws a chord (a region bounded by the intersection of an ellipse and a line segment, called a secant). The chord is outlined by using the current pen and filled by using the current brush.

Syntax

```
BOOL Chord(
    [in] HDC hdc,
    [in] int x1,
    [in] int y1,
    [in] int x2,
    [in] int y2,
    [in] int x3,
    [in] int y3,
    [in] int x4,
    [in] int y4
);
```

Parameters

[in] hdc

A handle to the device context in which the chord appears.

[in] x1

The x-coordinate, in logical coordinates, of the upper-left corner of the bounding rectangle.

[in] y1

The y-coordinate, in logical coordinates, of the upper-left corner of the bounding rectangle.

[in] x2

The x-coordinate, in logical coordinates, of the lower-right corner of the bounding rectangle.

[in] y2

The y-coordinate, in logical coordinates, of the lower-right corner of the bounding rectangle.

[in] x3

The x-coordinate, in logical coordinates, of the endpoint of the radial defining the beginning of the chord.

[in] y3

The y-coordinate, in logical coordinates, of the endpoint of the radial defining the beginning of the chord.

[in] x4

The x-coordinate, in logical coordinates, of the endpoint of the radial defining the end of the chord.

[in] y4

The y-coordinate, in logical coordinates, of the endpoint of the radial defining the end of the chord.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The curve of the chord is defined by an ellipse that fits the specified bounding rectangle. The curve begins at the point where the ellipse intersects the first radial and extends counterclockwise to the point where the ellipse intersects the second radial. The chord is closed by drawing a line from the intersection of the first radial and the curve to the intersection of the second radial and the curve.

If the starting point and ending point of the curve are the same, a complete ellipse is drawn.

The current position is neither used nor updated by **Chord**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AngleArc](#)

[Arc](#)

[ArcTo](#)

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

[Pie](#)

CloseEnhMetaFile function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CloseEnhMetaFile** function closes an enhanced-metafile device context and returns a handle that identifies an enhanced-format metafile.

Syntax

```
HENHMETAFILE CloseEnhMetaFile(  
    [in] HDC hdc  
)
```

Parameters

[in] hdc

Handle to an enhanced-metafile device context.

Return value

If the function succeeds, the return value is a handle to an enhanced metafile.

If the function fails, the return value is **NULL**.

Remarks

An application can use the enhanced-metafile handle returned by the **CloseEnhMetaFile** function to perform the following tasks:

- Display a picture stored in an enhanced metafile
- Create copies of the enhanced metafile
- Enumerate, edit, or copy individual records in the enhanced metafile
- Retrieve an optional description of the metafile contents from the enhanced-metafile header
- Retrieve a copy of the enhanced-metafile header
- Retrieve a binary copy of the enhanced metafile
- Enumerate the colors in the optional palette
- Convert an enhanced-format metafile into a Windows-format metafile

When the application no longer needs the enhanced metafile handle, it should release the handle by calling the [DeleteEnhMetaFile](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CopyEnhMetaFile](#)

[CreateEnhMetaFile](#)

[DeleteEnhMetaFile](#)

[EnumEnhMetaFile](#)

[GetEnhMetaFileBits](#)

[GetWinMetaFileBits](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayEnhMetaFile](#)

CloseFigure function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CloseFigure** function closes an open figure in a path.

Syntax

```
BOOL CloseFigure(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

Handle to the device context in which the figure will be closed.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **CloseFigure** function closes the figure by drawing a line from the current position to the first point of the figure (usually, the point specified by the most recent call to the [MoveToEx](#) function) and then connects the lines by using the line join style. If a figure is closed by using the [LineTo](#) function instead of **CloseFigure**, end caps are used to create the corner instead of a join.

The **CloseFigure** function should only be called if there is an open path bracket in the specified device context.

A figure in a path is open unless it is explicitly closed by using this function. (A figure can be open even if the current point and the starting point of the figure are the same.)

After a call to **CloseFigure**, adding a line or curve to the path starts a new figure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[BeginPath](#)

[EndPath](#)

[ExtCreatePen](#)

[LineTo](#)

[MoveToEx](#)

[Path Functions](#)

[Paths Overview](#)

CloseMetaFile function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CloseMetaFile** function closes a metafile device context and returns a handle that identifies a Windows-format metafile.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [CloseEnhMetaFile](#).

Syntax

```
HMETAFILE CloseMetaFile(  
    [in] HDC hdc  
)
```

Parameters

[in] hdc

Handle to a metafile device context used to create a Windows-format metafile.

Return value

If the function succeeds, the return value is a handle to a Windows-format metafile.

If the function fails, the return value is **NULL**.

Remarks

To convert a Windows-format metafile into a new enhanced-format metafile, use the [SetWinMetaFileBits](#) function.

When an application no longer needs the Windows-format metafile handle, it should delete the handle by calling the [DeleteMetaFile](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CloseEnhMetaFile](#)

[CopyMetaFile](#)

[CreateMetaFile](#)

[DeleteMetaFile](#)

[EnumMetaFile](#)

[GetMetaFileBitsEx](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayMetaFile](#)

[SetWinMetaFileBits](#)

COLORADJUSTMENT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The COLORADJUSTMENT structure defines the color adjustment values used by the [StretchBlt](#) and [StretchDIBits](#) functions when the stretch mode is HALFTONE. You can set the color adjustment values by calling the [SetColorAdjustment](#) function.

Syntax

```
typedef struct tagCOLORADJUSTMENT {
    WORD caSize;
    WORD caFlags;
    WORD caIlluminantIndex;
    WORD caRedGamma;
    WORD caGreenGamma;
    WORD caBlueGamma;
    WORD caReferenceBlack;
    WORD caReferenceWhite;
    SHORT caContrast;
    SHORT caBrightness;
    SHORT caColorfulness;
    SHORT caRedGreenTint;
} COLORADJUSTMENT, *PCOLORADJUSTMENT, *LPCOLORADJUSTMENT;
```

Members

caSize

The size, in bytes, of the structure.

caFlags

Specifies how the output image should be prepared. This member may be set to **NULL** or any combination of the following values.

VALUE	MEANING
CA_NEGATIVE	Specifies that the negative of the original image should be displayed.
CA_LOG_FILTER	Specifies that a logarithmic function should be applied to the final density of the output colors. This will increase the color contrast when the luminance is low.

caIlluminantIndex

The type of standard light source under which the image is viewed. This member may be set to one of the following values.

VALUE	MEANING
ILLUMINANT_DEVICE_DEFAULT	Device's default. Standard used by output devices.

ILLUMINANT_A	Tungsten lamp.
ILLUMINANT_B	Noon sunlight.
ILLUMINANT_C	NTSC daylight.
ILLUMINANT_D50	Normal print.
ILLUMINANT_D55	Bond paper print.
ILLUMINANT_D65	Standard daylight. Standard for CRTs and pictures.
ILLUMINANT_D75	Northern daylight.
ILLUMINANT_F2	Cool white lamp.
ILLUMINANT_TUNGSTEN	Same as ILLUMINANT_A.
ILLUMINANT_DAYLIGHT	Same as ILLUMINANT_C.
ILLUMINANT_FLUORESCENT	Same as ILLUMINANT_F2.
ILLUMINANT_NTSC	Same as ILLUMINANT_C.

caRedGamma

Specifies the n^{th} power gamma-correction value for the red primary of the source colors. The value must be in the range from 2500 to 65,000. A value of 10,000 means no gamma correction.

caGreenGamma

Specifies the n^{th} power gamma-correction value for the green primary of the source colors. The value must be in the range from 2500 to 65,000. A value of 10,000 means no gamma correction.

caBlueGamma

Specifies the n^{th} power gamma-correction value for the blue primary of the source colors. The value must be in the range from 2500 to 65,000. A value of 10,000 means no gamma correction.

caReferenceBlack

The black reference for the source colors. Any colors that are darker than this are treated as black. The value must be in the range from 0 to 4000.

caReferenceWhite

The white reference for the source colors. Any colors that are lighter than this are treated as white. The value must be in the range from 6000 to 10,000.

caContrast

The amount of contrast to be applied to the source object. The value must be in the range from -100 to 100. A value of 0 means no contrast adjustment.

caBrightness

The amount of brightness to be applied to the source object. The value must be in the range from -100 to 100. A

value of 0 means no brightness adjustment.

caColorfulness

The amount of colorfulness to be applied to the source object. The value must be in the range from -100 to 100. A value of 0 means no colorfulness adjustment.

caRedGreenTint

The amount of red or green tint adjustment to be applied to the source object. The value must be in the range from -100 to 100. Positive numbers adjust toward red and negative numbers adjust toward green. Zero means no tint adjustment.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Bitmap Structures](#)

[Bitmaps Overview](#)

[GetColorAdjustment](#)

[SetColorAdjustment](#)

[SetStretchBltMode](#)

[StretchBlt](#)

[StretchDIBits](#)

CombineRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CombineRgn** function combines two regions and stores the result in a third region. The two regions are combined according to the specified mode.

Syntax

```
int CombineRgn(
    [in] HRGN hrgnDst,
    [in] HRGN hrgnSrc1,
    [in] HRGN hrgnSrc2,
    [in] int iMode
);
```

Parameters

[in] *hrgnDst*

A handle to a new region with dimensions defined by combining two other regions. (This region must exist before **CombineRgn** is called.)

[in] *hrgnSrc1*

A handle to the first of two regions to be combined.

[in] *hrgnSrc2*

A handle to the second of two regions to be combined.

[in] *iMode*

A mode indicating how the two regions will be combined. This parameter can be one of the following values.

VALUE	MEANING
RGN_AND	Creates the intersection of the two combined regions.
RGN_COPY	Creates a copy of the region identified by <i>hrgnSrc1</i> .
RGN_DIFF	Combines the parts of <i>hrgnSrc1</i> that are not part of <i>hrgnSrc2</i> .
RGN_OR	Creates the union of two combined regions.
RGN_XOR	Creates the union of two combined regions except for any overlapping areas.

Return value

The return value specifies the type of the resulting region. It can be one of the following values.

RETURN CODE	DESCRIPTION
NULLREGION	The region is empty.
SIMPLEREGION	The region is a single rectangle.
COMPLEXREGION	The region is more than a single rectangle.
ERROR	No region is created.

Remarks

The three regions need not be distinct. For example, the *hrgnSrc1* parameter can equal the *hrgnDest* parameter.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateEllipticRgn](#)

[CreateEllipticRgnIndirect](#)

[CreatePolyPolygonRgn](#)

[CreatePolygonRgn](#)

[CreateRectRgn](#)

[CreateRectRgnIndirect](#)

[CreateRoundRectRgn](#)

[Region Functions](#)

Regions Overview

CombineTransform function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CombineTransform** function concatenates two world-space to page-space transformations.

Syntax

```
BOOL CombineTransform(
    [out] LPXFORM     lpxfOut,
    [in]  const XFORM *lpxf1,
    [in]  const XFORM *lpxf2
);
```

Parameters

[out] *lpxfOut*

A pointer to an [XFORM](#) structure that receives the combined transformation.

[in] *lpxf1*

A pointer to an [XFORM](#) structure that specifies the first transformation.

[in] *lpxf2*

A pointer to an [XFORM](#) structure that specifies the second transformation.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Applying the combined transformation has the same effect as applying the first transformation and then applying the second transformation.

The three transformations need not be distinct. For example, *lpxform1* can point to the same [XFORM](#) structure as *lpxformResult*.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetWorldTransform](#)

[ModifyWorldTransform](#)

[SetWorldTransform](#)

[XFORM](#)

CopyEnhMetaFileA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CopyEnhMetaFile** function copies the contents of an enhanced-format metafile to a specified file.

Syntax

```
HENHMETAFILE CopyEnhMetaFileA(
    [in] HENHMETAFILE hEnh,
    [in] LPCSTR     lpFileName
);
```

Parameters

[in] **hEnh**

A handle to the enhanced metafile to be copied.

[in] **lpFileName**

A pointer to the name of the destination file. If this parameter is **NULL**, the source metafile is copied to memory.

Return value

If the function succeeds, the return value is a handle to the copy of the enhanced metafile.

If the function fails, the return value is **NULL**.

Remarks

Where text arguments must use Unicode characters, use the **CopyEnhMetaFile** function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

Applications can use metafiles stored in memory for temporary operations.

When the application no longer needs the enhanced-metafile handle, it should delete the handle by calling the [DeleteEnhMetaFile](#) function.

NOTE

The wingdi.h header defines **CopyEnhMetaFile** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

CopyEnhMetaFileW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CopyEnhMetaFile** function copies the contents of an enhanced-format metafile to a specified file.

Syntax

```
HENHMETAFILE CopyEnhMetaFileW(
    [in] HENHMETAFILE hEnh,
    [in] LPCWSTR     lpFileName
);
```

Parameters

[in] **hEnh**

A handle to the enhanced metafile to be copied.

[in] **lpFileName**

A pointer to the name of the destination file. If this parameter is **NULL**, the source metafile is copied to memory.

Return value

If the function succeeds, the return value is a handle to the copy of the enhanced metafile.

If the function fails, the return value is **NULL**.

Remarks

Where text arguments must use Unicode characters, use the **CopyEnhMetaFile** function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

Applications can use metafiles stored in memory for temporary operations.

When the application no longer needs the enhanced-metafile handle, it should delete the handle by calling the [DeleteEnhMetaFile](#) function.

NOTE

The wingdi.h header defines **CopyEnhMetaFile** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

CopyMetaFileA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CopyMetaFile** function copies the content of a Windows-format metafile to the specified file.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [CopyEnhMetaFile](#).

Syntax

```
HMETAFILE CopyMetaFileA(
    [in] HMETAFILE unnamedParam1,
    [in] LPCSTR     unnamedParam2
);
```

Parameters

[in] unnamedParam1

A handle to the source Windows-format metafile.

[in] unnamedParam2

A pointer to the name of the destination file. If this parameter is **NULL**, the source metafile is copied to memory.

Return value

If the function succeeds, the return value is a handle to the copy of the Windows-format metafile.

If the function fails, the return value is **NULL**.

Remarks

Where text arguments must use Unicode characters, use this function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

When the application no longer needs the Windows-format metafile handle, it should delete the handle by calling the [DeleteMetaFile](#) function.

NOTE

The wingdi.h header defines **CopyMetaFile** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

CopyMetaFileW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CopyMetaFile** function copies the content of a Windows-format metafile to the specified file.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [CopyEnhMetaFile](#).

Syntax

```
HMETAFILE CopyMetaFileW(
    [in] HMETAFILE unnamedParam1,
    [in] LPCWSTR    unnamedParam2
);
```

Parameters

[in] unnamedParam1

A handle to the source Windows-format metafile.

[in] unnamedParam2

A pointer to the name of the destination file. If this parameter is **NULL**, the source metafile is copied to memory.

Return value

If the function succeeds, the return value is a handle to the copy of the Windows-format metafile.

If the function fails, the return value is **NULL**.

Remarks

Where text arguments must use Unicode characters, use this function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

When the application no longer needs the Windows-format metafile handle, it should delete the handle by calling the [DeleteMetaFile](#) function.

NOTE

The wingdi.h header defines **CopyMetaFile** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

CreateBitmap function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateBitmap** function creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel).

Syntax

```
HBITMAP CreateBitmap(
    [in] int      nWidth,
    [in] int      nHeight,
    [in] UINT     nPlanes,
    [in] UINT     nBitCount,
    [in] const VOID *lpBits
);
```

Parameters

[in] **nWidth**

The bitmap width, in pixels.

[in] **nHeight**

The bitmap height, in pixels.

[in] **nPlanes**

The number of color planes used by the device.

[in] **nBitCount**

The number of bits required to identify the color of a single pixel.

[in] **lpBits**

A pointer to an array of color data used to set the colors in a rectangle of pixels. Each scan line in the rectangle must be word aligned (scan lines that are not word aligned must be padded with zeros). If this parameter is **NULL**, the contents of the new bitmap is undefined.

Return value

If the function succeeds, the return value is a handle to a bitmap.

If the function fails, the return value is **NULL**.

This function can return the following value.

RETURN CODE	DESCRIPTION
ERROR_INVALID_BITMAP	The calculated size of the bitmap is less than zero.

Remarks

The [CreateBitmap](#) function creates a device-dependent bitmap.

After a bitmap is created, it can be selected into a device context by calling the [SelectObject](#) function. However, the bitmap can only be selected into a device context if the bitmap and the DC have the same format.

The [CreateBitmap](#) function can be used to create color bitmaps. However, for performance reasons applications should use [CreateBitmap](#) to create monochrome bitmaps and [CreateCompatibleBitmap](#) to create color bitmaps. Whenever a color bitmap returned from [CreateBitmap](#) is selected into a device context, the system checks that the bitmap matches the format of the device context it is being selected into. Because [CreateCompatibleBitmap](#) takes a device context, it returns a bitmap that has the same format as the specified device context. Thus, subsequent calls to [SelectObject](#) are faster with a color bitmap from [CreateCompatibleBitmap](#) than with a color bitmap returned from [CreateBitmap](#).

If the bitmap is monochrome, zeros represent the foreground color and ones represent the background color for the destination device context.

If an application sets the *nWidth* or *nHeight* parameters to zero, [CreateBitmap](#) returns the handle to a 1-by-1 pixel, monochrome bitmap.

When you no longer need the bitmap, call the [DeleteObject](#) function to delete it.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateBitmapIndirect](#)

[CreateCompatibleBitmap](#)

[CreateDIBitmap](#)

[DeleteObject](#)

[GetBitmapBits](#)

[SelectObject](#)

[SetBitmapBits](#)

CreateBitmapIndirect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateBitmapIndirect** function creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel).

Syntax

```
HBITMAP CreateBitmapIndirect(
    [in] const BITMAP *pbm
);
```

Parameters

[in] pbm

A pointer to a [BITMAP](#) structure that contains information about the bitmap. If an application sets the **bmWidth** or **bmHeight** members to zero, **CreateBitmapIndirect** returns the handle to a 1-by-1 pixel, monochrome bitmap.

Return value

If the function succeeds, the return value is a handle to the bitmap.

If the function fails, the return value is **NULL**.

This function can return the following values.

RETURN CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	One or more of the input parameters is invalid.
ERROR_NOT_ENOUGH_MEMORY	The bitmap is too big for memory to be allocated.

Remarks

The **CreateBitmapIndirect** function creates a device-dependent bitmap.

After a bitmap is created, it can be selected into a device context by calling the [SelectObject](#) function. However, the bitmap can only be selected into a device context if the bitmap and the DC have the same format.

While the **CreateBitmapIndirect** function can be used to create color bitmaps, for performance reasons applications should use **CreateBitmapIndirect** to create monochrome bitmaps and [CreateCompatibleBitmap](#) to create color bitmaps. Whenever a color bitmap from **CreateBitmapIndirect** is selected into a device context, the system must ensure that the bitmap matches the format of the device context it is being selected into. Because [CreateCompatibleBitmap](#) takes a device context, it returns a bitmap that has the same format as the specified device context. Thus, subsequent calls to [SelectObject](#) are faster with a color bitmap from [CreateCompatibleBitmap](#) than with a color bitmap returned from **CreateBitmapIndirect**.

If the bitmap is monochrome, zeros represent the foreground color and ones represent the background color for the destination device context.

When you no longer need the bitmap, call the [DeleteObject](#) function to delete it.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAP](#)

[BitBlt](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateBitmap](#)

[CreateCompatibleBitmap](#)

[CreateDIBitmap](#)

[DeleteObject](#)

[SelectObject](#)

CreateBrushIndirect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateBrushIndirect** function creates a logical brush that has the specified style, color, and pattern.

Syntax

```
HBRUSH CreateBrushIndirect(
    [in] const LOGBRUSH *plbrush
);
```

Parameters

[in] plbrush

A pointer to a [LOGBRUSH](#) structure that contains information about the brush.

Return value

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is **NULL**.

Remarks

A brush is a bitmap that the system uses to paint the interiors of filled shapes.

After an application creates a brush by calling **CreateBrushIndirect**, it can select it into any device context by calling the [SelectObject](#) function.

A brush created by using a monochrome bitmap (one color plane, one bit per pixel) is drawn using the current text and background colors. Pixels represented by a bit set to 0 are drawn with the current text color; pixels represented by a bit set to 1 are drawn with the current background color.

When you no longer need the brush, call the [DeleteObject](#) function to delete it.

ICM: No color is done at brush creation. However, color management is performed when the brush is selected into an ICM-enabled device context.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Brush Functions](#)

[Brushes Overview](#)

[DeleteObject](#)

[GetBrushOrgEx](#)

[LOGBRUSH](#)

[SelectObject](#)

[SetBrushOrgEx](#)

CreateCompatibleBitmap function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateCompatibleBitmap** function creates a bitmap compatible with the device that is associated with the specified device context.

Syntax

```
HBITMAP CreateCompatibleBitmap(
    [in] HDC hdc,
    [in] int cx,
    [in] int cy
);
```

Parameters

[in] `hdc`

A handle to a device context.

[in] `cx`

The bitmap width, in pixels.

[in] `cy`

The bitmap height, in pixels.

Return value

If the function succeeds, the return value is a handle to the compatible bitmap (DDB).

If the function fails, the return value is **NULL**.

Remarks

The color format of the bitmap created by the **CreateCompatibleBitmap** function matches the color format of the device identified by the `hdc` parameter. This bitmap can be selected into any memory device context that is compatible with the original device.

Because memory device contexts allow both color and monochrome bitmaps, the format of the bitmap returned by the **CreateCompatibleBitmap** function differs when the specified device context is a memory device context. However, a compatible bitmap that was created for a nonmemory device context always possesses the same color format and uses the same color palette as the specified device context.

Note: When a memory device context is created, it initially has a 1-by-1 monochrome bitmap selected into it. If this memory device context is used in **CreateCompatibleBitmap**, the bitmap that is created is a *monochrome* bitmap. To create a color bitmap, use the **HDC** that was used to create the memory device context, as shown in the following code:

```
HDC memDC = CreateCompatibleDC ( hDC );
HBITMAP memBM = CreateCompatibleBitmap ( hDC, nWidth, nHeight );
SelectObject ( memDC, memBM );
```

If an application sets the *nWidth* or *nHeight* parameters to zero, **CreateCompatibleBitmap** returns the handle to a 1-by-1 pixel, monochrome bitmap.

If a DIB section, which is a bitmap created by the [CreateDIBSection](#) function, is selected into the device context identified by the *hdc* parameter, **CreateCompatibleBitmap** creates a DIB section.

When you no longer need the bitmap, call the [DeleteObject](#) function to delete it.

Examples

For an example, see [Scaling an Image](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateDIBSection](#)

[DeleteObject](#)

[SelectObject](#)

CreateCompatibleDC function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateCompatibleDC** function creates a memory device context (DC) compatible with the specified device.

Syntax

```
HDC CreateCompatibleDC(  
    [in] HDC hdc  
)
```

Parameters

[in] *hdc*

A handle to an existing DC. If this handle is **NULL**, the function creates a memory DC compatible with the application's current screen.

Return value

If the function succeeds, the return value is the handle to a memory DC.

If the function fails, the return value is **NULL**.

Remarks

A memory DC exists only in memory. When the memory DC is created, its display surface is exactly one monochrome pixel wide and one monochrome pixel high. Before an application can use a memory DC for drawing operations, it must select a bitmap of the correct width and height into the DC. To select a bitmap into a DC, use the [CreateCompatibleBitmap](#) function, specifying the height, width, and color organization required.

When a memory DC is created, all attributes are set to normal default values. The memory DC can be used as a normal DC. You can set the attributes; obtain the current settings of its attributes; and select pens, brushes, and regions.

The **CreateCompatibleDC** function can only be used with devices that support raster operations. An application can determine whether a device supports these operations by calling the [GetDeviceCaps](#) function.

When you no longer need the memory DC, call the [DeleteDC](#) function. We recommend that you call [DeleteDC](#) to delete the DC. However, you can also call [DeleteObject](#) with the HDC to delete the DC.

If *hdc* is **NULL**, the thread that calls **CreateCompatibleDC** owns the HDC that is created. When this thread is destroyed, the HDC is no longer valid. Thus, if you create the HDC and pass it to another thread, then exit the first thread, the second thread will not be able to use the HDC.

ICM: If the DC that is passed to this function is enabled for Image Color Management (ICM), the DC created by the function is ICM-enabled. The source and destination color spaces are specified in the DC.

Examples

For an example, see [Capturing an Image](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateCompatibleBitmap](#)

[DeleteDC](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetDeviceCaps](#)

CreateDCA function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The `CreateDC` function creates a device context (DC) for a device using the specified name.

Syntax

```
HDC CreateDCA(
    LPCSTR      pwszDriver,
    [in] LPCSTR  pwszDevice,
    LPCSTR      pszPort,
    [in] const DEVMODEA *pdm
);
```

Parameters

`pwszDriver`

A pointer to a null-terminated character string that specifies either DISPLAY or the name of a specific display device. For printing, we recommend that you pass **NULL** to *lpszDriver* because GDI ignores *lpszDriver* for printer devices.

`[in] pwszDevice`

A pointer to a null-terminated character string that specifies the name of the specific output device being used, as shown by the Print Manager (for example, Epson FX-80). It is not the printer model name. The *lpszDevice* parameter must be used.

To obtain valid names for displays, call [EnumDisplayDevices](#).

If *lpszDriver* is DISPLAY or the device name of a specific display device, then *lpszDevice* must be **NULL** or that same device name. If *lpszDevice* is **NULL**, then a DC is created for the primary display device.

If there are multiple monitors on the system, calling `CreateDC(TEXT("DISPLAY"),NULL,NULL,NULL)` will create a DC covering all the monitors.

`pszPort`

This parameter is ignored and should be set to **NULL**. It is provided only for compatibility with 16-bit Windows.

`[in] pdm`

A pointer to a [DEVMODE](#) structure containing device-specific initialization data for the device driver. The [DocumentProperties](#) function retrieves this structure filled in for a specified device. The *pdm* parameter must be **NULL** if the device driver is to use the default initialization (if any) specified by the user.

If *lpszDriver* is DISPLAY, *pdm* must be **NULL**; GDI then uses the display device's current [DEVMODE](#).

Return value

If the function succeeds, the return value is the handle to a DC for the specified device.

If the function fails, the return value is **NULL**.

Remarks

Note that the handle to the DC can only be used by a single thread at any one time.

For parameters *lpszDriver* and *lpszDevice*, call [EnumDisplayDevices](#) to obtain valid names for displays.

When you no longer need the DC, call the [DeleteDC](#) function.

If *lpszDriver* or *lpszDevice* is DISPLAY, the thread that calls [CreateDC](#) owns the HDC that is created. When this thread is destroyed, the HDC is no longer valid. Thus, if you create the HDC and pass it to another thread, then exit the first thread, the second thread will not be able to use the HDC.

When you call [CreateDC](#) to create the HDC for a display device, you must pass to *pdm* either **NULL** or a pointer to [DEVMODE](#) that matches the current DEVMODE of the display device that *lpszDevice* specifies. We recommend to pass **NULL** and not to try to exactly match the DEVMODE for the current display device.

When you call [CreateDC](#) to create the HDC for a printer device, the printer driver validates the [DEVMODE](#). If the printer driver determines that the DEVMODE is invalid (that is, printer driver can't convert or consume the DEVMODE), the printer driver provides a default DEVMODE to create the HDC for the printer device.

ICM: To enable ICM, set the **dmICMMETHOD** member of the [DEVMODE](#) structure (pointed to by the *pInitData* parameter) to the appropriate value.

Examples

For an example, see [Capturing an Image](#).

NOTE

The wingdi.h header defines CreateDC as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DEVMODE](#)

[DOCINFO](#)

[DeleteDC](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[DocumentProperties](#)

[EnumDisplayDevices](#)

[Multiple Display Monitors](#)

[StartDoc](#)

CreateDCW function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The `CreateDC` function creates a device context (DC) for a device using the specified name.

Syntax

```
HDC CreateDCW(
    LPCWSTR     pwszDriver,
    [in] LPCWSTR   pwszDevice,
    LPCWSTR     pszPort,
    [in] const DEVMODEW *pdm
);
```

Parameters

`pwszDriver`

A pointer to a null-terminated character string that specifies either DISPLAY or the name of a specific display device. For printing, we recommend that you pass **NULL** to *lpszDriver* because GDI ignores *lpszDriver* for printer devices.

`[in] pwszDevice`

A pointer to a null-terminated character string that specifies the name of the specific output device being used, as shown by the Print Manager (for example, Epson FX-80). It is not the printer model name. The *lpszDevice* parameter must be used.

To obtain valid names for displays, call [EnumDisplayDevices](#).

If *lpszDriver* is DISPLAY or the device name of a specific display device, then *lpszDevice* must be **NULL** or that same device name. If *lpszDevice* is **NULL**, then a DC is created for the primary display device.

If there are multiple monitors on the system, calling `CreateDC(TEXT("DISPLAY"),NULL,NULL,NULL)` will create a DC covering all the monitors.

`pszPort`

This parameter is ignored and should be set to **NULL**. It is provided only for compatibility with 16-bit Windows.

`[in] pdm`

A pointer to a [DEVMODE](#) structure containing device-specific initialization data for the device driver. The [DocumentProperties](#) function retrieves this structure filled in for a specified device. The *pdm* parameter must be **NULL** if the device driver is to use the default initialization (if any) specified by the user.

If *lpszDriver* is DISPLAY, *pdm* must be **NULL**; GDI then uses the display device's current [DEVMODE](#).

Return value

If the function succeeds, the return value is the handle to a DC for the specified device.

If the function fails, the return value is **NULL**.

Remarks

Note that the handle to the DC can only be used by a single thread at any one time.

For parameters *lpszDriver* and *lpszDevice*, call [EnumDisplayDevices](#) to obtain valid names for displays.

When you no longer need the DC, call the [DeleteDC](#) function.

If *lpszDriver* or *lpszDevice* is DISPLAY, the thread that calls [CreateDC](#) owns the HDC that is created. When this thread is destroyed, the HDC is no longer valid. Thus, if you create the HDC and pass it to another thread, then exit the first thread, the second thread will not be able to use the HDC.

When you call [CreateDC](#) to create the HDC for a display device, you must pass to *pdm* either **NULL** or a pointer to [DEVMODE](#) that matches the current DEVMODE of the display device that *lpszDevice* specifies. We recommend to pass **NULL** and not to try to exactly match the DEVMODE for the current display device.

When you call [CreateDC](#) to create the HDC for a printer device, the printer driver validates the [DEVMODE](#). If the printer driver determines that the DEVMODE is invalid (that is, printer driver can't convert or consume the DEVMODE), the printer driver provides a default DEVMODE to create the HDC for the printer device.

ICM: To enable ICM, set the **dmICMMETHOD** member of the [DEVMODE](#) structure (pointed to by the *pInitData* parameter) to the appropriate value.

Examples

For an example, see [Capturing an Image](#).

NOTE

The wingdi.h header defines CreateDC as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DEVMODE](#)

[DOCINFO](#)

[DeleteDC](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[DocumentProperties](#)

[EnumDisplayDevices](#)

[Multiple Display Monitors](#)

[StartDoc](#)

CreateDIBitmap function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateDIBitmap** function creates a compatible bitmap (DDB) from a DIB and, optionally, sets the bitmap bits.

Syntax

```
HBITMAP CreateDIBitmap(
    [in] HDC                 hdc,
    [in] const BITMAPINFOHEADER *pbmih,
    [in] DWORD                flInit,
    [in] const VOID            *pjBits,
    [in] const BITMAPINFO      *pbmi,
    [in] UINT                 iUsage
);
```

Parameters

[in] *hdc*

A handle to a device context.

[in] *pbmih*

A pointer to a bitmap information header structure, [BITMAPV5HEADER](#).

If *fdwInit* is CBM_INIT, the function uses the bitmap information header structure to obtain the desired width and height of the bitmap as well as other information. Note that a positive value for the height indicates a bottom-up DIB while a negative value for the height indicates a top-down DIB. Calling **CreateDIBitmap** with *fdwInit* as CBM_INIT is equivalent to calling the [CreateCompatibleBitmap](#) function to create a DDB in the format of the device and then calling the [SetDIBits](#) function to translate the DIB bits to the DDB.

[in] *flInit*

Specifies how the system initializes the bitmap bits. The following value is defined.

VALUE	MEANING
CBM_INIT	If this flag is set, the system uses the data pointed to by the <i>lpblInit</i> and <i>lpbmi</i> parameters to initialize the bitmap bits. If this flag is clear, the data pointed to by those parameters is not used.

If *fdwInit* is zero, the system does not initialize the bitmap bits.

[in] *pjBits*

A pointer to an array of bytes containing the initial bitmap data. The format of the data depends on the **biBitCount** member of the [BITMAPINFO](#) structure to which the *lpbmi* parameter points.

[in] *pbmi*

A pointer to a [BITMAPINFO](#) structure that describes the dimensions and color format of the array pointed to by the *lpbInit* parameter.

[in] *iUsage*

Specifies whether the **bmiColors** member of the [BITMAPINFO](#) structure was initialized and, if so, whether **bmiColors** contains explicit red, green, blue (RGB) values or palette indexes. The *fuUsage* parameter must be one of the following values.

VALUE	MEANING
DIB_PAL_COLORS	A color table is provided and consists of an array of 16-bit indexes into the logical palette of the device context into which the bitmap is to be selected.
DIB_RGB_COLORS	A color table is provided and contains literal RGB values.

Return value

If the function succeeds, the return value is a handle to the compatible bitmap.

If the function fails, the return value is **NULL**.

Remarks

The DDB that is created will be whatever bit depth your reference DC is. To create a bitmap that is of different bit depth, use [CreateDIBSection](#).

For a device to reach optimal bitmap-drawing speed, specify *fdwInit* as CBM_INIT. Then, use the same color depth DIB as the video mode. When the video is running 4- or 8-bpp, use DIB_PAL_COLORS.

The CBM_CREATDIB flag for the *fdwInit* parameter is no longer supported.

When you no longer need the bitmap, call the [DeleteObject](#) function to delete it.

ICM: No color management is performed. The contents of the resulting bitmap are not color matched after the bitmap has been created.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPINFO](#)

[BITMAPINFOHEADER](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateCompatibleBitmap](#)

[CreateDIBSection](#)

[DeleteObject](#)

[GetDeviceCaps](#)

[GetSystemPaletteEntries](#)

[SelectObject](#)

[SetDIBits](#)

CreateDIBPatternBrush function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateDIBPatternBrush** function creates a logical brush that has the pattern specified by the specified device-independent bitmap (DIB). The brush can subsequently be selected into any device context that is associated with a device that supports raster operations.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the [CreateDIBPatternBrushPt](#) function.

Syntax

```
HBRUSH CreateDIBPatternBrush(
    [in] HGLOBAL h,
    [in] UINT     iUsage
);
```

Parameters

[in] *h*

A handle to a global memory object containing a packed DIB, which consists of a [BITMAPINFO](#) structure immediately followed by an array of bytes defining the pixels of the bitmap.

[in] *iUsage*

Specifies whether the *bmiColors* member of the [BITMAPINFO](#) structure is initialized and, if so, whether this member contains explicit red, green, blue (RGB) values or indexes into a logical palette. The *fuColorSpec* parameter must be one of the following values.

VALUE	MEANING
DIB_PAL_COLORS	A color table is provided and consists of an array of 16-bit indexes into the logical palette of the device context into which the brush is to be selected.
DIB_RGB_COLORS	A color table is provided and contains literal RGB values.

Return value

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is **NULL**.

Remarks

When an application selects a two-color DIB pattern brush into a monochrome device context, the system does

not acknowledge the colors specified in the DIB; instead, it displays the pattern brush using the current background and foreground colors of the device context. Pixels mapped to the first color of the DIB (offset 0 in the DIB color table) are displayed using the foreground color; pixels mapped to the second color (offset 1 in the color table) are displayed using the background color.

When you no longer need the brush, call the [DeleteObject](#) function to delete it.

ICM: No color is done at brush creation. However, color management is performed when the brush is selected into an ICM-enabled device context.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPINFO](#)

[Brush Functions](#)

[Brushes Overview](#)

[CreateDIBPatternBrushPt](#)

[CreateHatchBrush](#)

[CreatePatternBrush](#)

[CreateSolidBrush](#)

[DeleteObject](#)

[SetBkColor](#)

[SetTextColor](#)

CreateDIBPatternBrushPt function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateDIBPatternBrushPt** function creates a logical brush that has the pattern specified by the device-independent bitmap (DIB).

Syntax

```
HBRUSH CreateDIBPatternBrushPt(
    [in] const VOID *lpPackedDIB,
    [in] UINT      iUsage
);
```

Parameters

[in] *lpPackedDIB*

A pointer to a packed DIB consisting of a [BITMAPINFO](#) structure immediately followed by an array of bytes defining the pixels of the bitmap.

[in] *iUsage*

Specifies whether the *bmiColors* member of the [BITMAPINFO](#) structure contains a valid color table and, if so, whether the entries in this color table contain explicit red, green, blue (RGB) values or palette indexes. The *iUsage* parameter must be one of the following values.

VALUE	MEANING
DIB_PAL_COLORS	A color table is provided and consists of an array of 16-bit indexes into the logical palette of the device context into which the brush is to be selected.
DIB_RGB_COLORS	A color table is provided and contains literal RGB values.

Return value

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is **NULL**.

Remarks

A brush is a bitmap that the system uses to paint the interiors of filled shapes.

After an application creates a brush by calling **CreateDIBPatternBrushPt**, it can select that brush into any device context by calling the [SelectObject](#) function.

When you no longer need the brush, call the [DeleteObject](#) function to delete it.

ICM: No color is done at brush creation. However, color management is performed when the brush is selected into an ICM-enabled device context.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPINFO](#)

[Brush Functions](#)

[Brushes Overview](#)

[CreateDIBPatternBrush](#)

[CreateHatchBrush](#)

[CreatePatternBrush](#)

[CreateSolidBrush](#)

[DeleteObject](#)

[GetBrushOrgEx](#)

[SelectObject](#)

[SetBrushOrgEx](#)

CreateDIBSection function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **CreateDIBSection** function creates a DIB that applications can write to directly. The function gives you a pointer to the location of the bitmap bit values. You can supply a handle to a file-mapping object that the function will use to create the bitmap, or you can let the system allocate the memory for the bitmap.

Syntax

```
HBITMAP CreateDIBSection(
    [in]    HDC             hdc,
    [in]    const BITMAPINFO *pbmi,
    [in]    UINT            usage,
    [out]   VOID            **ppvBits,
    [in]    HANDLE          hSection,
    [in]    DWORD           offset
);
```

Parameters

[in] `hdc`

A handle to a device context. If the value of *iUsage* is **DIB_PAL_COLORS**, the function uses this device context's logical palette to initialize the DIB colors.

[in] `pbmi`

A pointer to a **BITMAPINFO** structure that specifies various attributes of the DIB, including the bitmap dimensions and colors.

[in] `usage`

The type of data contained in the **bmiColors** array member of the **BITMAPINFO** structure pointed to by *pbmi* (either logical palette indexes or literal RGB values). The following values are defined.

VALUE	MEANING
DIB_PAL_COLORS	The bmiColors member is an array of 16-bit indexes into the logical palette of the device context specified by <i>hdc</i> .
DIB_RGB_COLORS	The BITMAPINFO structure contains an array of literal RGB values.

[out] `ppvBits`

A pointer to a variable that receives a pointer to the location of the DIB bit values.

[in] `hSection`

A handle to a file-mapping object that the function will use to create the DIB. This parameter can be **NULL**.

If *hSection* is not **NULL**, it must be a handle to a file-mapping object created by calling the **CreateFileMapping** function with the **PAGE_READWRITE** or **PAGE_WRITECOPY** flag. Read-only DIB sections are not supported.

Handles created by other means will cause [CreateDIBSection](#) to fail.

If *hSection* is not **NULL**, the [CreateDIBSection](#) function locates the bitmap bit values at offset *dwOffset* in the file-mapping object referred to by *hSection*. An application can later retrieve the *hSection* handle by calling the [GetObject](#) function with the **HBITMAP** returned by [CreateDIBSection](#).

If *hSection* is **NULL**, the system allocates memory for the DIB. In this case, the [CreateDIBSection](#) function ignores the *dwOffset* parameter. An application cannot later obtain a handle to this memory. The **dshSection** member of the **DIBSECTION** structure filled in by calling the [GetObject](#) function will be **NULL**.

[in] *offset*

The offset from the beginning of the file-mapping object referenced by *hSection* where storage for the bitmap bit values is to begin. This value is ignored if *hSection* is **NULL**. The bitmap bit values are aligned on doubleword boundaries, so *dwOffset* must be a multiple of the size of a **DWORD**.

Return value

If the function succeeds, the return value is a handle to the newly created DIB, and **ppvBits* points to the bitmap bit values.

If the function fails, the return value is **NULL**, and **ppvBits* is **NULL**. To get extended error information, call [GetLastError](#).

[GetLastError](#) can return the following value:

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	One or more of the input parameters is invalid.

Remarks

As noted above, if *hSection* is **NULL**, the system allocates memory for the DIB. The system closes the handle to that memory when you later delete the DIB by calling the [DeleteObject](#) function. If *hSection* is not **NULL**, you must close the *hSection* memory handle yourself after calling [DeleteObject](#) to delete the bitmap.

You cannot paste a DIB section from one application into another application.

[CreateDIBSection](#) does not use the **BITMAPINFOHEADER** parameters *biXPelsPerMeter* or *biYPelsPerMeter* and will not provide resolution information in the **BITMAPINFO** structure.

You need to guarantee that the GDI subsystem has completed any drawing to a bitmap created by [CreateDIBSection](#) before you draw to the bitmap yourself. Access to the bitmap must be synchronized. Do this by calling the [GdiFlush](#) function. This applies to any use of the pointer to the bitmap bit values, including passing the pointer in calls to functions such as [SetDIBits](#).

ICM: No color management is done.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPINFO](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateFileMapping](#)

[DIBSECTION](#)

[DeleteObject](#)

[GdiFlush](#)

[GetDIBColorTable](#)

[GetObject](#)

[SetDIBColorTable](#)

[SetDIBits](#)

CreateDiscardableBitmap function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateDiscardableBitmap** function creates a discardable bitmap that is compatible with the specified device. The bitmap has the same bits-per-pixel format and the same color palette as the device. An application can select this bitmap as the current bitmap for a memory device that is compatible with the specified device.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the [CreateCompatibleBitmap](#) function.

Syntax

```
HBITMAP CreateDiscardableBitmap(  
    [in] HDC hdc,  
    [in] int cx,  
    [in] int cy  
)
```

Parameters

[in] `hdc`

A handle to a device context.

[in] `cx`

The width, in pixels, of the bitmap.

[in] `cy`

The height, in pixels, of the bitmap.

Return value

If the function succeeds, the return value is a handle to the compatible bitmap (DDB).

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the bitmap, call the [DeleteObject](#) function to delete it.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateCompatibleBitmap](#)

[DeleteObject](#)

CreateEllipticRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The `CreateEllipticRgn` function creates an elliptical region.

Syntax

```
HRGN CreateEllipticRgn(  
    [in] int x1,  
    [in] int y1,  
    [in] int x2,  
    [in] int y2  
);
```

Parameters

[in] `x1`

Specifies the x-coordinate in logical units, of the upper-left corner of the bounding rectangle of the ellipse.

[in] `y1`

Specifies the y-coordinate in logical units, of the upper-left corner of the bounding rectangle of the ellipse.

[in] `x2`

Specifies the x-coordinate in logical units, of the lower-right corner of the bounding rectangle of the ellipse.

[in] `y2`

Specifies the y-coordinate in logical units, of the lower-right corner of the bounding rectangle of the ellipse.

Return value

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the `HRGN` object, call the [DeleteObject](#) function to delete it.

A bounding rectangle defines the size, shape, and orientation of the region: The long sides of the rectangle define the length of the ellipse's major axis; the short sides define the length of the ellipse's minor axis; and the center of the rectangle defines the intersection of the major and minor axes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateEllipticRegionIndirect](#)

[DeleteObject](#)

[Region Functions](#)

[Regions Overview](#)

[SelectObject](#)

CreateEllipticRgnIndirect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The `CreateEllipticRgnIndirect` function creates an elliptical region.

Syntax

```
HRGN CreateEllipticRgnIndirect(
    [in] const RECT *lprect
);
```

Parameters

[in] `lprect`

Pointer to a [RECT](#) structure that contains the coordinates of the upper-left and lower-right corners of the bounding rectangle of the ellipse in logical units.

Return value

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the **HRGN** object, call the [DeleteObject](#) function to delete it.

A bounding rectangle defines the size, shape, and orientation of the region: The long sides of the rectangle define the length of the ellipse's major axis; the short sides define the length of the ellipse's minor axis; and the center of the rectangle defines the intersection of the major and minor axes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateEllipticRegn](#)

[DeleteObject](#)

[RECT](#)

[Region Functions](#)

[Regions Overview](#)

[SelectObject](#)

CreateEnhMetaFileA function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **CreateEnhMetaFile** function creates a device context for an enhanced-format metafile. This device context can be used to store a device-independent picture.

Syntax

```
HDC CreateEnhMetaFileA(
    [in] HDC      hdc,
    [in] LPCSTR   lpFilename,
    [in] const RECT *lprc,
    [in] LPCSTR   lpDesc
);
```

Parameters

[in] `hdc`

A handle to a reference device for the enhanced metafile. This parameter can be **NULL**; for more information, see Remarks.

[in] `lpFilename`

A pointer to the file name for the enhanced metafile to be created. If this parameter is **NULL**, the enhanced metafile is memory based and its contents are lost when it is deleted by using the [DeleteEnhMetaFile](#) function.

[in] `lprc`

A pointer to a [RECT](#) structure that specifies the dimensions (in .01-millimeter units) of the picture to be stored in the enhanced metafile.

[in] `lpDesc`

A pointer to a string that specifies the name of the application that created the picture, as well as the picture's title. This parameter can be **NULL**; for more information, see Remarks.

Return value

If the function succeeds, the return value is a handle to the device context for the enhanced metafile.

If the function fails, the return value is **NULL**.

Remarks

Where text arguments must use Unicode characters, use the [CreateEnhMetaFile](#) function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

The system uses the reference device identified by the `hdcRef` parameter to record the resolution and units of the device on which a picture originally appeared. If the `hdcRef` parameter is **NULL**, it uses the current display device for reference.

The **left** and **top** members of the [RECT](#) structure pointed to by the */pRect* parameter must be less than the **right** and **bottom** members, respectively. Points along the edges of the rectangle are included in the picture. If */pRect* is **NULL**, the graphics device interface (GDI) computes the dimensions of the smallest rectangle that surrounds the picture drawn by the application. The */pRect* parameter should be provided where possible.

The string pointed to by the */pDescription* parameter must contain a null character between the application name and the picture name and must terminate with two null characters, for example, "XYZ Graphics Editor\0Bald Eagle\0\0", where \0 represents the null character. If */pDescription* is **NULL**, there is no corresponding entry in the enhanced-metafile header.

Applications use the device context created by this function to store a graphics picture in an enhanced metafile. The handle identifying this device context can be passed to any GDI function.

After an application stores a picture in an enhanced metafile, it can display the picture on any output device by calling the [PlayEnhMetaFile](#) function. When displaying the picture, the system uses the rectangle pointed to by the */pRect* parameter and the resolution data from the reference device to position and scale the picture.

The device context returned by this function contains the same default attributes associated with any new device context.

Applications must use the [GetWinMetaFileBits](#) function to convert an enhanced metafile to the older Windows metafile format.

The file name for the enhanced metafile should use the .emf extension.

Examples

For an example, see [Creating an Enhanced Metafile](#).

NOTE

The wingdi.h header defines CreateEnhMetaFile as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CloseEnhMetaFile](#)

[DeleteEnhMetaFile](#)

[GetEnhMetaFileDescription](#)

[GetEnhMetaFileHeader](#)

[GetWinMetaFileBits](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayEnhMetaFile](#)

[RECT](#)

CreateEnhMetaFileW function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **CreateEnhMetaFile** function creates a device context for an enhanced-format metafile. This device context can be used to store a device-independent picture.

Syntax

```
HDC CreateEnhMetaFileW(
    [in] HDC      hdc,
    [in] LPCWSTR  lpFilename,
    [in] const RECT *lprc,
    [in] LPCWSTR  lpDesc
);
```

Parameters

[in] hdc

A handle to a reference device for the enhanced metafile. This parameter can be **NULL**; for more information, see Remarks.

[in] lpFilename

A pointer to the file name for the enhanced metafile to be created. If this parameter is **NULL**, the enhanced metafile is memory based and its contents are lost when it is deleted by using the [DeleteEnhMetaFile](#) function.

[in] lprc

A pointer to a [RECT](#) structure that specifies the dimensions (in .01-millimeter units) of the picture to be stored in the enhanced metafile.

[in] lpDesc

A pointer to a string that specifies the name of the application that created the picture, as well as the picture's title. This parameter can be **NULL**; for more information, see Remarks.

Return value

If the function succeeds, the return value is a handle to the device context for the enhanced metafile.

If the function fails, the return value is **NULL**.

Remarks

Where text arguments must use Unicode characters, use the **CreateEnhMetaFile** function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

The system uses the reference device identified by the *hdcRef* parameter to record the resolution and units of the device on which a picture originally appeared. If the *hdcRef* parameter is **NULL**, it uses the current display device for reference.

The **left** and **top** members of the [RECT](#) structure pointed to by the */pRect* parameter must be less than the **right** and **bottom** members, respectively. Points along the edges of the rectangle are included in the picture. If */pRect* is **NULL**, the graphics device interface (GDI) computes the dimensions of the smallest rectangle that surrounds the picture drawn by the application. The */pRect* parameter should be provided where possible.

The string pointed to by the */pDescription* parameter must contain a null character between the application name and the picture name and must terminate with two null characters, for example, "XYZ Graphics Editor\0Bald Eagle\0\0", where \0 represents the null character. If */pDescription* is **NULL**, there is no corresponding entry in the enhanced-metafile header.

Applications use the device context created by this function to store a graphics picture in an enhanced metafile. The handle identifying this device context can be passed to any GDI function.

After an application stores a picture in an enhanced metafile, it can display the picture on any output device by calling the [PlayEnhMetaFile](#) function. When displaying the picture, the system uses the rectangle pointed to by the */pRect* parameter and the resolution data from the reference device to position and scale the picture.

The device context returned by this function contains the same default attributes associated with any new device context.

Applications must use the [GetWinMetaFileBits](#) function to convert an enhanced metafile to the older Windows metafile format.

The file name for the enhanced metafile should use the .emf extension.

Examples

For an example, see [Creating an Enhanced Metafile](#).

NOTE

The wingdi.h header defines CreateEnhMetaFile as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CloseEnhMetaFile](#)

[DeleteEnhMetaFile](#)

[GetEnhMetaFileDescription](#)

[GetEnhMetaFileHeader](#)

[GetWinMetaFileBits](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayEnhMetaFile](#)

[RECT](#)

CreateFontA function (wingdi.h)

4/21/2022 • 12 minutes to read • [Edit Online](#)

The **CreateFont** function creates a logical font with the specified characteristics. The logical font can subsequently be selected as the font for any device.

Syntax

```
HFONT CreateFont(
    [in] int    cHeight,
    [in] int    cWidth,
    [in] int    cEscapement,
    [in] int    cOrientation,
    [in] int    cWeight,
    [in] DWORD  bItalic,
    [in] DWORD  bUnderline,
    [in] DWORD  bStrikeOut,
    [in] DWORD  iCharSet,
    [in] DWORD  iOutPrecision,
    [in] DWORD  iClipPrecision,
    [in] DWORD  iQuality,
    [in] DWORD  iPitchAndFamily,
    [in] LPCSTR pszFaceName
);
```

Parameters

[in] *cHeight*

The height, in logical units, of the font's character cell or character. The character height value (also known as the em height) is the character cell height value minus the internal-leading value. The font mapper interprets the value specified in *nHeight* in the following manner.

VALUE	MEANING
> 0	The font mapper transforms this value into device units and matches it against the cell height of the available fonts.
0	The font mapper uses a default height value when it searches for a match.
< 0	The font mapper transforms this value into device units and matches its absolute value against the character height of the available fonts.

For all height comparisons, the font mapper looks for the largest font that does not exceed the requested size.

This mapping occurs when the font is used for the first time.

For the MM_TEXT mapping mode, you can use the following formula to specify a height for a font with a specified point size:

```
nHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);
```

[in] cWidth

The average width, in logical units, of characters in the requested font. If this value is zero, the font mapper chooses a closest match value. The closest match value is determined by comparing the absolute values of the difference between the current device's aspect ratio and the digitized aspect ratio of available fonts.

[in] cEscapement

The angle, in tenths of degrees, between the escapement vector and the x-axis of the device. The escapement vector is parallel to the base line of a row of text.

When the graphics mode is set to GM_ADVANCED, you can specify the escapement angle of the string independently of the orientation angle of the string's characters.

When the graphics mode is set to GM_COMPATIBLE, *nEscapement* specifies both the escapement and orientation. You should set *nEscapement* and *nOrientation* to the same value.

[in] cOrientation

The angle, in tenths of degrees, between each character's base line and the x-axis of the device.

[in] cWeight

The weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is zero, a default weight is used.

The following values are defined for convenience.

WEIGHT	VALUE
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400

FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

[in] **bItalic**

Specifies an italic font if set to TRUE.

[in] **bUnderline**

Specifies an underlined font if set to TRUE.

[in] **bStrikeOut**

A strikeout font if set to TRUE.

[in] **iCharSet**

The character set. The following values are predefined:

- ANSI_CHARSET
- BALTIC_CHARSET
- CHINESEBIG5_CHARSET
- DEFAULT_CHARSET
- EASTEUROPE_CHARSET
- GB2312_CHARSET
- GREEK_CHARSET
- HANGUL_CHARSET
- MAC_CHARSET
- OEM_CHARSET
- RUSSIAN_CHARSET
- SHIFTJIS_CHARSET

- SYMBOL_CHARSET
- TURKISH_CHARSET
- VIETNAMESE_CHARSET

Korean language edition of Windows:

- JOHAB_CHARSET

Middle East language edition of Windows:

- ARABIC_CHARSET
- HEBREW_CHARSET

Thai language edition of Windows:

- THAI_CHARSET

The OEM_CHARSET value specifies a character set that is operating-system dependent.

DEFAULT_CHARSET is set to a value based on the current system locale. For example, when the system locale is English (United States), it is set as ANSI_CHARSET.

Fonts with other character sets may exist in the operating system. If an application uses a font with an unknown character set, it should not attempt to translate or interpret strings that are rendered with that font.

To ensure consistent results when creating a font, do not specify OEM_CHARSET or DEFAULT_CHARSET. If you specify a typeface name in the *lpszFace* parameter, make sure that the *fdwCharSet* value matches the character set of the typeface specified in *lpszFace*.

[in] iOutPrecision

The output precision. The output precision defines how closely the output must match the requested font's height, width, character orientation, escapement, pitch, and font type. It can be one of the following values.

VALUE	MEANING
OUT_CHARACTER_PRECIS	Not used.
OUT_DEFAULT_PRECIS	The default font mapper behavior.
OUT_DEVICE_PRECIS	Instructs the font mapper to choose a Device font when the system contains multiple fonts with the same name.
OUT_OUTLINE_PRECIS	This value instructs the font mapper to choose from TrueType and other outline-based fonts.
OUT_PS_ONLY_PRECIS	Instructs the font mapper to choose from only PostScript fonts. If there are no PostScript fonts installed in the system, the font mapper returns to default behavior.
OUT_RASTER_PRECIS	Instructs the font mapper to choose a raster font when the system contains multiple fonts with the same name.
OUT_STRING_PRECIS	This value is not used by the font mapper, but it is returned when raster fonts are enumerated.

OUT_STROKE_PRECIS	This value is not used by the font mapper, but it is returned when TrueType, other outline-based fonts, and vector fonts are enumerated.
OUT_TT_ONLY_PRECIS	Instructs the font mapper to choose from only TrueType fonts. If there are no TrueType fonts installed in the system, the font mapper returns to default behavior.
OUT_TT_PRECIS	Instructs the font mapper to choose a TrueType font when the system contains multiple fonts with the same name.

Applications can use the OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS, OUT_TT_PRECIS, and OUT_PS_ONLY_PRECIS values to control how the font mapper chooses a font when the operating system contains more than one font with a specified name. For example, if an operating system contains a font named Symbol in raster and TrueType form, specifying OUT_TT_PRECIS forces the font mapper to choose the TrueType version. Specifying OUT_TT_ONLY_PRECIS forces the font mapper to choose a TrueType font, even if it must substitute a TrueType font of another name.

[in] iClipPrecision

The clipping precision. The clipping precision defines how to clip characters that are partially outside the clipping region. It can be one or more of the following values.

VALUE	MEANING
CLIP_CHARACTER_PRECIS	Not used.
CLIP_DEFAULT_PRECIS	Specifies default clipping behavior.
CLIP_DFA_DISABLE	Windows XP SP1: Turns off font association for the font. Note that this flag is not guaranteed to have any effect on any platform after Windows Server 2003.
CLIP_EMBEDDED	You must specify this flag to use an embedded read-only font.
CLIP_LH_ANGLES	When this value is used, the rotation for all fonts depends on whether the orientation of the coordinate system is left-handed or right-handed. If not used, device fonts always rotate counterclockwise, but the rotation of other fonts is dependent on the orientation of the coordinate system. For more information about the orientation of coordinate systems, see the description of the <i>nOrientation</i> parameter
CLIP_MASK	Not used.

CLIP_DFA_OVERRIDE	Turns off font association for the font. This is identical to CLIP_DFA_DISABLE, but it can have problems in some situations; the recommended flag to use is CLIP_DFA_DISABLE.
CLIP_STROKE_PRECIS	Not used by the font mapper, but is returned when raster, vector, or TrueType fonts are enumerated. For compatibility, this value is always returned when enumerating fonts.
CLIP_TT_ALWAYS	Not used.

[in] iQuality

The output quality. The output quality defines how carefully GDI must attempt to match the logical-font attributes to those of an actual physical font. It can be one of the following values.

VALUE	MEANING
ANTIALIASED_QUALITY	Font is antialiased, or smoothed, if the font supports it and the size of the font is not too small or too large.
CLEARTYPE_QUALITY	If set, text is rendered (when possible) using ClearType antialiasing method. See Remarks for more information.
DEFAULT_QUALITY	Appearance of the font does not matter.
DRAFT_QUALITY	Appearance of the font is less important than when the PROOF_QUALITY value is used. For GDI raster fonts, scaling is enabled, which means that more font sizes are available, but the quality may be lower. Bold, italic, underline, and strikeout fonts are synthesized, if necessary.
NONANTIALIASED_QUALITY	Font is never antialiased, that is, font smoothing is not done.
PROOF_QUALITY	Character quality of the font is more important than exact matching of the logical-font attributes. For GDI raster fonts, scaling is disabled and the font closest in size is chosen. Although the chosen font size may not be mapped exactly when PROOF_QUALITY is used, the quality of the font is high and there is no distortion of appearance. Bold, italic, underline, and strikeout fonts are synthesized, if necessary.

If the output quality is DEFAULT_QUALITY, DRAFT_QUALITY, or PROOF_QUALITY, then the font is antialiased if the SPI_GETFONTSMOOTHING system parameter is TRUE. Users can control this system parameter from the Control Panel. (The precise wording of the setting in the Control panel depends on the version of Windows, but it will be words to the effect of "Smooth edges of screen fonts".)

[in] iPitchAndFamily

The pitch and family of the font. The two low-order bits specify the pitch of the font and can be one of the

following values:

- DEFAULT_PITCH
- FIXED_PITCH
- VARIABLE_PITCH

The four high-order bits specify the font family and can be one of the following values.

VALUE	MEANING
FF_DECORATIVE	Novelty fonts. Old English is an example.
FF_DONTCARE	Use default font.
FF_MODERN	Fonts with constant stroke width, with or without serifs. Pica, Elite, and Courier New are examples.
FF_ROMAN	Fonts with variable stroke width and with serifs. MS Serif is an example.
FF_SCRIPT	Fonts designed to look like handwriting. Script and Cursive are examples.
FF_SWISS	Fonts with variable stroke width and without serifs. MS Sans Serif is an example.

An application can specify a value for the *fdwPitchAndFamily* parameter by using the Boolean OR operator to join a pitch constant with a family constant.

Font families describe the look of a font in a general way. They are intended for specifying fonts when the exact typeface requested is not available.

[in] *pszFaceName*

A pointer to a null-terminated string that specifies the typeface name of the font. The length of this string must not exceed 32 characters, including the terminating null character. The [EnumFontFamilies](#) function can be used to enumerate the typeface names of all currently available fonts. For more information, see the Remarks.

If *lpszFace* is **NULL** or empty string, GDI uses the first font that matches the other specified attributes.

Return value

If the function succeeds, the return value is a handle to a logical font.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the font, call the [DeleteObject](#) function to delete it.

To help protect the copyrights of vendors who provide fonts for Windows, applications should always report the exact name of a selected font. Because available fonts can vary from system to system, do not assume that the

selected font is always the same as the requested font. For example, if you request a font named Palatino, but no such font is available on the system, the font mapper will substitute a font that has similar attributes but a different name. Always report the name of the selected font to the user.

To get the appropriate font on different language versions of the OS, call [EnumFontFamiliesEx](#) with the desired font characteristics in the [LOGFONT](#) structure, then retrieve the appropriate typeface name and create the font using [CreateFont](#) or [CreateFontIndirect](#).

The font mapper for [CreateFont](#), [CreateFontIndirect](#), and [CreateFontIndirectEx](#) recognizes both the English and the localized typeface name, regardless of locale.

The following situations do not support ClearType antialiasing:

- Text rendered on a printer.
- A display set for 256 colors or less.
- Text rendered to a terminal server client.
- The font is not a TrueType font or an OpenType font with TrueType outlines. For example, the following do not support ClearType antialiasing: Type 1 fonts, Postscript OpenType fonts without TrueType outlines, bitmap fonts, vector fonts, and device fonts.
- The font has tuned embedded bitmaps, only for the font sizes that contain the embedded bitmaps. For example, this occurs commonly in East Asian fonts.

Examples

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    switch (message)
    {

        case WM_PAINT:
        {
            RECT rect;
            HBRUSH hBrush;
            HFONT hFont;
            hdc = BeginPaint(hWnd, &ps);

            //Logical units are device dependent pixels, so this will create a handle to a logical font that
            //is 48 pixels in height.
            //The width, when set to 0, will cause the font mapper to choose the closest matching value.
            //The font face name will be Impact.
            hFont = CreateFont(48,0,0,0,FW_DONTCARE, FALSE, TRUE, FALSE, DEFAULT_CHARSET, OUT_OUTLINE_PRECIS,
                CLIP_DEFAULT_PRECIS,CLEARTYPE_QUALITY, VARIABLE_PITCH,TEXT("Impact"));
            SelectObject(hdc, hFont);

            //Sets the coordinates for the rectangle in which the text is to be formatted.
            SetRect(&rect, 100,100,700,200);
            SetTextColor(hdc, RGB(255,0,0));
            DrawText(hdc, TEXT("Drawing Text with Impact"), -1,&rect, DT_NOCLIP);

            //Logical units are device dependent pixels, so this will create a handle to a logical font that
            //is 36 pixels in height.
            //The width, when set to 20, will cause the font mapper to choose a font which, in this case, is
            //stretched.
            //The font face name will be Times New Roman. This time nEscapement is at -300 tenths of a
            //degree (-30 degrees)
            hFont = CreateFont(36,20,-300,0,FW_DONTCARE, FALSE, TRUE, FALSE, DEFAULT_CHARSET, OUT_OUTLINE_PRECIS,
                CLIP_DEFAULT_PRECIS,CLEARTYPE_QUALITY, VARIABLE_PITCH,TEXT("Times New Roman"));
            SelectObject(hdc.hFont);
```

```

//Sets the coordinates for the rectangle in which the text is to be formatted.
SetRect(&rect, 100, 200, 900, 800);
SetTextColor(hdc, RGB(0,128,0));
DrawText(hdc, TEXT("Drawing Text with Times New Roman"), -1,&rect, DT_NOCLIP);

//Logical units are device dependent pixels, so this will create a handle to a logical font that
is 36 pixels in height.
//The width, when set to 10, will cause the font mapper to choose a font which, in this case, is
compressed.
//The font face name will be Arial. This time nEscapement is at 250 tenths of a degree (25
degrees)
hFont = CreateFont(36,10,250,0,FW_DONTCARE, FALSE, TRUE, FALSE, DEFAULT_CHARSET, OUT_OUTLINE_PRECIS,
CLIP_DEFAULT_PRECIS,ANTIALIASED_QUALITY, VARIABLE_PITCH,TEXT("Arial"));
SelectObject(hdc,hFont);

//Sets the coordinates for the rectangle in which the text is to be formatted.
SetRect(&rect, 500, 200, 1400, 600);
SetTextColor(hdc, RGB(0,0,255));
DrawText(hdc, TEXT("Drawing Text with Arial"), -1,&rect, DT_NOCLIP);
DeleteObject(hFont);

EndPaint(hWnd, &ps);
break;
}
case WM_DESTROY:
PostQuitMessage(0);
break;
default:
return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

For another example, see "Setting Fonts for Menu-Item Text Strings" in [Using Menus](#).

NOTE

The wingdi.h header defines CreateFont as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[CreateFontIndirect](#)

[CreateFontIndirectEx](#)

[DeleteObject](#)

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

[SelectObject](#)

CreateFontIndirectA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateFontIndirect** function creates a logical font that has the specified characteristics. The font can subsequently be selected as the current font for any device context.

Syntax

```
HFONT CreateFontIndirectA(  
    [in] const LOGFONTA *lplf  
) ;
```

Parameters

[in] lplf

A pointer to a [LOGFONT](#) structure that defines the characteristics of the logical font.

Return value

If the function succeeds, the return value is a handle to a logical font.

If the function fails, the return value is **NULL**.

Remarks

The **CreateFontIndirect** function creates a logical font with the characteristics specified in the [LOGFONT](#) structure. When this font is selected by using the [SelectObject](#) function, GDI's font mapper attempts to match the logical font with an existing physical font. If it fails to find an exact match, it provides an alternative whose characteristics match as many of the requested characteristics as possible.

To get the appropriate font on different language versions of the OS, call [EnumFontFamiliesEx](#) with the desired font characteristics in the [LOGFONT](#) structure, retrieve the appropriate typeface name, and create the font using [CreateFont](#) or [CreateFontIndirect](#).

When you no longer need the font, call the [DeleteObject](#) function to delete it.

The fonts for many East Asian languages have two typeface names: an English name and a localized name. [CreateFont](#) and [CreateFontIndirect](#) take the localized typeface name only on a system locale that matches the language, while they take the English typeface name on all other system locales. The best method is to try one name and, on failure, try the other. Note that [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) return the English typeface name if the system locale does not match the language of the font.

The font mapper for [CreateFont](#), [CreateFontIndirect](#), and [CreateFontIndirectEx](#) recognizes both the English and the localized typeface name, regardless of locale.

Examples

For an example, see [Creating a Logical Font](#).

NOTE

The wingdi.h header defines CreateFontIndirect as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

- [CreateFont](#)
- [CreateFontIndirectEx](#)
- [DeleteObject](#)
- [EnumFontFamilies](#)
- [EnumFontFamiliesEx](#)
- [EnumFonts](#)
- [Font and Text Functions](#)
- [Fonts and Text Overview](#)
- [LOGFONT](#)
- [SelectObject](#)

CreateFontIndirectExA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateFontIndirectEx** function specifies a logical font that has the characteristics in the specified structure. The font can subsequently be selected as the current font for any device context.

Syntax

```
HFONT CreateFontIndirectExA(  
    [in] const ENUMLOGFONTEXDVA *unnamedParam1  
) ;
```

Parameters

[in] unnamedParam1

Pointer to an [ENUMLOGFONTEXDV](#) structure that defines the characteristics of a multiple master font.

Note, this function ignores the **elfDesignVector** member in [ENUMLOGFONTEXDV](#).

Return value

If the function succeeds, the return value is the handle to the new [ENUMLOGFONTEXDV](#) structure.

If the function fails, the return value is zero. No extended error information is available.

Remarks

The **CreateFontIndirectEx** function creates a logical font with the characteristics specified in the [ENUMLOGFONTEXDV](#) structure. When this font is selected by using the [SelectObject](#) function, GDI's font mapper attempts to match the logical font with an existing physical font. If it fails to find an exact match, it provides an alternative whose characteristics match as many of the requested characteristics as possible.

When you no longer need the font, call the [DeleteObject](#) function to delete it.

The font mapper for [CreateFont](#), [CreateFontIndirect](#), and [CreateFontIndirectEx](#) recognizes both the English and the localized typeface name, regardless of locale.

NOTE

The wingdi.h header defines [CreateFontIndirectEx](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateFont](#)

[CreateFontIndirect](#)

[ENUMLOGFONTEXDV](#)

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

CreateFontIndirectExW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateFontIndirectEx** function specifies a logical font that has the characteristics in the specified structure. The font can subsequently be selected as the current font for any device context.

Syntax

```
HFONT CreateFontIndirectExW(
    [in] const ENUMLOGFONTEXDVW *unnamedParam1
);
```

Parameters

[in] unnamedParam1

Pointer to an [ENUMLOGFONTEXDV](#) structure that defines the characteristics of a multiple master font.

Note, this function ignores the **elfDesignVector** member in [ENUMLOGFONTEXDV](#).

Return value

If the function succeeds, the return value is the handle to the new [ENUMLOGFONTEXDV](#) structure.

If the function fails, the return value is zero. No extended error information is available.

Remarks

The **CreateFontIndirectEx** function creates a logical font with the characteristics specified in the [ENUMLOGFONTEXDV](#) structure. When this font is selected by using the [SelectObject](#) function, GDI's font mapper attempts to match the logical font with an existing physical font. If it fails to find an exact match, it provides an alternative whose characteristics match as many of the requested characteristics as possible.

When you no longer need the font, call the [DeleteObject](#) function to delete it.

The font mapper for [CreateFont](#), [CreateFontIndirect](#), and [CreateFontIndirectEx](#) recognizes both the English and the localized typeface name, regardless of locale.

NOTE

The wingdi.h header defines [CreateFontIndirectEx](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateFont](#)

[CreateFontIndirect](#)

[ENUMLOGFONTEXDV](#)

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

CreateFontIndirectW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateFontIndirect** function creates a logical font that has the specified characteristics. The font can subsequently be selected as the current font for any device context.

Syntax

```
HFONT CreateFontIndirectW(
    [in] const LOGFONTW *lpLF
);
```

Parameters

[in] lpLF

A pointer to a [LOGFONT](#) structure that defines the characteristics of the logical font.

Return value

If the function succeeds, the return value is a handle to a logical font.

If the function fails, the return value is **NULL**.

Remarks

The **CreateFontIndirect** function creates a logical font with the characteristics specified in the [LOGFONT](#) structure. When this font is selected by using the [SelectObject](#) function, GDI's font mapper attempts to match the logical font with an existing physical font. If it fails to find an exact match, it provides an alternative whose characteristics match as many of the requested characteristics as possible.

To get the appropriate font on different language versions of the OS, call [EnumFontFamiliesEx](#) with the desired font characteristics in the [LOGFONT](#) structure, retrieve the appropriate typeface name, and create the font using [CreateFont](#) or [CreateFontIndirect](#).

When you no longer need the font, call the [DeleteObject](#) function to delete it.

The fonts for many East Asian languages have two typeface names: an English name and a localized name. [CreateFont](#) and [CreateFontIndirect](#) take the localized typeface name only on a system locale that matches the language, while they take the English typeface name on all other system locales. The best method is to try one name and, on failure, try the other. Note that [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) return the English typeface name if the system locale does not match the language of the font.

The font mapper for [CreateFont](#), [CreateFontIndirect](#), and [CreateFontIndirectEx](#) recognizes both the English and the localized typeface name, regardless of locale.

Examples

For an example, see [Creating a Logical Font](#).

NOTE

The wingdi.h header defines CreateFontIndirect as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

- [CreateFont](#)
- [CreateFontIndirectEx](#)
- [DeleteObject](#)
- [EnumFontFamilies](#)
- [EnumFontFamiliesEx](#)
- [EnumFonts](#)
- [Font and Text Functions](#)
- [Fonts and Text Overview](#)
- [LOGFONT](#)
- [SelectObject](#)

CreateFontW function (wingdi.h)

4/21/2022 • 12 minutes to read • [Edit Online](#)

The **CreateFont** function creates a logical font with the specified characteristics. The logical font can subsequently be selected as the font for any device.

Syntax

```
HFONT CreateFontW(
    [in] int      cHeight,
    [in] int      cWidth,
    [in] int      cEscapement,
    [in] int      cOrientation,
    [in] int      cWeight,
    [in] DWORD    bItalic,
    [in] DWORD    bUnderline,
    [in] DWORD    bStrikeOut,
    [in] DWORD    iCharSet,
    [in] DWORD    iOutPrecision,
    [in] DWORD    iClipPrecision,
    [in] DWORD    iQuality,
    [in] DWORD    iPitchAndFamily,
    [in] LPCWSTR  pszFaceName
);
```

Parameters

[in] **cHeight**

The height, in logical units, of the font's character cell or character. The character height value (also known as the em height) is the character cell height value minus the internal-leading value. The font mapper interprets the value specified in *nHeight* in the following manner.

VALUE	MEANING
> 0	The font mapper transforms this value into device units and matches it against the cell height of the available fonts.
0	The font mapper uses a default height value when it searches for a match.
< 0	The font mapper transforms this value into device units and matches its absolute value against the character height of the available fonts.

For all height comparisons, the font mapper looks for the largest font that does not exceed the requested size.

This mapping occurs when the font is used for the first time.

For the MM_TEXT mapping mode, you can use the following formula to specify a height for a font with a specified point size:

```
nHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);
```

[in] cWidth

The average width, in logical units, of characters in the requested font. If this value is zero, the font mapper chooses a closest match value. The closest match value is determined by comparing the absolute values of the difference between the current device's aspect ratio and the digitized aspect ratio of available fonts.

[in] cEscapement

The angle, in tenths of degrees, between the escapement vector and the x-axis of the device. The escapement vector is parallel to the base line of a row of text.

When the graphics mode is set to GM_ADVANCED, you can specify the escapement angle of the string independently of the orientation angle of the string's characters.

When the graphics mode is set to GM_COMPATIBLE, *nEscapement* specifies both the escapement and orientation. You should set *nEscapement* and *nOrientation* to the same value.

[in] cOrientation

The angle, in tenths of degrees, between each character's base line and the x-axis of the device.

[in] cWeight

The weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is zero, a default weight is used.

The following values are defined for convenience.

WEIGHT	VALUE
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400

FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

[in] **bItalic**

Specifies an italic font if set to TRUE.

[in] **bUnderline**

Specifies an underlined font if set to TRUE.

[in] **bStrikeOut**

A strikeout font if set to TRUE.

[in] **iCharSet**

The character set. The following values are predefined:

- ANSI_CHARSET
- BALTIC_CHARSET
- CHINESEBIG5_CHARSET
- DEFAULT_CHARSET
- EASTEUROPE_CHARSET
- GB2312_CHARSET
- GREEK_CHARSET
- HANGUL_CHARSET
- MAC_CHARSET
- OEM_CHARSET
- RUSSIAN_CHARSET
- SHIFTJIS_CHARSET

- SYMBOL_CHARSET
- TURKISH_CHARSET
- VIETNAMESE_CHARSET

Korean language edition of Windows:

- JOHAB_CHARSET

Middle East language edition of Windows:

- ARABIC_CHARSET
- HEBREW_CHARSET

Thai language edition of Windows:

- THAI_CHARSET

The OEM_CHARSET value specifies a character set that is operating-system dependent.

DEFAULT_CHARSET is set to a value based on the current system locale. For example, when the system locale is English (United States), it is set as ANSI_CHARSET.

Fonts with other character sets may exist in the operating system. If an application uses a font with an unknown character set, it should not attempt to translate or interpret strings that are rendered with that font.

To ensure consistent results when creating a font, do not specify OEM_CHARSET or DEFAULT_CHARSET. If you specify a typeface name in the *lpszFace* parameter, make sure that the *fdwCharSet* value matches the character set of the typeface specified in *lpszFace*.

[in] iOutPrecision

The output precision. The output precision defines how closely the output must match the requested font's height, width, character orientation, escapement, pitch, and font type. It can be one of the following values.

VALUE	MEANING
OUT_CHARACTER_PRECIS	Not used.
OUT_DEFAULT_PRECIS	The default font mapper behavior.
OUT_DEVICE_PRECIS	Instructs the font mapper to choose a Device font when the system contains multiple fonts with the same name.
OUT_OUTLINE_PRECIS	This value instructs the font mapper to choose from TrueType and other outline-based fonts.
OUT_PS_ONLY_PRECIS	Instructs the font mapper to choose from only PostScript fonts. If there are no PostScript fonts installed in the system, the font mapper returns to default behavior.
OUT_RASTER_PRECIS	Instructs the font mapper to choose a raster font when the system contains multiple fonts with the same name.
OUT_STRING_PRECIS	This value is not used by the font mapper, but it is returned when raster fonts are enumerated.

OUT_STROKE_PRECIS	This value is not used by the font mapper, but it is returned when TrueType, other outline-based fonts, and vector fonts are enumerated.
OUT_TT_ONLY_PRECIS	Instructs the font mapper to choose from only TrueType fonts. If there are no TrueType fonts installed in the system, the font mapper returns to default behavior.
OUT_TT_PRECIS	Instructs the font mapper to choose a TrueType font when the system contains multiple fonts with the same name.

Applications can use the OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS, OUT_TT_PRECIS, and OUT_PS_ONLY_PRECIS values to control how the font mapper chooses a font when the operating system contains more than one font with a specified name. For example, if an operating system contains a font named Symbol in raster and TrueType form, specifying OUT_TT_PRECIS forces the font mapper to choose the TrueType version. Specifying OUT_TT_ONLY_PRECIS forces the font mapper to choose a TrueType font, even if it must substitute a TrueType font of another name.

[in] iClipPrecision

The clipping precision. The clipping precision defines how to clip characters that are partially outside the clipping region. It can be one or more of the following values.

VALUE	MEANING
CLIP_CHARACTER_PRECIS	Not used.
CLIP_DEFAULT_PRECIS	Specifies default clipping behavior.
CLIP_DFA_DISABLE	Windows XP SP1: Turns off font association for the font. Note that this flag is not guaranteed to have any effect on any platform after Windows Server 2003.
CLIP_EMBEDDED	You must specify this flag to use an embedded read-only font.
CLIP_LH_ANGLES	When this value is used, the rotation for all fonts depends on whether the orientation of the coordinate system is left-handed or right-handed. If not used, device fonts always rotate counterclockwise, but the rotation of other fonts is dependent on the orientation of the coordinate system. For more information about the orientation of coordinate systems, see the description of the <i>nOrientation</i> parameter
CLIP_MASK	Not used.

CLIP_DFA_OVERRIDE	Turns off font association for the font. This is identical to CLIP_DFA_DISABLE, but it can have problems in some situations; the recommended flag to use is CLIP_DFA_DISABLE.
CLIP_STROKE_PRECIS	Not used by the font mapper, but is returned when raster, vector, or TrueType fonts are enumerated. For compatibility, this value is always returned when enumerating fonts.
CLIP_TT_ALWAYS	Not used.

[in] iQuality

The output quality. The output quality defines how carefully GDI must attempt to match the logical-font attributes to those of an actual physical font. It can be one of the following values.

VALUE	MEANING
ANTIALIASED_QUALITY	Font is antialiased, or smoothed, if the font supports it and the size of the font is not too small or too large.
CLEARTYPE_QUALITY	If set, text is rendered (when possible) using ClearType antialiasing method. See Remarks for more information.
DEFAULT_QUALITY	Appearance of the font does not matter.
DRAFT_QUALITY	Appearance of the font is less important than when the PROOF_QUALITY value is used. For GDI raster fonts, scaling is enabled, which means that more font sizes are available, but the quality may be lower. Bold, italic, underline, and strikeout fonts are synthesized, if necessary.
NONANTIALIASED_QUALITY	Font is never antialiased, that is, font smoothing is not done.
PROOF_QUALITY	Character quality of the font is more important than exact matching of the logical-font attributes. For GDI raster fonts, scaling is disabled and the font closest in size is chosen. Although the chosen font size may not be mapped exactly when PROOF_QUALITY is used, the quality of the font is high and there is no distortion of appearance. Bold, italic, underline, and strikeout fonts are synthesized, if necessary.

If the output quality is DEFAULT_QUALITY, DRAFT_QUALITY, or PROOF_QUALITY, then the font is antialiased if the SPI_GETFONTSMOOTHING system parameter is TRUE. Users can control this system parameter from the Control Panel. (The precise wording of the setting in the Control panel depends on the version of Windows, but it will be words to the effect of "Smooth edges of screen fonts".)

[in] iPitchAndFamily

The pitch and family of the font. The two low-order bits specify the pitch of the font and can be one of the

following values:

- DEFAULT_PITCH
- FIXED_PITCH
- VARIABLE_PITCH

The four high-order bits specify the font family and can be one of the following values.

VALUE	MEANING
FF_DECORATIVE	Novelty fonts. Old English is an example.
FF_DONTCARE	Use default font.
FF_MODERN	Fonts with constant stroke width, with or without serifs. Pica, Elite, and Courier New are examples.
FF_ROMAN	Fonts with variable stroke width and with serifs. MS Serif is an example.
FF_SCRIPT	Fonts designed to look like handwriting. Script and Cursive are examples.
FF_SWISS	Fonts with variable stroke width and without serifs. MS Sans Serif is an example.

An application can specify a value for the *fdwPitchAndFamily* parameter by using the Boolean OR operator to join a pitch constant with a family constant.

Font families describe the look of a font in a general way. They are intended for specifying fonts when the exact typeface requested is not available.

[in] *pszFaceName*

A pointer to a null-terminated string that specifies the typeface name of the font. The length of this string must not exceed 32 characters, including the terminating null character. The [EnumFontFamilies](#) function can be used to enumerate the typeface names of all currently available fonts. For more information, see the Remarks.

If *lpszFace* is **NULL** or empty string, GDI uses the first font that matches the other specified attributes.

Return value

If the function succeeds, the return value is a handle to a logical font.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the font, call the [DeleteObject](#) function to delete it.

To help protect the copyrights of vendors who provide fonts for Windows, applications should always report the exact name of a selected font. Because available fonts can vary from system to system, do not assume that the

selected font is always the same as the requested font. For example, if you request a font named Palatino, but no such font is available on the system, the font mapper will substitute a font that has similar attributes but a different name. Always report the name of the selected font to the user.

To get the appropriate font on different language versions of the OS, call [EnumFontFamiliesEx](#) with the desired font characteristics in the [LOGFONT](#) structure, then retrieve the appropriate typeface name and create the font using [CreateFont](#) or [CreateFontIndirect](#).

The font mapper for [CreateFont](#), [CreateFontIndirect](#), and [CreateFontIndirectEx](#) recognizes both the English and the localized typeface name, regardless of locale.

The following situations do not support ClearType antialiasing:

- Text rendered on a printer.
- A display set for 256 colors or less.
- Text rendered to a terminal server client.
- The font is not a TrueType font or an OpenType font with TrueType outlines. For example, the following do not support ClearType antialiasing: Type 1 fonts, Postscript OpenType fonts without TrueType outlines, bitmap fonts, vector fonts, and device fonts.
- The font has tuned embedded bitmaps, only for the font sizes that contain the embedded bitmaps. For example, this occurs commonly in East Asian fonts.

Examples

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    switch (message)
    {

        case WM_PAINT:
        {
            RECT rect;
            HBRUSH hBrush;
            HFONT hFont;
            hdc = BeginPaint(hWnd, &ps);

            //Logical units are device dependent pixels, so this will create a handle to a logical font that
            //is 48 pixels in height.
            //The width, when set to 0, will cause the font mapper to choose the closest matching value.
            //The font face name will be Impact.
            hFont = CreateFont(48,0,0,0,FW_DONTCARE, FALSE, TRUE, FALSE, DEFAULT_CHARSET, OUT_OUTLINE_PRECIS,
                CLIP_DEFAULT_PRECIS,CLEARTYPE_QUALITY, VARIABLE_PITCH,TEXT("Impact"));
            SelectObject(hdc, hFont);

            //Sets the coordinates for the rectangle in which the text is to be formatted.
            SetRect(&rect, 100,100,700,200);
            SetTextColor(hdc, RGB(255,0,0));
            DrawText(hdc, TEXT("Drawing Text with Impact"), -1,&rect, DT_NOCLIP);

            //Logical units are device dependent pixels, so this will create a handle to a logical font that
            //is 36 pixels in height.
            //The width, when set to 20, will cause the font mapper to choose a font which, in this case, is
            //stretched.
            //The font face name will be Times New Roman. This time nEscapement is at -300 tenths of a
            //degree (-30 degrees)
            hFont = CreateFont(36,20,-300,0,FW_DONTCARE, FALSE, TRUE, FALSE, DEFAULT_CHARSET, OUT_OUTLINE_PRECIS,
                CLIP_DEFAULT_PRECIS,CLEARTYPE_QUALITY, VARIABLE_PITCH,TEXT("Times New Roman"));
            SelectObject(hdc.hFont);
```

```

//Sets the coordinates for the rectangle in which the text is to be formatted.
SetRect(&rect, 100, 200, 900, 800);
SetTextColor(hdc, RGB(0,128,0));
DrawText(hdc, TEXT("Drawing Text with Times New Roman"), -1,&rect, DT_NOCLIP);

//Logical units are device dependent pixels, so this will create a handle to a logical font that
is 36 pixels in height.
//The width, when set to 10, will cause the font mapper to choose a font which, in this case, is
compressed.
//The font face name will be Arial. This time nEscapement is at 250 tenths of a degree (25
degrees)
hFont = CreateFont(36,10,250,0,FW_DONTCARE, FALSE, TRUE, FALSE, DEFAULT_CHARSET, OUT_OUTLINE_PRECIS,
CLIP_DEFAULT_PRECIS,ANTIALIASED_QUALITY, VARIABLE_PITCH,TEXT("Arial"));
SelectObject(hdc,hFont);

//Sets the coordinates for the rectangle in which the text is to be formatted.
SetRect(&rect, 500, 200, 1400, 600);
SetTextColor(hdc, RGB(0,0,255));
DrawText(hdc, TEXT("Drawing Text with Arial"), -1,&rect, DT_NOCLIP);
DeleteObject(hFont);

EndPaint(hWnd, &ps);
break;
}
case WM_DESTROY:
PostQuitMessage(0);
break;
default:
return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

For another example, see "Setting Fonts for Menu-Item Text Strings" in [Using Menus](#).

NOTE

The wingdi.h header defines CreateFont as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[CreateFontIndirect](#)

[CreateFontIndirectEx](#)

[DeleteObject](#)

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

[SelectObject](#)

CreateHalftonePalette function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateHalftonePalette** function creates a halftone palette for the specified device context (DC).

Syntax

```
HPALETTE CreateHalftonePalette(
    [in] HDC hdc
);
```

Parameters

`[in] hdc`

A handle to the device context.

Return value

If the function succeeds, the return value is a handle to a logical halftone palette.

If the function fails, the return value is zero.

Remarks

An application should create a halftone palette when the stretching mode of a device context is set to HALFTONE. The logical halftone palette returned by **CreateHalftonePalette** should then be selected and realized into the device context before the [StretchBlt](#) or [StretchDIBits](#) function is called.

When you no longer need the palette, call the [DeleteObject](#) function to delete it.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[DeleteObject](#)

[RealizePalette](#)

[SelectPalette](#)

[SetStretchBltMode](#)

[StretchBlt](#)

[StretchDIBits](#)

CreateHatchBrush function (wingdi.h)

4/21/2022 • 5 minutes to read • [Edit Online](#)

The **CreateHatchBrush** function creates a logical brush that has the specified hatch pattern and color.

Syntax

```
HBRUSH CreateHatchBrush(
    [in] int      iHatch,
    [in] COLORREF color
);
```

Parameters

[in] **iHatch**

The [hatch style of the brush](#). This parameter can be one of the following values.

VALUE	MEANING
HS_BDIAGONAL	45-degree upward left-to-right hatch
HS_CROSS	Horizontal and vertical crosshatch
HS_DIAGCROSS	45-degree crosshatch
HS_FDIAGONAL	45-degree downward left-to-right hatch
HS_HORIZONTAL	Horizontal hatch
HS_VERTICAL	Vertical hatch

[in] **color**

The foreground color of the brush that is used for the hatches. To create a [COLORREF](#) color value, use the [RGB](#) macro.

Return value

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is **NULL**.

Remarks

A brush is a bitmap that the system uses to paint the interiors of filled shapes.

After an application creates a brush by calling [CreateHatchBrush](#), it can select that brush into any device context by calling the [SelectObject](#) function. It can also call [SetBkMode](#) to affect the rendering of the brush.

If an application uses a hatch brush to fill the backgrounds of both a parent and a child window with matching color, you must set the brush origin before painting the background of the child window. You can do this by calling the [SetBrushOrgEx](#) function. Your application can retrieve the current brush origin by calling the [GetBrushOrgEx](#) function.

When you no longer need the brush, call the [DeleteObject](#) function to delete it.

ICM: No color is defined at brush creation. However, color management is performed when the brush is selected into an ICM-enabled device context.

Examples

The following example creates a logical brush that has the specified hatch pattern and color. You can also set a hatch brush background to transparent or to opaque.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <tchar.h>
#include <stddef.h>

#include <gdiplus.h>
#include <assert.h>

using namespace Gdiplus;

// Reference to the GDI+ static library).
#pragma comment (lib,"Gdiplus.lib")

// Global variables

// The main window class name.
static TCHAR szWindowClass[] = _T("win32app");

// The string that appears in the application's title bar.
static TCHAR szTitle[] = _T("Win32 Application Hatch Brush");

HINSTANCE hInst;

#define BTN_MYBUTTON_ID_1      503
#define BTN_MYBUTTON_ID_2      504

// Forward declarations of functions included in this code module:
HRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{
    UNREFERENCED_PARAMETER(lpCmdLine);
    UNREFERENCED_PARAMETER(hPrevInstance);

    WNDCLASSEX wcex;
```

```

wcex.cbSize = sizeof(WNDCLASSEX);
wcex.style      = CS_HREDRAW | CS_VREDRAW;
wcex.lpfnWndProc = WndProc;
wcex.cbClsExtra = 0;
wcex.cbWndExtra = 0;
wcex.hInstance   = hInstance;
wcex.hIcon       = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
wcex.hCursor     = LoadCursor(NULL, IDC_ARROW);
wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wcex.lpszMenuName = NULL;
wcex.lpszClassName = szWindowClass;
wcex.hIconSm      = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_APPLICATION));

if (!RegisterClassEx(&wcex))
{
    MessageBox(NULL,
        _T("Call to RegisterClassEx failed!"),
        _T("Win32 Guided Tour"),
        NULL);

    return 1;
}

hInst = hInstance; // Store instance handle in our global variable

// The parameters to CreateWindow:
// szWindowClass: the name of the application
// szTitle: the text that appears in the title bar
// WS_OVERLAPPEDWINDOW: the type of window to create
// CW_USEDEFAULT, CW_USEDEFAULT: initial position (x, y)
// 500, 100: initial size (width, length)
// NULL: the parent of this window
// NULL: this application does not have a menu bar
// hInstance: the first parameter from WinMain
// NULL: not used in this application
HWND hWnd = CreateWindow(
    szWindowClass,
    szTitle,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    1000, 500,
    NULL,
    NULL,
    hInstance,
    NULL
);

if (!hWnd)
{
    MessageBox(NULL,
        _T("Call to CreateWindow failed!"),
        _T("Win32 Guided Tour"),
        NULL);

    return 1;
}

// Create button controls.
CreateWindowEx(NULL, L"BUTTON", L"Transparent", WS_VISIBLE | WS_CHILD,
    35, 35, 120, 20, hWnd, (HMENU)BTN_MYBUTTON_ID_1, NULL, NULL);

CreateWindowEx(NULL, L"BUTTON", L"Opaque", WS_VISIBLE | WS_CHILD,
    35, 65, 120, 20, hWnd, (HMENU)BTN_MYBUTTON_ID_2, NULL, NULL);

// The parameters to ShowWindow:
// hWnd: the value returned from CreateWindow
// nCmdShow: the fourth parameter from WinMain
ShowWindow(hWnd,
    nCmdShow);

```

```

UpdateWindow(hWnd);

// Main message loop:
MSG msg;
while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}

return (int) msg.wParam;
}

/***
*   This function creates the following rectangles:
*       1. An outer rectangle using a solid brush with blue background.
*       2. An inner rectangle using a hatch brush with red horizontal lines and yellow background.
*   It makes the background of the inner rectangle transparent or opaque in function of the user's input.
*   Inputs:
*       1. hdc, the display device context.
*       2. transparent, the hatch brush background user's value; true if transparent, false if opaque.
***/
VOID SetHatchBrushBackground(HDC hdc, bool transparent)
{
    // Define a brush handle.
    HBRUSH hBrush;

    // Create a solid blue brush.
    hBrush = CreateSolidBrush (RGB(0, 0, 255));

    // Associate the brush with the display device context.
    SelectObject (hdc, hBrush);

    // Draw a rectangle with blue background.
    Rectangle (hdc, 400,40,800,400);

    // Create a hatch brush that draws horizontal red lines.
    hBrush = CreateHatchBrush(HatchStyleHorizontal, RGB(255, 0, 0));

    // Set the background color to yellow.
    SetBkColor(hdc, RGB(255, 255, 0));

    // Select the hatch brush background transparency based on user's input.
    if (transparent == true)
        // Make the hatch brush background transparent.
        // This displays the outer rectangle blue background.
        SetBkMode(hdc, TRANSPARENT);
    else
        // Make the hatch brush background opaque.
        // This displays the inner rectangle yellow background.
        SetBkMode(hdc, OPAQUE);

    // Associate the hatch brush with the current device context.
    SelectObject(hdc, hBrush);

    // Draw a rectangle with the specified hatch brush.
    Rectangle(hdc, 500,130,700,300);
}

//
// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
// PURPOSE: Processes messages for the main window.
//
// WM_PAINT - Paint the main window
// WM_DESTROY - post a quit message and return

```

```

// This function handles all window messages.
//
// LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;
    TCHAR greeting[] = _T("Select your brush background.");
    TCHAR wmId;
    TCHAR wmEvent;

    switch (message)
    {
        case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);

            // Start application-specific layout section.
            // Just print the greeting string in the top left corner.
            TextOut(hdc,
                    5, 5,
                    greeting, (int)_tcslen(greeting));
            // End application-specific layout section.

            // Draw rectangles using hatch brush.
            SetHatchBrushBackground(hdc, true);

            EndPaint(hWnd, &ps);
            break;

        case WM_COMMAND:
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            hdc = GetDC(hWnd);

            switch (wmId) {

                case BTN_MYBUTTON_ID_1:
                    // Draw the inner rectangle using a hatch brush transparent background.
                    SetHatchBrushBackground(hdc, true);
                    MessageBox(hWnd, _T("Hatch brush background is TRANSPARENT"), _T("Information"), MB_OK);
                    break;

                case BTN_MYBUTTON_ID_2:
                    // Draw the inner rectangle using a hatch brush opaque background.
                    SetHatchBrushBackground(hdc, false);
                    MessageBox(hWnd, _T("Hatch brush background is OPAQUE"), _T("Information"), MB_OK);
                    break;
            }
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
            break;
    }

    return 0;
}

```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Brush Functions](#)

[Brushes Overview](#)

[COLORREF](#)

[CreateDIBPatternBrush](#)

[CreateDIBPatternBrushPt](#)

[CreatePatternBrush](#)

[CreateSolidBrush](#)

[DeleteObject](#)

[GetBrushOrgEx](#)

[RGB](#)

[SelectObject](#)

[SetBkMode](#)

[SetBrushOrgEx](#)

CreateICA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateIC** function creates an information context for the specified device. The information context provides a fast way to get information about the device without creating a device context (DC). However, GDI drawing functions cannot accept a handle to an information context.

Syntax

```
HDC CreateICA(
    [in] LPCSTR      pszDriver,
    [in] LPCSTR      pszDevice,
    [in] LPCSTR      pszPort,
    [in] const DEVMODEA *pdm
);
```

Parameters

[in] pszDriver

A pointer to a null-terminated character string that specifies the name of the device driver (for example, Epson).

[in] pszDevice

A pointer to a null-terminated character string that specifies the name of the specific output device being used, as shown by the Print Manager (for example, Epson FX-80). It is not the printer model name. The *lpSzDevice* parameter must be used.

pszPort

This parameter is ignored and should be set to **NULL**. It is provided only for compatibility with 16-bit Windows.

[in] pdm

A pointer to a **DEVMODE** structure containing device-specific initialization data for the device driver. The **DocumentProperties** function retrieves this structure filled in for a specified device. The *lpdvmInit* parameter must be **NULL** if the device driver is to use the default initialization (if any) specified by the user.

Return value

If the function succeeds, the return value is the handle to an information context.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the information DC, call the **DeleteDC** function.

NOTE

The wingdi.h header defines CreateIC as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DEVMODE](#)

[DeleteDC](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[DocumentProperties](#)

[GetDeviceCaps](#)

CreateICW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateIC** function creates an information context for the specified device. The information context provides a fast way to get information about the device without creating a device context (DC). However, GDI drawing functions cannot accept a handle to an information context.

Syntax

```
HDC CreateICW(
    [in] LPCWSTR      pszDriver,
    [in] LPCWSTR      pszDevice,
    [in] LPCWSTR      pszPort,
    [in] const DEVMODEW *pdm
);
```

Parameters

[in] `pszDriver`

A pointer to a null-terminated character string that specifies the name of the device driver (for example, Epson).

[in] `pszDevice`

A pointer to a null-terminated character string that specifies the name of the specific output device being used, as shown by the Print Manager (for example, Epson FX-80). It is not the printer model name. The *lpszDevice* parameter must be used.

`pszPort`

This parameter is ignored and should be set to **NULL**. It is provided only for compatibility with 16-bit Windows.

[in] `pdm`

A pointer to a **DEVMODE** structure containing device-specific initialization data for the device driver. The **DocumentProperties** function retrieves this structure filled in for a specified device. The *lpdvmInit* parameter must be **NULL** if the device driver is to use the default initialization (if any) specified by the user.

Return value

If the function succeeds, the return value is the handle to an information context.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the information DC, call the **DeleteDC** function.

NOTE

The wingdi.h header defines CreateIC as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DEVMODE](#)

[DeleteDC](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[DocumentProperties](#)

[GetDeviceCaps](#)

CreateMetaFileA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateMetaFile** function creates a device context for a Windows-format metafile.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [CreateEnhMetaFile](#).

Syntax

```
HDC CreateMetaFileA(
    [in] LPCSTR pszFile
);
```

Parameters

[in] **pszFile**

A pointer to the file name for the Windows-format metafile to be created. If this parameter is **NULL**, the Windows-format metafile is memory based and its contents are lost when it is deleted by using the [DeleteMetaFile](#) function.

Return value

If the function succeeds, the return value is a handle to the device context for the Windows-format metafile.

If the function fails, the return value is **NULL**.

Remarks

Where text arguments must use Unicode characters, use the **CreateMetaFile** function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

CreateMetaFile is a Windows-format metafile function. This function supports only 16-bit Windows-based applications, which are listed in [Windows-Format Metafiles](#). It does not record or play back GDI functions such as [PolyBezier](#), which were not part of 16-bit Windows.

The device context created by this function can be used to record GDI output functions in a Windows-format metafile. It cannot be used with GDI query functions such as [GetTextColor](#). When the device context is used with a GDI output function, the return value of that function becomes **TRUE** if the function is recorded and **FALSE** otherwise. When an object is selected by using the [SelectObject](#) function, only a copy of the object is recorded. The object still belongs to the application.

To create a scalable Windows-format metafile, record the graphics output in the **MM_ANISOTROPIC** mapping mode. The file cannot contain functions that modify the viewport origin and extents, nor can it contain device-dependent functions such as the [SelectClipRgn](#) function. Once created, the Windows metafile can be scaled and

rendered to any output device-format by defining the viewport origin and extents of the picture before playing it.

NOTE

The wingdi.h header defines CreateMetaFile as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CloseMetaFile](#)

[CreateEnhMetaFile](#)

[DeleteMetaFile](#)

[GetTextColor](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SelectClipRgn](#)

[SelectObject](#)

CreateMetaFileW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateMetaFile** function creates a device context for a Windows-format metafile.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [CreateEnhMetaFile](#).

Syntax

```
HDC CreateMetaFileW(
    [in] LPCWSTR pszFile
);
```

Parameters

[in] **pszFile**

A pointer to the file name for the Windows-format metafile to be created. If this parameter is **NULL**, the Windows-format metafile is memory based and its contents are lost when it is deleted by using the [DeleteMetaFile](#) function.

Return value

If the function succeeds, the return value is a handle to the device context for the Windows-format metafile.

If the function fails, the return value is **NULL**.

Remarks

Where text arguments must use Unicode characters, use the **CreateMetaFile** function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

CreateMetaFile is a Windows-format metafile function. This function supports only 16-bit Windows-based applications, which are listed in [Windows-Format Metafiles](#). It does not record or play back GDI functions such as [PolyBezier](#), which were not part of 16-bit Windows.

The device context created by this function can be used to record GDI output functions in a Windows-format metafile. It cannot be used with GDI query functions such as [GetTextColor](#). When the device context is used with a GDI output function, the return value of that function becomes **TRUE** if the function is recorded and **FALSE** otherwise. When an object is selected by using the [SelectObject](#) function, only a copy of the object is recorded. The object still belongs to the application.

To create a scalable Windows-format metafile, record the graphics output in the **MM_ANISOTROPIC** mapping mode. The file cannot contain functions that modify the viewport origin and extents, nor can it contain device-dependent functions such as the [SelectClipRgn](#) function. Once created, the Windows metafile can be scaled and

rendered to any output device-format by defining the viewport origin and extents of the picture before playing it.

NOTE

The wingdi.h header defines CreateMetaFile as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CloseMetaFile](#)

[CreateEnhMetaFile](#)

[DeleteMetaFile](#)

[GetTextColor](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SelectClipRgn](#)

[SelectObject](#)

CreatePalette function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreatePalette** function creates a logical palette.

Syntax

```
HPALETTE CreatePalette(
    [in] const LOGPALETTE *plpal
);
```

Parameters

[in] plpal

A pointer to a [LOGPALETTE](#) structure that contains information about the colors in the logical palette.

Return value

If the function succeeds, the return value is a handle to a logical palette.

If the function fails, the return value is **NULL**.

Remarks

An application can determine whether a device supports palette operations by calling the [GetDeviceCaps](#) function and specifying the **RASTERCAPS** constant.

Once an application creates a logical palette, it can select that palette into a device context by calling the [SelectPalette](#) function. A palette selected into a device context can be realized by calling the [RealizePalette](#) function.

When you no longer need the palette, call the [DeleteObject](#) function to delete it.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[DeleteObject](#)

[GetDeviceCaps](#)

[LOGPALETTE](#)

[RealizePalette](#)

[SelectPalette](#)

CreatePatternBrush function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreatePatternBrush** function creates a logical brush with the specified bitmap pattern. The bitmap can be a DIB section bitmap, which is created by the **CreateDIBSection** function, or it can be a device-dependent bitmap.

Syntax

```
HBRUSH CreatePatternBrush(  
    [in] HBITMAP hbm  
)
```

Parameters

[in] `hbm`

A handle to the bitmap to be used to create the logical brush.

Return value

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is **NULL**.

Remarks

A pattern brush is a bitmap that the system uses to paint the interiors of filled shapes.

After an application creates a brush by calling **CreatePatternBrush**, it can select that brush into any device context by calling the **SelectObject** function.

You can delete a pattern brush without affecting the associated bitmap by using the **DeleteObject** function. Therefore, you can then use this bitmap to create any number of pattern brushes.

A brush created by using a monochrome (1 bit per pixel) bitmap has the text and background colors of the device context to which it is drawn. Pixels represented by a 0 bit are drawn with the current text color; pixels represented by a 1 bit are drawn with the current background color.

ICM: No color is done at brush creation. However, color management is performed when the brush is selected into an ICM-enabled device context.

Examples

For an example, see [Using Brushes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Brush Functions](#)

[Brushes Overview](#)

[CreateBitmap](#)

[CreateBitmapIndirect](#)

[CreateCompatibleBitmap](#)

[CreateDIBPatternBrush](#)

[CreateDIBPatternBrushPt](#)

[CreateDIBSection](#)

[CreateHatchBrush](#)

[DeleteObject](#)

[GetBrushOrgEx](#)

[LoadBitmap](#)

[SelectObject](#)

[SetBrushOrgEx](#)

CreatePen function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **CreatePen** function creates a logical pen that has the specified style, width, and color. The pen can subsequently be selected into a device context and used to draw lines and curves.

Syntax

```
HPEN CreatePen(
    [in] int      iStyle,
    [in] int      cWidth,
    [in] COLORREF color
);
```

Parameters

[in] iStyle

The pen style. It can be any one of the following values.

VALUE	MEANING
PS_SOLID	The pen is solid.
PS_DASH	The pen is dashed. This style is valid only when the pen width is one or less in device units.
PS_DOT	The pen is dotted. This style is valid only when the pen width is one or less in device units.
PS_DASHDOT	The pen has alternating dashes and dots. This style is valid only when the pen width is one or less in device units.
PS_DASHDOTDOT	The pen has alternating dashes and double dots. This style is valid only when the pen width is one or less in device units.
PS_NULL	The pen is invisible.
PS_INSIDEFRAME	The pen is solid. When this pen is used in any GDI drawing function that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to geometric pens.

[in] cWidth

The width of the pen, in logical units. If *nWidth* is zero, the pen is a single pixel wide, regardless of the current transformation.

CreatePen returns a pen with the specified width but with the PS_SOLID style if you specify a width greater than one for the following styles: PS_DASH, PS_DOT, PS_DASHDOT, PS_DASHDOTDOT.

[in] color

A color reference for the pen color. To generate a [COLORREF](#) structure, use the [RGB](#) macro.

Return value

If the function succeeds, the return value is a handle that identifies a logical pen.

If the function fails, the return value is **NULL**.

Remarks

After an application creates a logical pen, it can select that pen into a device context by calling the [SelectObject](#) function. After a pen is selected into a device context, it can be used to draw lines and curves.

If the value specified by the *nWidth* parameter is zero, a line drawn with the created pen always is a single pixel wide regardless of the current transformation.

If the value specified by *nWidth* is greater than 1, the *fnPenStyle* parameter must be PS_NULL, PS_SOLID, or PS_INSIDEFRAME.

If the value specified by *nWidth* is greater than 1 and *fnPenStyle* is PS_INSIDEFRAME, the line associated with the pen is drawn inside the frame of all primitives except polygons and polylines.

If the value specified by *nWidth* is greater than 1, *fnPenStyle* is PS_INSIDEFRAME, and the color specified by the *crColor* parameter does not match one of the entries in the logical palette, the system draws lines by using a dithered color. Dithered colors are not available with solid pens.

When using an *iStyle* parameter of PS_DASH, PS_DOT, PS_DASHDOT or PS_DASHDOTDOT, in order to make the gaps between the dashes or dots transparent, use [SetBkMode](#) to set the mode to TRANSPARENT.

When you no longer need the pen, call the [DeleteObject](#) function to delete it.

ICM: No color management is done at creation. However, color management is performed when the pen is selected into an ICM-enabled device context.

Examples

For an example, see [Creating Colored Pens and Brushes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORREF](#)

[CreatePenIndirect](#)

[DeleteObject](#)

[ExtCreatePen](#)

[GetObject](#)

[Pen Functions](#)

[Pens Overview](#)

[RGB](#)

[SelectObject](#)

CreatePenIndirect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreatePenIndirect** function creates a logical cosmetic pen that has the style, width, and color specified in a structure.

Syntax

```
HPEN CreatePenIndirect(
    [in] const LOGPEN *plpen
);
```

Parameters

[in] `plpen`

Pointer to a [LOGPEN](#) structure that specifies the pen's style, width, and color.

Return value

If the function succeeds, the return value is a handle that identifies a logical cosmetic pen.

If the function fails, the return value is **NULL**.

Remarks

After an application creates a logical pen, it can select that pen into a device context by calling the [SelectObject](#) function. After a pen is selected into a device context, it can be used to draw lines and curves.

When you no longer need the pen, call the [DeleteObject](#) function to delete it.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePen](#)

[DeleteObject](#)

[ExtCreatePen](#)

[GetObject](#)

[LOGPEN](#)

[Pen Functions](#)

[Pens Overview](#)

[RGB](#)

[SelectObject](#)

CreatePolygonRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreatePolygonRgn** function creates a polygonal region.

Syntax

```
HRGN CreatePolygonRgn(
    [in] const POINT *pptl,
    [in] int         cPoint,
    [in] int         iMode
);
```

Parameters

[in] **pptl**

A pointer to an array of **POINT** structures that define the vertices of the polygon in logical units. The polygon is presumed closed. Each vertex can be specified only once.

[in] **cPoint**

The number of points in the array.

[in] **iMode**

The fill mode used to determine which pixels are in the region. This parameter can be one of the following values.

VALUE	MEANING
ALTERNATE	Selects alternate mode (fills area between odd-numbered and even-numbered polygon sides on each scan line).
WINDING	Selects winding mode (fills any region with a nonzero winding value).

For more information about these modes, see the [SetPolyFillMode](#) function.

Return value

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the **HRGN** object, call the [DeleteObject](#) function to delete it.

Region coordinates are represented as 27-bit signed integers.

Regions created by the Create<shape>Rgn methods (such as [CreateRectRgn](#) and [CreatePolygonRgn](#)) only include the interior of the shape; the shape's outline is excluded from the region. This means that any point on a line between two sequential vertices is not included in the region. If you were to call [PtInRegion](#) for such a point, it would return zero as the result.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePolyPolygonRgn](#)

[CreateRectRgn](#)

[CreateRectRgnIndirect](#)

[CreateRoundRectRgn](#)

[DeleteObject](#)

[ExtCreateRegion](#)

[GetRegionData](#)

[POINT](#)

[Region Functions](#)

[Regions Overview](#)

[SelectObject](#)

[SetPolyFillMode](#)

CreatePolyPolygonRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreatePolyPolygonRgn** function creates a region consisting of a series of polygons. The polygons can overlap.

Syntax

```
HRGN CreatePolyPolygonRgn(
    [in] const POINT *pptl,
    [in] const INT    *pc,
    [in] int         cPoly,
    [in] int         iMode
);
```

Parameters

[in] *pptr*

A pointer to an array of **POINT** structures that define the vertices of the polygons in logical units. The polygons are specified consecutively. Each polygon is presumed closed and each vertex is specified only once.

[in] *pc*

A pointer to an array of integers, each of which specifies the number of points in one of the polygons in the array pointed to by *lppt*.

[in] *cPoly*

The total number of integers in the array pointed to by *lpPolyCounts*.

[in] *iMode*

The fill mode used to determine which pixels are in the region. This parameter can be one of the following values.

VALUE	MEANING
ALTERNATE	Selects alternate mode (fills area between odd-numbered and even-numbered polygon sides on each scan line).
WINDING	Selects winding mode (fills any region with a nonzero winding value).

For more information about these modes, see the [SetPolyFillMode](#) function.

Return value

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is zero.

Remarks

When you no longer need the **HRGN** object, call the [DeleteObject](#) function to delete it.

Region coordinates are represented as 27-bit signed integers.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePolygonRgn](#)

[CreateRectRgn](#)

[CreateRectRgnIndirect](#)

[CreateRoundRectRgn](#)

[DeleteObject](#)

[ExtCreateRegion](#)

[GetRegionData](#)

[POINT](#)

[Region Functions](#)

[Regions Overview](#)

[SelectObject](#)

[SetPolyFillMode](#)

CreateRectRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateRectRgn** function creates a rectangular region.

Syntax

```
HRGN CreateRectRgn(
    [in] int x1,
    [in] int y1,
    [in] int x2,
    [in] int y2
);
```

Parameters

[in] **x1**

Specifies the x-coordinate of the upper-left corner of the region in logical units.

[in] **y1**

Specifies the y-coordinate of the upper-left corner of the region in logical units.

[in] **x2**

Specifies the x-coordinate of the lower-right corner of the region in logical units.

[in] **y2**

Specifies the y-coordinate of the lower-right corner of the region in logical units.

Return value

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the **HRGN** object, call the [DeleteObject](#) function to delete it.

Region coordinates are represented as 27-bit signed integers.

Regions created by the Create<shape>Rgn methods (such as [CreateRectRgn](#) and [CreatePolygonRgn](#)) only include the interior of the shape; the shape's outline is excluded from the region. This means that any point on a line between two sequential vertices is not included in the region. If you were to call [PtInRegion](#) for such a point, it would return zero as the result.

Examples

For an example, see [Drawing Markers](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePolyPolygonRgn](#)

[CreatePolygonRgn](#)

[CreateRectRgnIndirect](#)

[CreateRoundRectRgn](#)

[DeleteObject](#)

[ExtCreateRegion](#)

[GetRegionData](#)

[Region Functions](#)

[Regions Overview](#)

[SelectObject](#)

CreateRectRgnIndirect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateRectRgnIndirect** function creates a rectangular region.

Syntax

```
HRGN CreateRectRgnIndirect(  
    [in] const RECT *lprect  
)
```

Parameters

[in] lprect

Pointer to a [RECT](#) structure that contains the coordinates of the upper-left and lower-right corners of the rectangle that defines the region in logical units.

Return value

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the **HRGN** object, call the [DeleteObject](#) function to delete it.

Region coordinates are represented as 27-bit signed integers.

The region will be exclusive of the bottom and right edges.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePolyPolygonRgn](#)

[CreatePolygonRgn](#)

[CreateRectRgn](#)

[CreateRoundRectRgn](#)

[DeleteObject](#)

[ExtCreateRegion](#)

[GetRegionData](#)

[RECT](#)

[Region Functions](#)

[Regions Overview](#)

[SelectObject](#)

CreateRoundRectRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateRoundRectRgn** function creates a rectangular region with rounded corners.

Syntax

```
HRGN CreateRoundRectRgn(
    [in] int x1,
    [in] int y1,
    [in] int x2,
    [in] int y2,
    [in] int w,
    [in] int h
);
```

Parameters

[in] **x1**

Specifies the x-coordinate of the upper-left corner of the region in device units.

[in] **y1**

Specifies the y-coordinate of the upper-left corner of the region in device units.

[in] **x2**

Specifies the x-coordinate of the lower-right corner of the region in device units.

[in] **y2**

Specifies the y-coordinate of the lower-right corner of the region in device units.

[in] **w**

Specifies the width of the ellipse used to create the rounded corners in device units.

[in] **h**

Specifies the height of the ellipse used to create the rounded corners in device units.

Return value

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is **NULL**.

Remarks

When you no longer need the **HRGN** object call the [DeleteObject](#) function to delete it.

Region coordinates are represented as 27-bit signed integers.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePolyPolygonRgn](#)

[CreatePolygonRgn](#)

[CreateRectRgn](#)

[CreateRectRgnIndirect](#)

[DeleteObject](#)

[ExtCreateRegion](#)

[GetRegionData](#)

[Region Functions](#)

[Regions Overview](#)

[SelectObject](#)

CreateScalableFontResourceA function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

[The **CreateScalableFontResource** function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions.]

The **CreateScalableFontResource** function creates a font resource file for a scalable font.

Syntax

```
BOOL CreateScalableFontResourceA(
    [in] DWORD  fdwHidden,
    [in] LPCSTR lpszFont,
    [in] LPCSTR lpszFile,
    [in] LPCSTR lpszPath
);
```

Parameters

[in] fdwHidden

Specifies whether the font is a read-only font. This parameter can be one of the following values.

VALUE	MEANING
0	The font has read/write permission.
1	The font has read-only permission and should be hidden from other applications in the system. When this flag is set, the font is not enumerated by the EnumFonts or EnumFontFamilies function.

[in] lpszFont

A pointer to a null-terminated string specifying the name of the font resource file to create. If this parameter specifies an existing font resource file, the function fails.

[in] lpszFile

A pointer to a null-terminated string specifying the name of the scalable font file that this function uses to create the font resource file.

[in] lpszPath

A pointer to a null-terminated string specifying the path to the scalable font file.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

If *lpszFontRes* specifies an existing font file, [GetLastError](#) returns ERROR_FILE_EXISTS

Remarks

The **CreateScalableFontResource** function is used by applications that install TrueType fonts. An application uses the **CreateScalableFontResource** function to create a font resource file (typically with a .fot file name extension) and then uses the [AddFontResource](#) function to install the font. The TrueType font file (typically with a .ttf file name extension) must be in the System subdirectory of the Windows directory to be used by the [AddFontResource](#) function.

The **CreateScalableFontResource** function currently supports only TrueType-technology scalable fonts.

When the *lpszFontFile* parameter specifies only a file name and extension, the *lpszCurrentPath* parameter must specify a path. When the *lpszFontFile* parameter specifies a full path, the *lpszCurrentPath* parameter must be NULL or a pointer to NULL.

When only a file name and extension are specified in the *lpszFontFile* parameter and a path is specified in the *lpszCurrentPath* parameter, the string in *lpszFontFile* is copied into the .fot file as the .ttf file that belongs to this resource. When the [AddFontResource](#) function is called, the operating system assumes that the .ttf file has been copied into the System directory (or into the main Windows directory in the case of a network installation). The .ttf file need not be in this directory when the **CreateScalableFontResource** function is called, because the *lpszCurrentPath* parameter contains the directory information. A resource created in this manner does not contain absolute path information and can be used in any installation.

When a path is specified in the *lpszFontFile* parameter and NULL is specified in the *lpszCurrentPath* parameter, the string in *lpszFontFile* is copied into the .fot file. In this case, when the [AddFontResource](#) function is called, the .ttf file must be at the location specified in the *lpszFontFile* parameter when the **CreateScalableFontResource** function was called; the *lpszCurrentPath* parameter is not needed. A resource created in this manner contains absolute references to paths and drives and does not work if the .ttf file is moved to a different location.

NOTE

The wingdi.h header defines CreateScalableFontResource as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AddFontResource](#)

[EnumFontFamilies](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

CreateScalableFontResourceW function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

[The **CreateScalableFontResource** function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions.]

The **CreateScalableFontResource** function creates a font resource file for a scalable font.

Syntax

```
BOOL CreateScalableFontResourceW(
    [in] DWORD    fdwHidden,
    [in] LPCWSTR lpszFont,
    [in] LPCWSTR lpszFile,
    [in] LPCWSTR lpszPath
);
```

Parameters

[in] fdwHidden

Specifies whether the font is a read-only font. This parameter can be one of the following values.

VALUE	MEANING
0	The font has read/write permission.
1	The font has read-only permission and should be hidden from other applications in the system. When this flag is set, the font is not enumerated by the EnumFonts or EnumFontFamilies function.

[in] lpszFont

A pointer to a null-terminated string specifying the name of the font resource file to create. If this parameter specifies an existing font resource file, the function fails.

[in] lpszFile

A pointer to a null-terminated string specifying the name of the scalable font file that this function uses to create the font resource file.

[in] lpszPath

A pointer to a null-terminated string specifying the path to the scalable font file.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

If *lpszFontRes* specifies an existing font file, [GetLastError](#) returns ERROR_FILE_EXISTS

Remarks

The **CreateScalableFontResource** function is used by applications that install TrueType fonts. An application uses the **CreateScalableFontResource** function to create a font resource file (typically with a .fot file name extension) and then uses the [AddFontResource](#) function to install the font. The TrueType font file (typically with a .ttf file name extension) must be in the System subdirectory of the Windows directory to be used by the [AddFontResource](#) function.

The **CreateScalableFontResource** function currently supports only TrueType-technology scalable fonts.

When the *lpszFontFile* parameter specifies only a file name and extension, the *lpszCurrentPath* parameter must specify a path. When the *lpszFontFile* parameter specifies a full path, the *lpszCurrentPath* parameter must be NULL or a pointer to NULL.

When only a file name and extension are specified in the *lpszFontFile* parameter and a path is specified in the *lpszCurrentPath* parameter, the string in *lpszFontFile* is copied into the .fot file as the .ttf file that belongs to this resource. When the [AddFontResource](#) function is called, the operating system assumes that the .ttf file has been copied into the System directory (or into the main Windows directory in the case of a network installation). The .ttf file need not be in this directory when the **CreateScalableFontResource** function is called, because the *lpszCurrentPath* parameter contains the directory information. A resource created in this manner does not contain absolute path information and can be used in any installation.

When a path is specified in the *lpszFontFile* parameter and NULL is specified in the *lpszCurrentPath* parameter, the string in *lpszFontFile* is copied into the .fot file. In this case, when the [AddFontResource](#) function is called, the .ttf file must be at the location specified in the *lpszFontFile* parameter when the **CreateScalableFontResource** function was called; the *lpszCurrentPath* parameter is not needed. A resource created in this manner contains absolute references to paths and drives and does not work if the .ttf file is moved to a different location.

NOTE

The wingdi.h header defines CreateScalableFontResource as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AddFontResource](#)

[EnumFontFamilies](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

CreateSolidBrush function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CreateSolidBrush** function creates a logical brush that has the specified solid color.

Syntax

```
HBRUSH CreateSolidBrush(  
    [in] COLORREF color  
)
```

Parameters

[in] `color`

The color of the brush. To create a `COLORREF` color value, use the `RGB` macro.

Return value

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is `NULL`.

Remarks

When you no longer need the `HBRUSH` object, call the `DeleteObject` function to delete it.

A solid brush is a bitmap that the system uses to paint the interiors of filled shapes.

After an application creates a brush by calling `CreateSolidBrush`, it can select that brush into any device context by calling the `SelectObject` function.

To paint with a system color brush, an application should use `GetSysColorBrush (nIndex)` instead of `CreateSolidBrush(GetSysColor(nIndex))`, because `GetSysColorBrush` returns a cached brush instead of allocating a new one.

ICM: No color management is done at brush creation. However, color management is performed when the brush is selected into an ICM-enabled device context.

Examples

For an example, see [Creating Colored Pens and Brushes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Brush Functions](#)

[Brushes Overview](#)

[COLORREF](#)

[CreateDIBPatternBrush](#)

[CreateDIBPatternBrushPt](#)

[CreateHatchBrush](#)

[CreatePatternBrush](#)

[DeleteObject](#)

[GetSysColorBrush](#)

[RGB](#)

[SelectObject](#)

DeleteDC function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DeleteDC** function deletes the specified device context (DC).

Syntax

```
BOOL DeleteDC(  
    [in] HDC hdc  
>;
```

Parameters

`[in] hdc`

A handle to the device context.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

An application must not delete a DC whose handle was obtained by calling the [GetDC](#) function. Instead, it must call the [ReleaseDC](#) function to free the DC.

Examples

For an example, see [Retrieving the Capabilities of a Printer](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateDC](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetDC](#)

[ReleaseDC](#)

DeleteEnhMetaFile function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DeleteEnhMetaFile** function deletes an enhanced-format metafile or an enhanced-format metafile handle.

Syntax

```
BOOL DeleteEnhMetaFile(
    [in] HENHMETAFILE hmf
);
```

Parameters

[in] **hmf**

A handle to an enhanced metafile.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

If the *hmf* parameter identifies an enhanced metafile stored in memory, the **DeleteEnhMetaFile** function deletes the metafile. If *hmf* identifies a metafile stored on a disk, the function deletes the metafile handle but does not destroy the actual metafile. An application can retrieve the file by calling the [GetEnhMetaFile](#) function.

Examples

For an example, see [Opening an Enhanced Metafile and Displaying Its Contents](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CopyEnhMetaFile](#)

[CreateEnhMetaFile](#)

[GetEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

DeleteMetaFile function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DeleteMetaFile** function deletes a Windows-format metafile or Windows-format metafile handle.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [DeleteEnhMetaFile](#).

Syntax

```
BOOL DeleteMetaFile(  
    [in] HMETAFILE hmf  
)
```

Parameters

[in] *hmf*

A handle to a Windows-format metafile.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

If the metafile identified by the *hmf* parameter is stored in memory (rather than on a disk), its content is lost when it is deleted by using the **DeleteMetaFile** function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CopyMetaFile](#)

[CreateMetaFile](#)

[DeleteEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

DeleteObject function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DeleteObject** function deletes a logical pen, brush, font, bitmap, region, or palette, freeing all system resources associated with the object. After the object is deleted, the specified handle is no longer valid.

Syntax

```
BOOL DeleteObject(  
    [in] HGDIOBJ ho  
);
```

Parameters

[in] *ho*

A handle to a logical pen, brush, font, bitmap, region, or palette.

Return value

If the function succeeds, the return value is nonzero.

If the specified handle is not valid or is currently selected into a DC, the return value is zero.

Remarks

Do not delete a drawing object (pen or brush) while it is still selected into a DC.

When a pattern brush is deleted, the bitmap associated with the brush is not deleted. The bitmap must be deleted independently.

Examples

For an example, see [Creating Colored Pens and Brushes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Device Context Functions](#)

[Device Contexts Overview](#)

[SelectObject](#)

DESIGNVECTOR structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The DESIGNVECTOR structure is used by an application to specify values for the axes of a multiple master font.

Syntax

```
typedef struct tagDESIGNVECTOR {  
    DWORD dvReserved;  
    DWORD dvNumAxes;  
    LONG  dvValues[MM_MAX_NUMAXES];  
} DESIGNVECTOR, *PDESIGNVECTOR, *LPDESIGNVECTOR;
```

Members

dvReserved

Reserved. Must be STAMP_DESIGNVECTOR.

dvNumAxes

Number of values in the **dvValues** array.

dvValues

An array specifying the values of the axes of a multiple master OpenType font. This array corresponds to the **axlAxisInfo** array in the [AXESLIST](#) structure.

Remarks

The **dvNumAxes** member determines the actual size of **dvValues**, and thus, of DESIGNVECTOR. The constant MM_MAX_NUMAXES, which is 16, specifies the maximum allowed size of the **dvValues** array.

The PostScript Open Type Font does not support multiple master functionality.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[AXESLIST](#)

[AddFontMemResourceEx](#)

[AddFontResourceEx](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[RemoveFontResourceEx](#)

DIBSECTION structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DIBSECTION** structure contains information about a DIB created by calling the [CreateDIBSection](#) function. A **DIBSECTION** structure includes information about the bitmap's dimensions, color format, color masks, optional file mapping object, and optional bit values storage offset. An application can obtain a filled-in **DIBSECTION** structure for a given DIB by calling the [GetObject](#) function.

Syntax

```
typedef struct tagDIBSECTION {  
    BITMAP          dsBm;  
    BITMAPINFOHEADER dsBmih;  
    DWORD           dsBitfields[3];  
    HANDLE          dshSection;  
    DWORD           dsOffset;  
} DIBSECTION, *LPDIBSECTION, *PDIBSECTION;
```

Members

dsBm

A [BITMAP](#) data structure that contains information about the DIB: its type, its dimensions, its color capacities, and a pointer to its bit values.

dsBmih

A [BITMAPINFOHEADER](#) structure that contains information about the color format of the DIB.

dsBitfields

Specifies three color masks for the DIB. This field is only valid when the **BitCount** member of the [BITMAPINFOHEADER](#) structure has a value greater than 8. Each color mask indicates the bits that are used to encode one of the three color channels (red, green, and blue).

dshSection

Contains a handle to the file mapping object that the [CreateDIBSection](#) function used to create the DIB. If [CreateDIBSection](#) was called with a **NULL** value for its *hSection* parameter, causing the system to allocate memory for the bitmap, the *dshSection* member will be **NULL**.

dsOffset

The offset to the bitmap's bit values within the file mapping object referenced by *dshSection*. If *dshSection* is **NULL**, the *dsOffset* value has no meaning.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAP](#)

[BITMAPINFOHEADER](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

[CreateDIBSection](#)

[GetDIBColorTable](#)

[GetObject](#)

DISPLAY_DEVICEA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DISPLAY_DEVICE** structure receives information about the display device specified by the *iDevNum* parameter of the [EnumDisplayDevices](#) function.

Syntax

```
typedef struct _DISPLAY_DEVICEA {
    DWORD cb;
    CHAR DeviceName[32];
    CHAR DeviceString[128];
    DWORD StateFlags;
    CHAR DeviceID[128];
    CHAR DeviceKey[128];
} DISPLAY_DEVICEA, *PDISPLAY_DEVICEA, *LPPDISPLAY_DEVICEA;
```

Members

cb

Size, in bytes, of the **DISPLAY_DEVICE** structure. This must be initialized prior to calling [EnumDisplayDevices](#).

DeviceName

An array of characters identifying the device name. This is either the adapter device or the monitor device.

DeviceString

An array of characters containing the device context string. This is either a description of the display adapter or of the display monitor.

StateFlags

Device state flags. It can be any reasonable combination of the following.

VALUE	MEANING
DISPLAY_DEVICE_ACTIVE	DISPLAY_DEVICE_ACTIVE specifies whether a monitor is presented as being "on" by the respective GDI view. Windows Vista: <code>EnumDisplayDevices</code> will only enumerate monitors that can be presented as being "on."
DISPLAY_DEVICE_MIRRORING_DRIVER	Represents a pseudo device used to mirror application drawing for remoting or other purposes. An invisible pseudo monitor is associated with this device. For example, NetMeeting uses it. Note that GetSystemMetrics (<code>SM_MONITORS</code>) only accounts for visible display monitors.
DISPLAY_DEVICE_MODESPRUNED	The device has more display modes than its output devices support.

DISPLAY_DEVICE_PRIMARY_DEVICE	The primary desktop is on the device. For a system with a single display card, this is always set. For a system with multiple display cards, only one device can have this set.
DISPLAY_DEVICE_REMOVABLE	The device is removable; it cannot be the primary display.
DISPLAY_DEVICE_VGA_COMPATIBLE	The device is VGA compatible.

DeviceID

Not used.

DeviceKey

Reserved.

Remarks

The four string members are set based on the parameters passed to [EnumDisplayDevices](#). If the *lpDevice* param is NULL, then DISPLAY_DEVICE is filled in with information about the display adapter(s). If it is a valid device name, then it is filled in with information about the monitor(s) for that device.

NOTE

The wingdi.h header defines DISPLAY_DEVICE as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Device Context Structures](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

[GetSystemMetrics](#)

DISPLAY_DEVICEW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DISPLAY_DEVICE** structure receives information about the display device specified by the *iDevNum* parameter of the [EnumDisplayDevices](#) function.

Syntax

```
typedef struct _DISPLAY_DEVICEW {
    DWORD cb;
    WCHAR DeviceName[32];
    WCHAR DeviceString[128];
    DWORD StateFlags;
    WCHAR DeviceID[128];
    WCHAR DeviceKey[128];
} DISPLAY_DEVICEW, *PDISPLAY_DEVICEW, *LPPDISPLAY_DEVICEW;
```

Members

cb

Size, in bytes, of the **DISPLAY_DEVICE** structure. This must be initialized prior to calling [EnumDisplayDevices](#).

DeviceName

An array of characters identifying the device name. This is either the adapter device or the monitor device.

DeviceString

An array of characters containing the device context string. This is either a description of the display adapter or of the display monitor.

StateFlags

Device state flags. It can be any reasonable combination of the following.

VALUE	MEANING
DISPLAY_DEVICE_ACTIVE	DISPLAY_DEVICE_ACTIVE specifies whether a monitor is presented as being "on" by the respective GDI view. Windows Vista: <code>EnumDisplayDevices</code> will only enumerate monitors that can be presented as being "on."
DISPLAY_DEVICE_MIRRORING_DRIVER	Represents a pseudo device used to mirror application drawing for remoting or other purposes. An invisible pseudo monitor is associated with this device. For example, NetMeeting uses it. Note that GetSystemMetrics (<code>SM_MONITORS</code>) only accounts for visible display monitors.
DISPLAY_DEVICE_MODESPRUNED	The device has more display modes than its output devices support.

DISPLAY_DEVICE_PRIMARY_DEVICE	The primary desktop is on the device. For a system with a single display card, this is always set. For a system with multiple display cards, only one device can have this set.
DISPLAY_DEVICE_REMOVABLE	The device is removable; it cannot be the primary display.
DISPLAY_DEVICE_VGA_COMPATIBLE	The device is VGA compatible.

DeviceID

Not used.

DeviceKey

Reserved.

Remarks

The four string members are set based on the parameters passed to [EnumDisplayDevices](#). If the *lpDevice* param is NULL, then DISPLAY_DEVICE is filled in with information about the display adapter(s). If it is a valid device name, then it is filled in with information about the monitor(s) for that device.

NOTE

The wingdi.h header defines DISPLAY_DEVICE as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Device Context Structures](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

[GetSystemMetrics](#)

DPtoLP function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DPtoLP** function converts device coordinates into logical coordinates. The conversion depends on the mapping mode of the device context, the settings of the origins and extents for the window and viewport, and the world transformation.

Syntax

```
BOOL DPtoLP(
    [in]      HDC      hdc,
    [in, out] LPPOINT lppt,
    [in]      int      c
);
```

Parameters

[in] hdc

A handle to the device context.

[in, out] lppt

A pointer to an array of **POINT** structures. The x- and y-coordinates contained in each **POINT** structure will be transformed.

[in] c

The number of points in the array.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **DPtoLP** function fails if the device coordinates exceed 27 bits, or if the converted logical coordinates exceed 32 bits. In the case of such an overflow, the results for all the points are undefined.

Examples

For an example, see [Using Coordinate Spaces and Transformations](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[LPtoDP](#)

[POINT](#)

DrawEscape function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DrawEscape** function provides drawing capabilities of the specified video display that are not directly available through the graphics device interface (GDI).

Syntax

```
int DrawEscape(
    [in] HDC     hdc,
    [in] int     iEscape,
    [in] int     cjIn,
    [in] LPCSTR lpIn
);
```

Parameters

[in] `hdc`

A handle to the DC for the specified video display.

[in] `iEscape`

The escape function to be performed.

[in] `cjIn`

The number of bytes of data pointed to by the `/pszInData` parameter.

[in] `lpIn`

A pointer to the input structure required for the specified escape.

Return value

If the function is successful, the return value is greater than zero except for the QUERYESCSUPPORT draw escape, which checks for implementation only.

If the escape is not implemented, the return value is zero.

If an error occurred, the return value is less than zero.

Remarks

When an application calls the **DrawEscape** function, the data identified by `cbInInput` and `/pszInData` is passed directly to the specified display driver.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Device Context Functions](#)

[Device Contexts Overview](#)

Ellipse function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **Ellipse** function draws an ellipse. The center of the ellipse is the center of the specified bounding rectangle. The ellipse is outlined by using the current pen and is filled by using the current brush.

Syntax

```
BOOL Ellipse(
    [in] HDC hdc,
    [in] int left,
    [in] int top,
    [in] int right,
    [in] int bottom
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `left`

The x-coordinate, in logical coordinates, of the upper-left corner of the bounding rectangle.

[in] `top`

The y-coordinate, in logical coordinates, of the upper-left corner of the bounding rectangle.

[in] `right`

The x-coordinate, in logical coordinates, of the lower-right corner of the bounding rectangle.

[in] `bottom`

The y-coordinate, in logical coordinates, of the lower-right corner of the bounding rectangle.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The current position is neither used nor updated by **Ellipse**.

Examples

For an example, see [Using Filled Shapes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Arc](#)

[ArcTo](#)

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

EMR structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMR** structure provides the base structure for all enhanced metafile records. An enhanced metafile record contains the parameters for a specific GDI function used to create part of a picture in an enhanced format metafile.

Syntax

```
typedef struct tagEMR {  
    DWORD iType;  
    DWORD nSize;  
} EMR, *PEMR;
```

Members

iType

The record type. The parameter can be one of the following (with a link to the associated record structure).

EMR_ABORTPATH EMR_ALPHABLEND EMR_ANGLEARC EMR_ARC EMR_ARCTO EMR_BEGINPATH EMR_BITBLT
EMR_CHORD EMR_CLOSEFIGURE EMR_COLORCORRECTPALETTE EMR_COLORMATCHTOTARGETW
EMR_CREATEBRUSHINDIRECT EMR_CREATECOLORSPACE EMR_CREATECOLORSPACEW
EMR_CREATEDIBPATTERNBRUSHPT EMR_CREATEMONOBRAHUS EMR_CREATEPALETTE EMR_CREATEPEN
EMR_DELETECOLORSPACE EMR_DELETEOBJECT EMR_ELLIPSE EMR_ENDPATH EMR_EOF
EMR_EXCLUDECLIPRECT EMR_EXTCREATEFONTINDIRECTW EMR_EXTCREATEPEN EMR_EXTFLOODFILL
EMR_EXTSELECTCLIPRGN EMR_EXTEXTOUTA EMR_EXTEXTOUTW EMR_FILLPATH EMR_FILLRGN
EMR_FLATTENPATH EMR_FRAMEGN EMR_GDICOMMENT EMR_GLSBOUNDEDRECORD EMR_GLSRECORD
EMR_GRADIENTFILL EMR_INTERSECTCLIPRECT EMR_INVERTRGN EMR_LINETO EMR_MASKBLT
EMR_MODIFYWORLDTRANSFORM EMR_MOVETOEX EMR_OFFSETCLIPRGN EMR_PAINTRGN EMR_PIE
EMR_PIXELFORMAT EMR_PLGBLT EMR_POLYBEZIER EMR_POLYBEZIER16 EMR_POLYBEZIERTO
EMR_POLYBEZIERTO16 EMR_POLYDRAW EMR_POLYDRAW16 EMR_POLYGON EMR_POLYGON16
EMR_POLYLINE EMR_POLYLINE16 EMR_POLYLINETO EMR_POLYLINETO16 EMR_POLYPOLYGON
EMR_POLYPOLYGON16 EMR_POLYPOLYLINE EMR_POLYPOLYLINE16 EMR_POLYTEXTOUTA
EMR_POLYTEXTOUTW EMR_REALIZEPALETTE EMR_RECTANGLE EMR_RESIZEPALETTE EMR_RESTOREDC
EMR_ROUNDRECT EMR_SAVEDC EMR_SCALEVIEWPORTEXTEX EMR_SCALEWINDOWEXTEX
EMR_SELECTCLIPPATH EMR_SELECTOBJECT EMR_SELECTPALETTE EMR_SETARCDIRECTION EMR_SETBKCOLOR
EMR_SETBKMODE EMR_SETBRUSHORGEX EMR_SetCOLORADJUSTMENT EMR_SetCOLORSPACE
EMR_SETDIBITSODEVICE EMR_SETICMMODE EMR_SETICMPROFILEA EMR_SETICMPROFILEW
EMR_SETLAYOUT EMR_SETMAPMODE EMR_SETMAPPERFLAGS EMR_SETMETARGN EMR_SETMITERLIMIT
EMR_SETPALETTEENTRIES EMR_SetPIXELV EMR_SetPOLYFILLMODE EMR_SetROP2
EMR_SETSTRETCHBLTMODE EMR_SetTEXTALIGN EMR_SetTextColor EMR_SetVIEWPORTEXTEX
EMR_SetVIEWPORTORGEX EMR_SetWINDOWEXTEX EMR_SetWINDOWORGEX EMR_SetWORLDTRANSFORM
EMR_STRETCHBLT EMR_STRETCHDBITS EMR_STROKEANDFILLPATH EMR_STROKEPATH
EMR_TRANSPARENTBLT EMR_WIDENPATH

nSize

The size of the record, in bytes. This member must be a multiple of four.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRABORTPATH structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Contains data for the [AbortPath](#), [BeginPath](#), [EndPath](#), [CloseFigure](#), [FlattenPath](#), [WidenPath](#), [SetMetaRgn](#), [SaveDC](#), and [RealizePalette](#) enhanced metafile records.

Syntax

```
typedef struct tagABORTPATH {  
    EMR emr;  
} EMRABORTPATH, *PEMRABORTPATH, EMRBEGINPATH, *PEMRBEGINPATH, EMRENDPATH, *PEMRENDPATH, EMRCLOSEFIGURE,  
*PEMRCLOSEFIGURE, EMRFLATTENPATH, *PEMRFLATTENPATH, EMRWIDENPATH, *PEMRWIDENPATH, EMRSETMETARGN,  
*PEMRSETMETARGN, EMRSAVEDC, *PEMRSAVEDC, EMRREALIZEPALETTE, *PEMRREALIZEPALETTE;
```

Members

emr

The base structure for all record types.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRALPHABLEND structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRALPHABLEND** structure contains members for the [AlphaBlend](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRALPHABLEND {
    EMR        emr;
    RECTL     rclBounds;
    LONG       xDest;
    LONG       yDest;
    LONG       cxDest;
    LONG       cyDest;
    DWORD      dwRop;
    LONG       xSrc;
    LONG       ySrc;
    XFORM     xformSrc;
    COLORREF   crBkColorSrc;
    DWORD      iUsageSrc;
    DWORD      offBmiSrc;
    DWORD      cbBmiSrc;
    DWORD      offBitsSrc;
    DWORD      cbBitsSrc;
    LONG       cxSrc;
    LONG       cySrc;
} EMRALPHABLEND, *PEMRALPHABLEND;
```

Members

emr

The base structure for all record types.

rclBounds

Bounding rectangle, in device units.

xDest

The x coordinate, in logical units, of the upper-left corner of the destination rectangle.

yDest

The y coordinate, in logical units, of the upper-left corner of the destination rectangle.

cxDest

Logical width of the destination rectangle.

cyDest

Logical height of the destination rectangle.

dwRop

Stores the [BLENDFUNCTION](#) structure.

xSrc

Logical x coordinate of the upper-left corner of the source rectangle.

ySrc

Logical y coordinate of the upper-left corner of the source rectangle.

xformSrc

World-space to page-space transformation of the source device context.

crBkColorSrc

Background color (the RGB value) of the source device context. To make a [COLORREF](#) value, use the [RGB](#) macro.

iUsageSrc

Source bitmap information color table usage (DIB_RGB_COLORS).

offBmiSrc

Offset to the source [BITMAPINFO](#) structure.

cbBmiSrc

Size of the source [BITMAPINFO](#) structure.

offBitsSrc

Offset to the source bitmap bits.

cbBitsSrc

Size of the source bitmap bits.

cxSrc

Width of source rectangle in logical units.

cySrc

Height of the source rectangle in logical units.

Remarks

This structure is to be used during metafile playback.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[AlphaBlend](#)

[BITMAPINFO](#)

[COLORREF](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RGB](#)

EMRANGLEARC structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRANGLEARC** structure contains members for the [AngleArc](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRANGLEARC {  
    EMR      emr;  
    POINTL  ptlCenter;  
    DWORD    nRadius;  
    FLOAT    eStartAngle;  
    FLOAT    eSweepAngle;  
} EMRANGLEARC, *PEMRANGLEARC;
```

Members

`emr`

The base structure for all record types.

`ptlCenter`

Logical coordinates of a circle's center.

`nRadius`

A circle's radius, in logical units.

`eStartAngle`

An arc's start angle, in degrees.

`eSweepAngle`

An arc's sweep angle, in degrees.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[AngleArc](#)

[Metafile Structures](#)

Metafiles Overview

EMRARC structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRARC**, **EMRARCTO**, **EMRCHORD**, and **EMRPIE** structures contain members for the [Arc](#), [ArcTo](#), [Chord](#), and [Pie](#) enhanced metafile records.

Syntax

```
typedef struct tagEMRARC {  
    EMR     emr;  
    RECTL  rclBox;  
    POINTL ptlStart;  
    POINTL ptlEnd;  
} EMRARC, *PEMRARC, EMRARCTO, *PEMRARCTO, EMRCHORD, *PEMRCHORD, EMRPIE, *PEMRPIE;
```

Members

`emr`

Base structure for all record types.

`rclBox`

Bounding rectangle in logical units.

`ptlStart`

Coordinates of first radial ending point in logical units.

`ptlEnd`

Coordinates of second radial ending point in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRBITBLT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRBITBLT** structure contains members for the [BitBlt](#) enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

Syntax

```
typedef struct tagEMRBITBLT {  
    EMR      emr;  
    RECTL   rclBounds;  
    LONG     xDest;  
    LONG     yDest;  
    LONG     cxDest;  
    LONG     cyDest;  
    DWORD    dwRop;  
    LONG     xSrc;  
    LONG     ySrc;  
    XFORM    xformSrc;  
    COLORREF crBkColorSrc;  
    DWORD    iUsageSrc;  
    DWORD    offBmiSrc;  
    DWORD    cbBmiSrc;  
    DWORD    offBitsSrc;  
    DWORD    cbBitsSrc;  
} EMRBITBLT, *PEMRBITBLT;
```

Members

emr

The base structure for all record types.

rclBounds

Bounding rectangle, in device units.

xDest

Logical x-coordinate of the upper-left corner of the destination rectangle.

yDest

Logical y-coordinate of the upper-left corner of the destination rectangle.

cxDest

Logical width of the destination rectangle.

cyDest

Logical height of the destination rectangle.

dwRop

Raster-operation code. These codes define how the color data of the source rectangle is to be combined with the

color data of the destination rectangle to achieve the final color.

xSrc

Logical x-coordinate of the upper-left corner of the source rectangle.

ySrc

Logical y-coordinate of the upper-left corner of the source rectangle.

xformSrc

World-space to page-space transformation of the source device context.

crBkColorSrc

Background color (the RGB value) of the source device context. To make a [COLORREF](#) value, use the [RGB](#) macro.

iUsageSrc

Value of the **bmiColors** member of the [BITMAPINFO](#) structure. The **iUsageSrc** member can be either the DIB_PAL_COLORS or DIB_RGB_COLORS value.

offBmiSrc

Offset to source [BITMAPINFO](#) structure.

cbBmiSrc

Size of source [BITMAPINFO](#) structure.

offBitsSrc

Offset to source bitmap bits.

cbBitsSrc

Size of source bitmap bits.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[BitBlt](#)

[COLORREF](#)

[Metafile Structures](#)

[Metafiles Overview](#)

RGB

EMRCOLORCORRECTPALETTE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRCOLORCORRECTPALETTE** structure contains members for the [ColorCorrectPalette](#) enhanced metafile record.

Syntax

```
typedef struct tagCOLORCORRECTPALETTE {  
    EMR     emr;  
    DWORD   ihPalette;  
    DWORD   nFirstEntry;  
    DWORD   nPalEntries;  
    DWORD   nReserved;  
} EMRCOLORCORRECTPALETTE, *PEMRCOLORCORRECTPALETTE;
```

Members

emr

The base structure for all record types.

ihPalette

The index of the palette handle to color correct.

nFirstEntry

The index of the first entry in the palette to color correct.

nPalEntries

The number of palette entries to color correct.

nReserved

Reserved.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[ColorCorrectPalette](#)

[Metafile Structures](#)

Metafiles Overview

EMRCOLORMATCHTOTARGET structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRCOLORMATCHTOTARGET** structure contains members for the [ColorMatchToTarget](#) enhanced metafile record.

Syntax

```
typedef struct tagCOLORMATCHTOTARGET {  
    EMR     emr;  
    DWORD   dwAction;  
    DWORD   dwFlags;  
    DWORD   cbName;  
    DWORD   cbData;  
    BYTE    Data[1];  
} EMRCOLORMATCHTOTARGET, *PEMRCOLORMATCHTOTARGET;
```

Members

emr

The base structure for all record types.

dwAction

The action to be taken. This member can be one of the following values.

ACTION	MEANING
CS_ENABLE	Maps colors to the target device's color gamut. This enables color proofing. All subsequent draw commands to the DC will render colors as they would appear on the target device.
CS_DISABLE	Disables color proofing.
CS_DELETE_TRANSFORM	If color management is enabled for the target profile, disables it and deletes the concatenated transform.

dwFlags

This parameter can be the following value.

FLAG	MEANING
COLORMATCHTOTARGET_EMBEDDED	Indicates that a color profile has been embedded in the metafile.

cbName

The size of the desired target profile name, in bytes.

cbData

The size of the raw target profile data in bytes, if it is attached.

Data

An array containing the target profile name and the raw target profile data. The size of the array is **cbName** + **cbData**. If **cbData** is nonzero the raw target profile data is attached and follows the target profile name at location **Data[cbName]**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[ColorMatchToTarget](#)

[Metafile Structures](#)

[Metafiles Overview](#)

EMRCREATEBRUSHINDIRECT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRCREATEBRUSHINDIRECT** structure contains members for the [CreateBrushIndirect](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRCREATEBRUSHINDIRECT {  
    EMR         emr;  
    DWORD       ihBrush;  
    LOGBRUSH32  lb;  
} EMRCREATEBRUSHINDIRECT, *PEMRCREATEBRUSHINDIRECT;
```

Members

emr

The base structure for all record types.

ihBrush

Index of brush in handle table.

lb

A [LOGBRUSH32](#) structure containing information about the brush. The **IbStyle** member must be either the **BS_SOLID**, **BS_HOLLOW**, **BS_NULL**, or **BS_HATCHED** value.

Note, that if your code is used on both 32-bit and 64-bit platforms, you must use the [LOGBRUSH32](#) structure. This maintains compatibility between the platforms when you record the metafile on one platform and use it on the other platform. If your code remains on one platform, it is sufficient to use [LOGBRUSH](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[CreateBrushIndirect](#)

[LOGBRUSH](#)

[LOGBRUSH32](#)

[Metafile Structures](#)

Metafiles Overview

EMRCREATECOLORSPACE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRCREATECOLORSPACE** structure contains members for the **CreateColorSpace** enhanced metafile record.

Syntax

```
typedef struct tagEMRCREATECOLORSPACE {  
    EMR          emr;  
    DWORD        ihCS;  
    LOGCOLORSPACEA lcs;  
} EMRCREATECOLORSPACE, *PEMRCREATECOLORSPACE;
```

Members

emr

The base structure for all record types.

ihCS

The index of the color space in handle table.

lcs

The logical color space.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[CreateColorSpace](#)

[EMR](#)

[EMRCREATECOLORSPACEW](#)

[LOGCOLORSPACE](#)

[Metafile Structures](#)

[Metafiles Overview](#)

EMRCREATECOLORSPACEW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRCREATECOLORSPACEW** structure contains members for the [CreateColorSpace](#) enhanced metafile record. It differs from [EMRCREATECOLORSPACE](#) in that it has a Unicode logical color space and also has an optional array containing raw source profile data.

Syntax

```
typedef struct tagEMRCREATECOLORSPACEW {
    EMR          emr;
    DWORD        ihCS;
    LOGCOLORSPACEW lcs;
    DWORD        dwFlags;
    DWORD        cbData;
    BYTE         Data[1];
} EMRCREATECOLORSPACEW, *PEMRCREATECOLORSPACEW;
```

Members

emr

The base structure for all record types.

ihCS

Index of the color space in handle table.

lcs

Logical color space. Note, this is the Unicode version of the structure.

dwFlags

Can be the following.

FLAG	MEANING
CREATECOLORSPACE_EMBEDDED	Indicates that a color space is embedded in the metafile.

cbData

Size of the raw source profile data in bytes, if it is attached.

Data

An array containing the source profile data. The size of the array is **cbData**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[CreateColorSpace](#)

[EMRCREATECOLORSPACE](#)

[Metafile Structures](#)

[Metafiles Overview](#)

EMRCREATEDIBPATTERNBRUSHPT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRCREATEDIBPATTERNBRUSHPT** structure contains members for the [CreateDIBPatternBrushPt](#) enhanced metafile record. The [BITMAPINFO](#) structure is followed by the bitmap bits that form a packed device-independent bitmap (DIB).

Syntax

```
typedef struct tagEMRCREATEDIBPATTERNBRUSHPT {
    EMR     emr;
    DWORD   ihBrush;
    DWORD   iUsage;
    DWORD   offBmi;
    DWORD   cbBmi;
    DWORD   offBits;
    DWORD   cbBits;
} EMRCREATEDIBPATTERNBRUSHPT, *PEMRCREATEDIBPATTERNBRUSHPT;
```

Members

emr

The base structure for all record types.

ihBrush

Index of brush in handle table.

iUsage

Value specifying whether the **bmiColors** member of the [BITMAPINFO](#) structure was provided and, if so, whether **bmiColors** contains explicit red, green, blue (RGB) values or indices. The **iUsage** member must be either the **DIB_PAL_COLORS** or **DIB_RGB_COLORS** value.

offBmi

Offset to [BITMAPINFO](#) structure.

cbBmi

Size of [BITMAPINFO](#) structure.

offBits

Offset to bitmap bits.

cbBits

Size of bitmap bits.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[CreateDIBPatternBrushPt](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RGB](#)

EMRCREATEMONOBRUSH structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRCREATEMONOBRUSH** structure contains members for the [CreatePatternBrush](#) (when passed a monochrome bitmap) or [CreateDIBPatternBrush](#) (when passed a monochrome DIB) enhanced metafile records.

Syntax

```
typedef struct tagEMRCREATEMONOBRUSH {  
    EMR     emr;  
    DWORD   ihBrush;  
    DWORD   iUsage;  
    DWORD   offBmi;  
    DWORD   cbBmi;  
    DWORD   offBits;  
    DWORD   cbBits;  
} EMRCREATEMONOBRUSH, *PEMRCREATEMONOBRUSH;
```

Members

emr

The base structure for all record types.

ihBrush

Index of brush in handle table.

iUsage

Value specifying whether the **bmiColors** member of the [BITMAPINFO](#) structure was provided and, if so, whether **bmiColors** contains explicit red, green, blue (RGB) values or indices. The **iUsage** member must be either the **DIB_PAL_COLORS** or **DIB_RGB_COLORS** value.

offBmi

Offset to [BITMAPINFO](#) structure.

cbBmi

Size of [BITMAPINFO](#) structure.

offBits

Offset to bitmap bits.

cbBits

Size of bitmap bits.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[CreateDIBPatternBrush](#)

[CreatePatternBrush](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RGB](#)

EMRCREATEPALETTE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRCREATEPALETTE** structure contains members for the [CreatePalette](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRCREATEPALETTE {  
    EMR         emr;  
    DWORD       ihPal;  
    LOGPALETTE  lgp1;  
} EMRCREATEPALETTE, *PEMRCREATEPALETTE;
```

Members

emr

The base structure for all record types.

ihPal

Index of palette in handle table.

lgp1

A [LOGPALETTE](#) structure that contains information about the palette. Note that **peFlags** members in the [PALETTEENTRY](#) structures do not contain any flags.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[CreatePalette](#)

[LOGPALETTE](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[PALETTEENTRY](#)

EMRCREATEPEN structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRCREATEPEN** structure contains members for the **CreatePen** enhanced metafile record.

Syntax

```
typedef struct tagEMRCREATEPEN {  
    EMR     emr;  
    DWORD   ihPen;  
    LOGPEN  lopn;  
} EMRCREATEPEN, *PEMRCREATEPEN;
```

Members

emr

The base structure for all record types.

ihPen

Index to pen in handle table.

lopn

Logical pen.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[CreatePen](#)

[Metafile Structures](#)

[Metafiles Overview](#)

EMRELLIPSE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRELLIPSE** and **EMRRECTANGLE** structures contain members for the [Ellipse](#) and [Rectangle](#) enhanced metafile records.

Syntax

```
typedef struct tagEMRELLIPSE {  
    EMR     emr;  
    RECTL  rclBox;  
} EMRELLIPSE, *PEMRELLIPSE, EMRRECTANGLE, *PEMRRECTANGLE;
```

Members

`emr`

Base structure for all record types.

`rclBox`

Bounding rectangle in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Ellipse](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[Rectangle](#)

EMREOF structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMREOF** structure contains data for the enhanced metafile record that indicates the end of the metafile.

Syntax

```
typedef struct tagEMREOF {  
    EMR     emr;  
    DWORD   nPalEntries;  
    DWORD   offPalEntries;  
    DWORD   nSizeLast;  
} EMREOF, *PEMREOF;
```

Members

emr

The base structure for all record types.

nPalEntries

The number of palette entries.

offPalEntries

The offset, in bytes, to an array of [PALETTEENTRY](#) structures.

nSizeLast

The same size as the **nSize** member of the [EMR](#) structure. This member must be the last double word of the record. If palette entries exist, they precede this member.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[PALETTEENTRY](#)

EMREXCLUDECLIPRECT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMREXCLUDECLIPRECT** and **EMRINTERSECTCLIPRECT** structures contain members for the [ExcludeClipRect](#) and [IntersectClipRect](#) enhanced metafile records.

Syntax

```
typedef struct tagEMREXCLUDECLIPRECT {  
    EMR     emr;  
    RECTL  rclClip;  
} EMREXCLUDECLIPRECT, *PEMREXCLUDECLIPRECT, EMRINTERSECTCLIPRECT, *PEMRINTERSECTCLIPRECT;
```

Members

emr

Base structure for all record types.

rclClip

Clipping rectangle in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMREXTCREATEFONTINDIRECTW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMREXTCREATEFONTINDIRECTW** structure contains members for the [CreateFontIndirect](#) enhanced metafile record.

Syntax

```
typedef struct tagEMREXTCREATEFONTINDIRECTW {
    EMR          emr;
    DWORD        ihFont;
    EXTLOGFONTW elfw;
} EMREXTCREATEFONTINDIRECTW, *PEMREXTCREATEFONTINDIRECTW;
```

Members

`emr`

The base structure for all record types.

`ihFont`

Index to the font in handle table.

`elfw`

Logical font.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[CreateFontIndirect](#)

[Metafile Structures](#)

[Metafiles Overview](#)

EMREXTCREATEOPEN structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMREXTCREATEOPEN** structure contains members for the [ExtCreatePen](#) enhanced metafile record. If the record contains a [BITMAPINFO](#) structure, it is followed by the bitmap bits that form a packed device-independent bitmap (DIB).

Syntax

```
typedef struct tagEMREXTCREATEOPEN {
    EMR          emr;
    DWORD        ihPen;
    DWORD        offBmi;
    DWORD        cbBmi;
    DWORD        offBits;
    DWORD        cbBits;
    EXTLOGOPEN32 elp;
} EMREXTCREATEOPEN, *PEMREXTCREATEOPEN;
```

Members

emr

The base structure for all record types.

ihPen

Index to pen in handle table.

offBmi

Offset to [BITMAPINFO](#) structure, if any.

cbBmi

Size of [BITMAPINFO](#) structure, if any.

offBits

Offset to brush bitmap bits, if any.

cbBits

Size of brush bitmap bits, if any.

elp

Extended logical pen, including the **elpStyleEntry** member of the [EXTLOGOPEN](#) structure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[EXTLOGOPEN](#)

[ExtCreatePen](#)

[Metafile Structures](#)

[Metafiles Overview](#)

EMREXTFLOODFILL structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMREXTFLOODFILL** structure contains members for the [ExtFloodFill](#) enhanced metafile record.

Syntax

```
typedef struct tagEMREXTFLOODFILL {  
    EMR     emr;  
    POINTL  pt1Start;  
    COLORREF crColor;  
    DWORD    iMode;  
} EMREXTFLOODFILL, *PEMREXTFLOODFILL;
```

Members

emr

The base structure for all record types.

pt1Start

Coordinates, in logical units, where filling begins.

crColor

Color of fill. To make a [COLORREF](#) value, use the [RGB](#) macro.

iMode

Type of fill operation to be performed. This member must be either the FLOODFILLBORDER or FLOODFILLSURFACE value.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[COLORREF](#)

[ExtFloodFill](#)

[Metafile Structures](#)

[Metafiles Overview](#)

RGB

EMREXTSELECTCLIPRGN structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMREXTSELECTCLIPRGN** structure contains members for the [ExtSelectClipRgn](#) enhanced metafile record.

Syntax

```
typedef struct tagEMREXTSELECTCLIPRGN {  
    EMR     emr;  
    DWORD   cbRgnData;  
    DWORD   iMode;  
    BYTE    RgnData[1];  
} EMREXTSELECTCLIPRGN, *PEMREXTSELECTCLIPRGN;
```

Members

emr

The base structure for all record types.

cbRgnData

Size of region data, in bytes.

iMode

Operation to be performed. This member must be one of the following values: RGN_AND, RGN_COPY, RGN_DIFF, RGN_OR, or RGN_XOR.

RgnData

Buffer containing a [RGNDATA](#) structure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[ExtSelectClipRgn](#)

[Metafile Structures](#)

[Metafiles Overview](#)

EMREXTTEXTOUTA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMREXTTEXTOUTA** and **EMREXTTEXTOUTW** structures contain members for the [ExtTextOut](#), [TextOut](#), or [DrawText](#) enhanced metafile records.

Syntax

```
typedef struct tagEMREXTTEXTOUTA {
    EMR      emr;
    RECTL   rclBounds;
    DWORD    iGraphicsMode;
    FLOAT    exScale;
    FLOAT    eyScale;
    EMRTEXT  emrtext;
} EMREXTTEXTOUTA, *PEMREXTTEXTOUTA, EMREXTTEXTOUTW, *PEMREXTTEXTOUTW;
```

Members

emr

Base structure for all record types.

rclBounds

Bounding rectangle, in device units.

iGraphicsMode

Current graphics mode. This member can be either the GM_COMPATIBLE or GM_ADVANCED value.

exScale

X-scaling factor from page units to .01mm units if the graphics mode is the GM_COMPATIBLE value.

eyScale

Y-scaling factor from page units to .01mm units if the graphics mode is the GM_COMPATIBLE value.

emrtext

EMRTEXT structure, which is followed by the string and the intercharacter spacing array.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRFILLPATH structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRFILLPATH**, **EMRSTROKEANDFILLPATH**, and **EMRSTROKEPATH** structures contain members for the **FillPath**, **StrokeAndFillPath**, and **StrokePath** enhanced metafile records.

Syntax

```
typedef struct tagEMRFILLPATH {  
    EMR     emr;  
    RECTL  rclBounds;  
} EMRFILLPATH, *PEMRFILLPATH, EMRSTROKEANDFILLPATH, *PEMRSTROKEANDFILLPATH, EMRSTROKEPATH, *PEMRSTROKEPATH;
```

Members

emr

Base structure for all record types.

rclBounds

Bounding rectangle, in device units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRFILLRGN structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRFILLRGN** structure contains members for the [FillRgn](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRFILLRGN {  
    EMR     emr;  
    RECTL  rclBounds;  
    DWORD   cbRgnData;  
    DWORD   ihBrush;  
    BYTE    RgnData[1];  
} EMRFILLRGN, *PEMRFILLRGN;
```

Members

`emr`

The base structure for all record types.

`rclBounds`

Bounding rectangle, in device units.

`cbRgnData`

Size of region data, in bytes.

`ihBrush`

Index of brush, in handle table.

`RgnData`

Buffer containing [RGNDATA](#) structure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[FillRgn](#)

[Metafile Structures](#)

Metafiles Overview

RGNDATA

EMRFORMAT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRFORMAT** structure contains information that identifies graphics data in an enhanced metafile. A **GDICOMMENT_MULTIFORMATS** enhanced metafile public comment contains an array of **EMRFORMAT** structures.

Syntax

```
typedef struct tagEMRFORMAT {  
    DWORD dSignature;  
    DWORD nVersion;  
    DWORD cbData;  
    DWORD offData;  
} EMRFORMAT, *PEMRFORMAT;
```

Members

dSignature

Contains a picture format identifier. The following identifier values are defined.

IDENTIFIER	MEANING
ENHMETA_SIGNATURE	The picture is in enhanced metafile format.
EPS_SIGNATURE	The picture is in encapsulated PostScript file format.

nVersion

Contains a picture version number. The following version number value is defined.

VERSION	MEANING
1	This is the version number of a level 1 encapsulated PostScript file.

cbData

The size, in bytes, of the picture data.

offData

Specifies an offset to the picture data. The offset is figured from the start of the **GDICOMMENT_MULTIFORMATS** public comment within which this **EMRFORMAT** structure is embedded. The offset must be a **DWORD** offset.

Remarks

The reference page for [GdiComment](#) discusses enhanced metafile public comments in general, and the **GDICOMMENT_MULTIFORMATS** public comment in particular.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[GdiComment](#)

[Metafile Structures](#)

[Metafiles Overview](#)

EMRFRAMERGN structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRFRAMERGN** structure contains members for the [FrameRgn](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRFRAMERGN {  
    EMR     emr;  
    RECTL  rclBounds;  
    DWORD   cbRgnData;  
    DWORD   ihBrush;  
    SIZEL   szlStroke;  
    BYTE    RgnData[1];  
} EMRFRAMERGN, *PEMRFRAMERGN;
```

Members

`emr`

The base structure for all record types.

`rclBounds`

Bounding rectangle, in device units.

`cbRgnData`

Size of region data, in bytes.

`ihBrush`

Index of brush, in handle table.

`szlStroke`

Width and height of region frame, in logical units.

`RgnData`

Buffer containing [RGNDATA](#) structure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[FrameRgn](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RGNDATA](#)

EMRGDICONMENT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRGDICONMENT** structure contains application-specific data. This enhanced metafile record is only meaningful to applications that know the format of the data and how to utilize it. This record is ignored by graphics device interface (GDI) during playback of the enhanced metafile.

Syntax

```
typedef struct tagEMRGDICONMENT {
    EMR     emr;
    DWORD   cbData;
    BYTE    Data[1];
} EMRGDICONMENT, *PEMRGDICONMENT;
```

Members

emr

The base structure for all record types.

cbData

Size of data buffer, in bytes.

[1] Data

Application-specific data.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRGLSBOUNDEDRECORD structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRGLSBOUNDEDRECORD** structure contains members for an enhanced metafile record generated by OpenGL functions. It contains data for OpenGL functions with information in pixel units that must be scaled when playing the metafile.

Syntax

```
typedef struct tagEMRGLSBOUNDEDRECORD {  
    EMR     emr;  
    RECTL  rclBounds;  
    DWORD   cbData;  
    BYTE    Data[1];  
} EMRGLSBOUNDEDRECORD, *PEMRGLSBOUNDEDRECORD;
```

Members

emr

The base structure for all record types.

rclBounds

Bounds of the rectangle, in device coordinates, within which to perform the OpenGL function. For more information, see Remarks.

cbData

Size of *Data*, in bytes.

Data

Array of data representing the OpenGL function to be performed.

Remarks

The coordinates in **rclBounds** are in OpenGL pixel coordinates, which generally equate to window coordinates. For example, if the [glBitmap](#) function has width1 and height1, the bounds will be 0, 0, width1, height1.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[OpenGL on Windows NT, Windows 2000, and Windows 95/98](#)

EMRGLSRECORD structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRGLSRECORD** structure contains members for an enhanced metafile record generated by OpenGL functions. It contains data for OpenGL functions that scale automatically to the OpenGL viewport.

Syntax

```
typedef struct tagEMRGLSRECORD {  
    EMR     emr;  
    DWORD   cbData;  
    BYTE    Data[1];  
} EMRGLSRECORD, *PEMRGLSRECORD;
```

Members

emr

The base structure for all records.

cbData

Size of **Data**, in bytes.

Data

Array of data representing the OpenGL function to be performed.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[OpenGL on Windows NT, Windows 2000, and Windows 95/98](#)

EMRGRADIENTFILL structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRGRADIENTFILL** structure contains members for the [GradientFill](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRGRADIENTFILL {  
    EMR        emr;  
    RECTL     rclBounds;  
    DWORD      nVer;  
    DWORD      nTri;  
    ULONG      ulMode;  
    TRIVERTEX Ver[1];  
} EMRGRADIENTFILL, *PEMRGRADIENTFILL;
```

Members

emr

The base structure for all record types.

rclBounds

The bounding rectangle, in device units.

nVer

The number of vertices.

nTri

The number of rectangles or triangles to be passed to [GradientFill](#).

ulMode

The gradient fill mode. This member can be one of the following values.

VALUE	MEANING
GRADIENT_FILL_RECT_H	In this mode, two endpoints describe a rectangle. The rectangle is defined to have a constant color (specified by the TRIVERTEX structure) for the left and right edges. GDI interpolates the color from the left to right edge and fills the interior.
GRADIENT_FILL_RECT_V	In this mode, two endpoints describe a rectangle. The rectangle is defined to have a constant color (specified by the TRIVERTEX structure) for the top and bottom edges. GDI interpolates the color from the top to bottom edge and fills the interior.

GRADIENT_FILL_TRIANGLE	In this mode, an array of TRIVERTEX structures is passed to GDI along with a list of array indexes that describe separate triangles. GDI performs linear interpolation between triangle vertices and fills the interior. Drawing is done directly in 24- and 32-bpp modes. Dithering is performed in 16-, 8-, 4-, and 1-bpp mode.
-------------------------------	---

Ver

An array of [TRIVERTEX](#) structures that each define a vertex.

Remarks

This is a variable-length structure. The **Ver** member designates the beginning of the variable-length area. First comes an array of **nVer** [TRIVERTEX](#) structures to pass the vertices. Next comes an array of either **nTri** [GRADIENT_TRIANGLE](#) structures or **nTri** [GRADIENT_RECT](#) structures, depending on the value of **ulMode** (triangles or rectangles).

This structure is to be used during metafile playback.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[EMR](#)

[GRADIENT_RECT](#)

[GRADIENT_TRIANGLE](#)

[GradientFill](#)

[Metafile Structures](#)

[Metafiles](#)

[Metafiles Overview](#)

EMRINVERTRGN structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRINVERTRGN** and **EMRPAINTRGN** structures contain members for the [InvertRgn](#) and [PaintRgn](#) enhanced metafile records.

Syntax

```
typedef struct tagEMRINVERTRGN {  
    EMR     emr;  
    RECTL  rclBounds;  
    DWORD   cbRgnData;  
    BYTE    RgnData[1];  
} EMRINVERTRGN, *PEMRINVERTRGN, EMRPAINTRGN, *PEMRPAINTRGN;
```

Members

`emr`

Base structure for all record types.

`rclBounds`

Bounding rectangle, in device units.

`cbRgnData`

Size of region data, in bytes.

`RgnData`

Buffer containing an [RGNDATA](#) structure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[InvertRgn](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[PaintRgn](#)

EMRLINETO structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRLINETO** and **EMRMOVETOEX** structures contains members for the [LineTo](#) and [MoveToEx](#) enhanced metafile records.

Syntax

```
typedef struct tagEMRLINETO {  
    EMR     emr;  
    POINTL pt1;  
} EMRLINETO, *PEMRLINETO, EMRMOVETOEX, *PEMRMOVETOEX;
```

Members

emr

Base structure for all record types.

pt1

Coordinates of the line's ending point for the [LineTo](#) function or coordinates of the new current position for the [MoveToEx](#) function in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRMASKBLT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRMASKBLT** structure contains members for the [MaskBlt](#) enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

Syntax

```
typedef struct tagEMRMASKBLT {
    EMR      emr;
    RECTL   rclBounds;
    LONG     xDest;
    LONG     yDest;
    LONG     cxDest;
    LONG     cyDest;
    DWORD    dwRop;
    LONG     xSrc;
    LONG     ySrc;
    XFORM    xformSrc;
    COLORREF crBkColorSrc;
    DWORD    iUsageSrc;
    DWORD    offBmiSrc;
    DWORD    cbBmiSrc;
    DWORD    offBitsSrc;
    DWORD    cbBitsSrc;
    LONG     xMask;
    LONG     yMask;
    DWORD    iUsageMask;
    DWORD    offBmiMask;
    DWORD    cbBmiMask;
    DWORD    offBitsMask;
    DWORD    cbBitsMask;
} EMRMASKBLT, *PEMRMASKBLT;
```

Members

emr

The base structure for all record types.

rclBounds

Bounding rectangle, in device units.

xDest

Logical x-coordinate of the upper-left corner of the destination rectangle.

yDest

Logical y-coordinate of the upper-left corner of the destination rectangle.

cxDest

Logical width of the destination rectangle.

`cyDest`

Logical height of the destination rectangle.

`dwRop`

Raster-operation code. These codes define how the color data of the source rectangle is to be combined with the color data of the destination rectangle to achieve the final color.

`xSrc`

Logical x-coordinate of the upper-left corner of the source rectangle.

`ySrc`

Logical y-coordinate of the upper-left corner of the source rectangle.

`xformSrc`

World-space to page-space transformation of the source device context.

`crBkColorSrc`

Background color (the RGB value) of the source device context. To make a [COLORREF](#) value, use the [RGB](#) macro.

`iUsageSrc`

Value of the `bmiColors` member of the source [BITMAPINFO](#) structure. The `iUsageSrc` member can be either the DIB_PAL_COLORS or DIB_RGB_COLORS value.

`offBmiSrc`

Offset to source [BITMAPINFO](#) structure.

`cbBmiSrc`

Size of source [BITMAPINFO](#) structure.

`offBitsSrc`

Offset to source bitmap bits.

`cbBitsSrc`

Size of source bitmap bits.

`xMask`

Horizontal pixel offset into mask bitmap.

`yMask`

Vertical pixel offset into mask bitmap.

`iUsageMask`

Value of the `bmiColors` member of the mask [BITMAPINFO](#) structure.

`offBmiMask`

Offset to mask [BITMAPINFO](#) structure.

`cbBmiMask`

Size of mask [BITMAPINFO](#) structure.

`offBitsMask`

Offset to mask bitmap bits.

`cbBitsMask`

Size of mask bitmap bits.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[COLORREF](#)

[MaskBlt](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RGB](#)

EMRMODIFYWORLDTRANSFORM structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRMODIFYWORLDTRANSFORM** structure contains members for the [ModifyWorldTransform](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRMODIFYWORLDTRANSFORM {  
    EMR     emr;  
    XFORM   xform;  
    DWORD   iMode;  
} EMRMODIFYWORLDTRANSFORM, *PEMRMODIFYWORLDTRANSFORM;
```

Members

emr

The base structure for all record types.

xform

The world-space to page-space transform data.

iMode

Indicates how the transformation data modifies the current world transformation. This member can be one of the following values: MWT_IDENTITY, MWT_LEFTMULTIPLY, or MWT_RIGHTMULTIPLY.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[ModifyWorldTransform](#)

[XFORM](#)

EMROFFSETCLIPRGN structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMROFFSETCLIPRGN** structure contains members for the **OffsetClipRgn** enhanced metafile record.

Syntax

```
typedef struct tagEMROFFSETCLIPRGN {  
    EMR     emr;  
    POINTL pt1Offset;  
} EMROFFSETCLIPRGN, *PEMROFFSETCLIPRGN;
```

Members

`emr`

The base structure for all record types.

`pt1Offset`

The logical coordinates of offset.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[OffsetClipRgn](#)

EMRPIXELFORMAT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRPIXELFORMAT** structure contains the members for the [SetPixelFormat](#) enhanced metafile record. The pixel format information in [ENHMETAHEADER](#) refers to this structure.

Syntax

```
typedef struct tagEMRPIXELFORMAT {  
    EMR           emr;  
    PIXELEFORMATDESCRIPTOR pfd;  
} EMRPIXELFORMAT, *PEMRPIXELFORMAT;
```

Members

emr

The base structure for all record types.

pfd

A [PIXELEFORMATDESCRIPTOR](#) structure, which describes the pixel format.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[ENHMETAHEADER](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[PIXELEFORMATDESCRIPTOR](#)

[SetPixelFormat](#)

EMRPLGBT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRPLGBT** structure contains members for the [PlgBlt](#) enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

Syntax

```
typedef struct tagEMRPLGBT {
    EMR      emr;
    RECTL   rclBounds;
    POINTL  aptlDest[3];
    LONG     xSrc;
    LONG     ySrc;
    LONG     cxSrc;
    LONG     cySrc;
    XFORM   xformSrc;
    COLORREF crBkColorSrc;
    DWORD    iUsageSrc;
    DWORD    offBmiSrc;
    DWORD    cbBmiSrc;
    DWORD    offBitsSrc;
    DWORD    cbBitsSrc;
    LONG     xMask;
    LONG     yMask;
    DWORD    iUsageMask;
    DWORD    offBmiMask;
    DWORD    cbBmiMask;
    DWORD    offBitsMask;
    DWORD    cbBitsMask;
} EMRPLGBT, *PEMRPLGBT;
```

Members

emr

The base structure for all record types.

rclBounds

Bounding rectangle, in device units.

aptlDest

Array of three points in logical space that identify three corners of the destination parallelogram. The upper-left corner of the source rectangle is mapped to the first point in this array, the upper-right corner to the second point in this array, and the lower-left corner to the third point. The lower-right corner of the source rectangle is mapped to the implicit fourth point in the parallelogram.

xSrc

Logical x-coordinate of the upper-left corner of the source rectangle.

ySrc

Logical y-coordinate of the upper-left corner of the source rectangle.

`cxSrc`

Logical width of the source.

`cySrc`

Logical height of the source.

`xformSrc`

World-space to page-space transformation of the source device context.

`crBkColorSrc`

Background color (the RGB value) of the source device context. To make a [COLORREF](#) value, use the [RGB](#) macro.

`iUsageSrc`

Value of the `bmiColors` member of the [BITMAPINFO](#) structure. The `iUsageSrc` member can be either the `DIB_PAL_COLORS` or `DIB_RGB_COLORS` value.

`offBmiSrc`

Offset to source [BITMAPINFO](#) structure.

`cbBmiSrc`

Size of source [BITMAPINFO](#) structure.

`offBitsSrc`

Offset to source bitmap bits.

`cbBitsSrc`

Size of source bitmap bits.

`xMask`

Horizontal pixel offset into mask bitmap.

`yMask`

Vertical pixel offset into mask bitmap.

`iUsageMask`

Value of the `bmiColors` member of the mask [BITMAPINFO](#) structure.

`offBmiMask`

Offset to mask [BITMAPINFO](#) structure.

`cbBmiMask`

Size of mask [BITMAPINFO](#) structure.

`offBitsMask`

Offset to mask bitmap bits.

`cbBitsMask`

Size of mask bitmap bits.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[COLORREF](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[PlgBlt](#)

[RGB](#)

EMRPOLYDRAW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRPOLYDRAW** structure contains members for the [PolyDraw](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRPOLYDRAW {  
    EMR     emr;  
    RECTL   rclBounds;  
    DWORD   cptl;  
    POINTL  aptl[1];  
    BYTE    abTypes[1];  
} EMRPOLYDRAW, *PEMRPOLYDRAW;
```

Members

emr

The base structure for all record types.

rclBounds

The bounding rectangle, in device units.

cptl

The number of points.

aptl

An array of [POINTL](#) structures, representing the data points in logical units.

abTypes

An array of values that specifies how each point in the **aptl** array is used. Each element can be one of the following values: PT_MOVETO, PT LINETO, or PT_BEZIERTO. The PT LINETO or PT_BEZIERTO value can be combined with the PT CLOSEFIGURE value using the bitwise-OR operator.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[POINTL](#)

[PolyDraw](#)

[RECTL](#)

EMRPOLYDRAW16 structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRPOLYDRAW16** structure contains members for the [PolyDraw](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRPOLYDRAW16 {  
    EMR     emr;  
    RECTL   rclBounds;  
    DWORD   cpts;  
    POINTS  appts[1];  
    BYTE    abTypes[1];  
} EMRPOLYDRAW16, *PEMRPOLYDRAW16;
```

Members

emr

The base structure for all record types.

rclBounds

The bounding rectangle, in device units.

cpts

The number of points.

appts

An array of [POINTS](#) structures, representing the data points in logical units.

abTypes

An array of values that specifies how each point in the **appts** array is used. Each element can be one of the following values: PT_MOVETO, PT_LINETO, or PT_BEZIERTO. The PT_LINETO or PT_BEZIERTO value can be combined with the PT_CLOSEFIGURE value using the bitwise-OR operator.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[POINTS](#)

[PolyDraw](#)

[RECTL](#)

EMRPOLYLINE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRPOLYLINE**, **EMRPOLYBEZIER**, **EMRPOLYGON**, **EMRPOLYBEZIERTO**, and **EMRPOLYLINETO** structures contain members for the [Polyline](#), [PolyBezier](#), [Polygon](#), [PolyBezierTo](#), and [PolylineTo](#) enhanced metafile records.

Syntax

```
typedef struct tagEMRPOLYLINE {  
    EMR     emr;  
    RECTL  rclBounds;  
    DWORD   cptl;  
    POINTL aptl[1];  
} EMRPOLYLINE, *PEMRPOLYLINE, EMRPOLYBEZIER, *PEMRPOLYBEZIER, EMRPOLYGON, *PEMRPOLYGON, EMRPOLYBEZIERTO,  
*PEMRPOLYBEZIERTO, EMRPOLYLINETO, *PEMRPOLYLINETO;
```

Members

`emr`

Base structure for all record types.

`rclBounds`

Bounding rectangle, in device units.

`cptl`

Number of points array.

`aptl`

Array of 32-bit points, in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRPOLYLINE16 structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRPOLYLINE16**, **EMRPOLYBEZIER16**, **EMRPOLYGON16**, **EMRPOLYBEZIERTO16**, and **EMRPOLYLINETO16** structures contain members for the [Polyline](#), [PolyBezier](#), [Polygon](#), [PolyBezierTo](#), and [PolylineTo](#) enhanced metafile records.

Syntax

```
typedef struct tagEMRPOLYLINE16 {
    EMR     emr;
    RECTL  rclBounds;
    DWORD   cpts;
    POINTS  appts[1];
} EMRPOLYLINE16, *PEMRPOLYLINE16, EMRPOLYBEZIER16, *PEMRPOLYBEZIER16, EMRPOLYGON16, *PEMRPOLYGON16,
EMRPOLYBEZIERTO16, *PEMRPOLYBEZIERTO16, EMRPOLYLINETO16, *PEMRPOLYLINETO16;
```

Members

emr

Base structure for all record types.

rclBounds

Bounding rectangle, in device units.

cpts

Number of points in the array.

apts

Array of 16-bit points, in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRPOLYPOLYLINE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRPOLYPOLYLINE** and **EMRPOLYPOLYGON** structures contain members for the [PolyPolyline](#) and [PolyPolygon](#) enhanced metafile records.

Syntax

```
typedef struct tagEMRPOLYPOLYLINE {  
    EMR     emr;  
    RECTL   rclBounds;  
    DWORD   nPolys;  
    DWORD   cptl;  
    DWORD   aPolyCounts[1];  
    POINTL  aptl[1];  
} EMRPOLYPOLYLINE, *PEMRPOLYPOLYLINE, EMRPOLYPOLYGON, *PEMRPOLYPOLYGON;
```

Members

emr

The base structure for all record types.

rclBounds

The bounding rectangle, in device units.

nPolys

The number of polys.

cptl

The total number of points in all polys.

aPolyCounts

An array of point counts for each poly.

aptl

An array of [POINTL](#) structures, representing the points in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[POINTL](#)

[RECTL](#)

EMRPOLYPOLYLINE16 structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRPOLYPOLYLINE16** and **EMRPOLYPOLYGON16** structures contain members for the [PolyPolyline](#) and [PolyPolygon](#) enhanced metafile records.

Syntax

```
typedef struct tagEMRPOLYPOLYLINE16 {  
    EMR     emr;  
    RECTL   rclBounds;  
    DWORD   nPolys;  
    DWORD   cpts;  
    DWORD   aPolyCounts[1];  
    POINTS  appts[1];  
} EMRPOLYPOLYLINE16, *PEMRPOLYPOLYLINE16, EMRPOLYPOLYGON16, *PEMRPOLYPOLYGON16;
```

Members

emr

The base structure for all record types.

rclBounds

The bounding rectangle, in device units.

nPolys

The number of polys.

cpts

The total number of points in all polys.

aPolyCounts

An array of point counts for each poly.

apts

An array of [POINTS](#) structures, representing the points in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[POINTS](#)

[RECTL](#)

EMRPOLYTEXTOUTA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRPOLYTEXTOUTA** and **EMRPOLYTEXTOUTW** structures contain members for the **PolyTextOut** enhanced metafile record.

Syntax

```
typedef struct tagEMRPOLYTEXTOUTA {  
    EMR      emr;  
    RECTL   rclBounds;  
    DWORD    iGraphicsMode;  
    FLOAT    exScale;  
    FLOAT    eyScale;  
    LONG     cStrings;  
    EMRTEXT  aemrtext[1];  
} EMRPOLYTEXTOUTA, *PEMRPOLYTEXTOUTA, EMRPOLYTEXTOUTW, *PEMRPOLYTEXTOUTW;
```

Members

emr

Base structure for all record types.

rclBounds

Bounding rectangle, in device units.

iGraphicsMode

Current graphics mode. This member can be either the GM_COMPATIBLE or GM_ADVANCED value.

exScale

X-scaling factor from page units to .01mm units if the graphics mode is the GM_COMPATIBLE value.

eyScale

Y-scaling factor from page units to .01mm units if the graphics mode is the GM_COMPATIBLE value.

cStrings

Number of strings.

aemrtext

EMRTEXT structure, which is followed by the string and the intercharacter spacing array.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRRESIZEPALETTE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRRESIZEPALETTE** structure contains members for the **ResizePalette** enhanced metafile record.

Syntax

```
typedef struct tagEMRRESIZEPALETTE {  
    EMR     emr;  
    DWORD   ihPal;  
    DWORD   cEntries;  
} EMRRESIZEPALETTE, *PEMRRESIZEPALETTE;
```

Members

emr

The base structure for all record types.

ihPal

Index of the palette in the handle table.

cEntries

Number of entries in palette after resizing.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[ResizePalette](#)

EMRRESTOREDC structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRRESTOREDC** structure contains members for the **RestoreDC** enhanced metafile record.

Syntax

```
typedef struct tagEMRRESTOREDC {  
    EMR    emr;  
    LONG   iRelative;  
} EMRRESTOREDC, *PEMRRESTOREDC;
```

Members

`emr`

The base structure for all record types.

`iRelative`

Relative instance to restore.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[RestoreDC](#)

EMRROUNDRECT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRROUNDRECT** structure contains members for the **RoundRect** enhanced metafile record.

Syntax

```
typedef struct tagEMRROUNDRECT {  
    EMR     emr;  
    RECTL  rclBox;  
    SIZEL szlCorner;  
} EMRROUNDRECT, *PEMRROUNDRECT;
```

Members

`emr`

The base structure for all record types.

`rclBox`

Bounding rectangle, in logical units.

`szlCorner`

Width and height, in logical units, of the ellipse used to draw rounded corners.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[RoundRect](#)

EMRSCALEVIEWPORTEXTEX structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSCALEVIEWPORTEXTEX** and **EMRSCALEWINDOWEXTEX** structures contain members for the **ScaleViewportExtEx** and **ScaleWindowExtEx** enhanced metafile records.

Syntax

```
typedef struct tagEMRSCALEVIEWPORTEXTEX {  
    EMR     emr;  
    LONG    xNum;  
    LONG    xDenom;  
    LONG    yNum;  
    LONG    yDenom;  
} EMRSCALEVIEWPORTEXTEX, *PEMRSCALEVIEWPORTEXTEX, EMRSCALEWINDOWEXTEX, *PEMRSCALEWINDOWEXTEX;
```

Members

emr

Base structure for all record types.

xNum

Horizontal multiplicand.

xDenom

Horizontal divisor.

yNum

Vertical multiplicand.

yDenom

Vertical divisor.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRSELECTCLIPPATH structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Contains parameters for the [SelectClipPath](#), [SetBkMode](#), [SetMapMode](#), [SetPolyFillMode](#), [SetROP2](#), [SetStretchBltMode](#), [SetTextAlign](#), [SetICMMode](#), and [SetLayout](#) enhanced metafile records.

Syntax

```
typedef struct tagEMRSELECTCLIPPATH {  
    EMR     emr;  
    DWORD   iMode;  
} EMRSELECTCLIPPATH, *PEMRSELECTCLIPPATH, EMRSETBKMODE, *PEMRSETBKMODE, EMRSETPOLYFILLMODE, *PEMRSETPOLYFILLMODE,  
EMRSETROP2, *PEMRSETROP2, EMRSETSTRETCHBLTMODE, *PEMRSETSTRETCHBLTMODE, EMRSETICMMODE, *PEMRSETICMMODE, EMRSETTEXTALIGN,  
*PEMRSETTEXTALIGN;
```

Members

emr

The base structure for all record types.

iMode

A value and meaning that varies depending on the function contained in the enhanced metafile record. For a description of this member, see the documentation of the functions contained in this record.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[SelectClipPath](#)

[SetBkMode](#)

[SetICMMode](#)

[SetMapMode](#)

[SetPolyFillMode](#)

[SetROP2](#)

[SetStretchBltMode](#)

[SetTextAlign](#)

EMRSELECTOBJECT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSELECTOBJECT** and **EMRDELETEOBJECT** structures contain members for the **SelectObject** and **DeleteObject** enhanced metafile records.

Syntax

```
typedef struct tagEMRSELECTOBJECT {  
    EMR     emr;  
    DWORD   ihObject;  
} EMRSELECTOBJECT, *PEMRSELECTOBJECT, EMRDELETEOBJECT, *PEMRDELETEOBJECT;
```

Members

emr

The base structure for all record types.

ihObject

The index of an object in the handle table.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[GetStockObject](#)

[Metafile Structures](#)

[Metafiles Overview](#)

EMRSELECTPALETTE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSELECTPALETTE** structure contains members for the [SelectPalette](#) enhanced metafile record. Note that the *bForceBackground* parameter in [SelectPalette](#) is always recorded as **TRUE**, which causes the palette to be realized as a background palette.

Syntax

```
typedef struct tagEMRSELECTPALETTE {  
    EMR     emr;  
    DWORD   ihPal;  
} EMRSELECTPALETTE, *PEMRSELECTPALETTE;
```

Members

emr

The base structure for all record types.

ihPal

Index to logical palette in the handle table.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[SelectPalette](#)

EMRSETARCDIRECTION structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETARCDIRECTION** structure contains members for the **SetArcDirection** enhanced metafile record.

Syntax

```
typedef struct tagEMRSETARCDIRECTION {  
    EMR     emr;  
    DWORD   iArcDirection;  
} EMRSETARCDIRECTION, *PEMRSETARCDIRECTION;
```

Members

`emr`

The base structure for all record types.

`iArcDirection`

Arc direction. This member can be either the `AD_CLOCKWISE` or `AD_COUNTERCLOCKWISE` value.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Arc](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[SetArcDirection](#)

EMRSETBKCOLOR structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETBKCOLOR** and **EMRSETTEXTCOLOR** structures contain members for the **SetBkColor** and **SetTextColor** enhanced metafile records.

Syntax

```
typedef struct tagEMRSETTEXTCOLOR {  
    EMR      emr;  
    COLORREF crColor;  
} EMRSETBKCOLOR, *PEMRSETBKCOLOR, EMRSETTEXTCOLOR, *PEMRSETTEXTCOLOR;
```

Members

emr

Base structure for all record types.

crColor

Color value. To make a **COLORREF** value, use the **RGB** macro.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[COLORREF](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RGB](#)

EMRSETCOLORADJUSTMENT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETCOLORADJUSTMENT** structure contains members for the **SetColorAdjustment** enhanced metafile record.

Syntax

```
typedef struct tagEMRSETCOLORADJUSTMENT {  
    EMR           emr;  
    COLORADJUSTMENT ColorAdjustment;  
} EMRSETCOLORADJUSTMENT, *PEMRSETCOLORADJUSTMENT;
```

Members

emr

The base structure for all record types.

ColorAdjustment

A [COLORADJUSTMENT](#) structure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[SetColorAdjustment](#)

EMRSETCOLORSPACE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETCOLORSPACE**, **EMRSELECTCOLORSPACE**, and **EMRDELETECOLORSPACE** structures contain members for the **SetColorSpace** and **DeleteColorSpace** enhanced metafile records.

Syntax

```
typedef struct tagEMRSETCOLORSPACE {  
    EMR     emr;  
    DWORD   ihCS;  
} EMRSETCOLORSPACE, *PEMRSETCOLORSPACE, EMRSELECTCOLORSPACE, *PEMRSELECTCOLORSPACE, EMRDELETECOLORSPACE,  
*PEMRDELETECOLORSPACE;
```

Members

emr

Base structure for all record types.

ihCS

Color space handle index.

Remarks

There is no function that generates an enhanced metafile record with the **EMRSELECTCOLORSPACE** structure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRSETDIBITSTODEVICE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETDIBITSTODEVICE** structure contains members for the [SetDIBitsToDevice](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRSETDIBITSTODEVICE {  
    EMR     emr;  
    RECTL  rclBounds;  
    LONG   xDest;  
    LONG   yDest;  
    LONG   xSrc;  
    LONG   ySrc;  
    LONG   cxSrc;  
    LONG   cySrc;  
    DWORD  offBmiSrc;  
    DWORD  cbBmiSrc;  
    DWORD  offBitsSrc;  
    DWORD  cbBitsSrc;  
    DWORD  iUsageSrc;  
    DWORD  iStartScan;  
    DWORD  cScans;  
} EMRSETDIBITSTODEVICE, *PEMRSETDIBITSTODEVICE;
```

Members

emr

The base structure for all record types.

rclBounds

Bounding rectangle, in device units.

xDest

Logical x-coordinate of the upper-left corner of the destination rectangle.

yDest

Logical y-coordinate of the upper-left corner of the destination rectangle.

xSrc

Logical x-coordinate of the lower-left corner of the source device-independent bitmap (DIB).

ySrc

Logical y-coordinate of the lower-left corner of the source DIB.

cxSrc

Width of the source rectangle, in logical units.

cySrc

Height of the source rectangle, in logical units.

`offBmiSrc`

Offset to the source [BITMAPINFO](#) structure.

`cbBmiSrc`

Size of the source [BITMAPINFO](#) structure.

`offBitsSrc`

Offset to source bitmap bits.

`cbBitsSrc`

Size of source bitmap bits.

`iUsageSrc`

Value of the `bmiColors` member of the [BITMAPINFO](#) structure. The `iUsageSrc` member can be either the `DIB_PAL_COLORS` or `DIB_RGB_COLORS` value.

`iStartScan`

First scan line in the array.

`cScans`

Number of scan lines.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[SetDIBitsToDevice](#)

EMRSETICMPROFILE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETICMPROFILE** structure contains members for the [SetICMPProfile](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRSETICMPROFILE {  
    EMR     emr;  
    DWORD   dwFlags;  
    DWORD   cbName;  
    DWORD   cbData;  
    BYTE    Data[1];  
} EMRSETICMPROFILE, *PEMRSETICMPROFILE, EMRSETICMPROFILEA, *PEMRSETICMPROFILEA, EMRSETICMPROFILEW,  
*PEMRSETICMPROFILEW;
```

Members

emr

The base structure for all record types.

dwFlags

The profile flags. This member can be SETICMPROFILE_EMBEDDED (0x00000001).

cbName

The size of the desired profile name.

cbData

The size of profile data, if attached.

Data

An array that contains the profile data. The length of this array is **cbName** plus **cbData**.

Remarks

This structure is to be used during metafile playback.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[SetICMProfile](#)

EMRSETMAPPERFLAGS structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETMAPPERFLAGS** structure contains members for the **SetMapperFlags** enhanced metafile record.

Syntax

```
typedef struct tagEMRSETMAPPERFLAGS {  
    EMR     emr;  
    DWORD   dwFlags;  
} EMRSETMAPPERFLAGS, *PEMRSETMAPPERFLAGS;
```

Members

emr

The base structure for all record types.

dwFlags

Font mapper flag.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[SetMapperFlags](#)

EMRSETMITERLIMIT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETMITERLIMIT** structure contains members for the **SetMiterLimit** enhanced metafile record.

Syntax

```
typedef struct tagEMRSETMITERLIMIT {  
    EMR     emr;  
    FLOAT   eMiterLimit;  
} EMRSETMITERLIMIT, *PEMRSETMITERLIMIT;
```

Members

`emr`

The base structure for all record types.

`eMiterLimit`

New miter limit.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[SetMiterLimit](#)

EMRSETPALETTEENTRIES structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETPALETTEENTRIES** structure contains members for the [SetPaletteEntries](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRSETPALETTEENTRIES {
    EMR          emr;
    DWORD        ihPal;
    DWORD        iStart;
    DWORD        cEntries;
    PALETTEENTRY aPalEntries[1];
} EMRSETPALETTEENTRIES, *PEMRSETPALETTEENTRIES;
```

Members

emr

The base structure for all record types.

ihPal

Palette handle index.

iStart

Index of first entry to set.

cEntries

Number of entries.

aPalEntries

Array of [PALETTEENTRY](#) structures. Note that **peFlags** members in the structures do not contain any flags.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

PALETTEENTRY

[SetPaletteEntries](#)

EMRSETPIXELV structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETPIXELV** structure contains members for the [SetPixelV](#) enhanced metafile record. When an enhanced metafile is created, calls to [SetPixel](#) are also recorded in this record.

Syntax

```
typedef struct tagEMRSETPIXELV {  
    EMR      emr;  
    POINTL   ptlPixel;  
    COLORREF crColor;  
} EMRSETPIXELV, *PEMRSETPIXELV;
```

Members

`emr`

The base structure for all record types.

`ptlPixel`

Logical coordinates of pixel.

`crColor`

Color value. To make a [COLORREF](#) value, use the [RGB](#) macro.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[COLORREF](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RGB](#)

[SetPixel](#)

[SetPixelV](#)

EMRSETVIEWPORTEXTEX structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETVIEWPORTEXTEX** and **EMRSETWINDOWEXTEX** structures contains members for the **SetViewportExtEx** and **SetWindowExtEx** enhanced metafile records.

Syntax

```
typedef struct tagEMRSETVIEWPORTEXTEX {  
    EMR     emr;  
    SIZEL  szlExtent;  
} EMRSETVIEWPORTEXTEX, *PEMRSETVIEWPORTEXTEX, EMRSETWINDOWEXTEX, *PEMRSETWINDOWEXTEX;
```

Members

emr

Base structure for all record types.

szlExtent

Horizontal and vertical extents. For **SetViewportExtEx**, the extents are in device units, and for **SetWindowExtEx**, the extents are in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

EMRSETVIEWPORTORGEX structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETVIEWPORTORGEX**, **EMRSETWINDOWORGEX**, and **EMRSETBRUSHORGEX** structures contain members for the **SetViewportOrgEx**, **SetWindowOrgEx**, and **SetBrushOrgEx** enhanced metafile records.

Syntax

```
typedef struct tagEMRSETVIEWPORTORGEX {  
    EMR     emr;  
    POINTL ptlOrigin;  
} EMRSETVIEWPORTORGEX, *PEMRSETVIEWPORTORGEX, EMRSETWINDOWORGEX, *PEMRSETWINDOWORGEX, EMRSETBRUSHORGEX,  
*PEMRSETBRUSHORGEX;
```

Members

emr

Base structure for all record types.

ptlOrigin

Coordinates of the origin. For **EMRSETVIEWPORTORGEX** and **EMRSETBRUSHORGEX**, this is in device units. For **EMRSETWINDOWORGEX**, this is in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[SetBrushOrgEx](#)

[SetViewportOrgEx](#)

[SetWindowOrgEx](#)

EMRSETWORLDTRANSFORM structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSETWORLDTRANSFORM** structure contains members for the **SetWorldTransform** enhanced metafile record.

Syntax

```
typedef struct tagEMRSETWORLDTRANSFORM {  
    EMR     emr;  
    XFORM  xform;  
} EMRSETWORLDTRANSFORM, *PEMRSETWORLDTRANSFORM;
```

Members

emr

The base structure for all record types.

xform

World-space to page-space transformation data.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[SetWorldTransform](#)

EMRSTRETCHBLT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSTRETCHBLT** structure contains members for the [StretchBlt](#) enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.

Syntax

```
typedef struct tagEMRSTRETCHBLT {  
    EMR      emr;  
    RECTL   rclBounds;  
    LONG     xDest;  
    LONG     yDest;  
    LONG     cxDest;  
    LONG     cyDest;  
    DWORD    dwRop;  
    LONG     xSrc;  
    LONG     ySrc;  
    XFORM    xformSrc;  
    COLORREF crBkColorSrc;  
    DWORD    iUsageSrc;  
    DWORD    offBmiSrc;  
    DWORD    cbBmiSrc;  
    DWORD    offBitsSrc;  
    DWORD    cbBitsSrc;  
    LONG     cxSrc;  
    LONG     cySrc;  
} EMRSTRETCHBLT, *PEMRSTRETCHBLT;
```

Members

emr

The base structure for all record types.

rclBounds

Bounding rectangle, in device units.

xDest

Logical x-coordinate of the upper-left corner of the destination rectangle.

yDest

Logical y-coordinate of the upper-left corner of the destination rectangle.

cxDest

Logical width of the destination rectangle.

cyDest

Logical height of the destination rectangle.

dwRop

Raster-operation code. These codes define how the color data of the source rectangle is to be combined with the color data of the destination rectangle to achieve the final color.

xSrc

Logical x-coordinate of the upper-left corner of the source rectangle.

ySrc

Logical y-coordinate of the upper-left corner of the source rectangle.

xformSrc

World-space to page-space transformation of the source device context.

crBkColorSrc

Background color (the RGB value) of the source device context. To make a [COLORREF](#) value, use the [RGB](#) macro.

iUsageSrc

Value of the **bmiColors** member of the [BITMAPINFO](#) structure. The **iUsageSrc** member can be either the DIB_PAL_COLORS or DIB_RGB_COLORS value.

offBmiSrc

Offset to the source [BITMAPINFO](#) structure.

cbBmiSrc

Size of the source [BITMAPINFO](#) structure.

offBitsSrc

Offset to source bitmap bits.

cbBitsSrc

Size of source bitmap bits.

cxSrc

Width of the source rectangle, in logical units.

cySrc

Height of the source rectangle, in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[COLORREF](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RGB](#)

[StretchBlt](#)

EMRSTRETCHDIBITS structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRSTRETCHDIBITS** structure contains members for the [StretchDIBits](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRSTRETCHDIBITS {  
    EMR     emr;  
    RECTL  rclBounds;  
    LONG   xDest;  
    LONG   yDest;  
    LONG   xSrc;  
    LONG   ySrc;  
    LONG   cxSrc;  
    LONG   cySrc;  
    DWORD  offBmiSrc;  
    DWORD  cbBmiSrc;  
    DWORD  offBitsSrc;  
    DWORD  cbBitsSrc;  
    DWORD  iUsageSrc;  
    DWORD  dwRop;  
    LONG   cxDest;  
    LONG   cyDest;  
} EMRSTRETCHDIBITS, *PEMRSTRETCHDIBITS;
```

Members

emr

The base structure for all record types.

rclBounds

Bounding rectangle, in device units.

xDest

Logical x-coordinate of the upper-left corner of the destination rectangle.

yDest

Logical y-coordinate of the upper-left corner of the destination rectangle.

xSrc

Logical x-coordinate of the upper-left corner of the source rectangle.

ySrc

Logical y-coordinate of the upper-left corner of the source rectangle.

cxSrc

Width of the source rectangle, in logical units.

cySrc

Height of the source rectangle, in logical units.

`offBmiSrc`

Offset to the source [BITMAPINFO](#) structure.

`cbBmiSrc`

Size of the source [BITMAPINFO](#) structure.

`offBitsSrc`

Offset to source bitmap bits.

`cbBitsSrc`

Size of source bitmap bits.

`iUsageSrc`

Value of the `bmiColors` member of the [BITMAPINFO](#) structure. The `iUsageSrc` member can be either the `DIB_PAL_COLORS` or `DIB_RGB_COLORS` value.

`dwRop`

Raster-operation code. These codes define how the color data of the source rectangle is to be combined with the color data of the destination rectangle to achieve the final color.

`cxDest`

Logical width of the destination rectangle.

`cyDest`

Logical height of the destination rectangle.

Requirements

Requirements	
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[StretchDIBits](#)

EMRTEXT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRTEXT** structure contains members for text output.

Syntax

```
typedef struct tagEMRTEXT {  
    POINTL ptlReference;  
    DWORD nChars;  
    DWORD offString;  
    DWORD fOptions;  
    RECTL rcl;  
    DWORD offDx;  
} EMRTEXT, *PEMRTEXT;
```

Members

ptlReference

The logical coordinates of the reference point used to position the string.

nChars

The number of characters in the string.

offString

The offset to the string.

fOptions

A value that indicates how to use the application-defined rectangle. This member can be a combination of the ETO_CLIPPED and ETO_OPAQUE values.

rcl

An optional clipping and/or opaquing rectangle, in logical units.

offDx

The offset to the intercharacter spacing array.

Remarks

The **EMRTEXT** structure is used as a member in the **EMREXTTEXTOUT** and **EMRPOLYTEXTOUT** structures.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[POINTL](#)

[RECTL](#)

EMRTRANSPARENTBLT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EMRTRANSPARENTBLT** structure contains members for the [TransparentBLT](#) enhanced metafile record.

Syntax

```
typedef struct tagEMRTRANSPARENTBLT {  
    EMR        emr;  
    RECTL     rclBounds;  
    LONG       xDest;  
    LONG       yDest;  
    LONG       cxDest;  
    LONG       cyDest;  
    DWORD      dwRop;  
    LONG       xSrc;  
    LONG       ySrc;  
    XFORM     xformSrc;  
    COLORREF   crBkColorSrc;  
    DWORD      iUsageSrc;  
    DWORD      offBmiSrc;  
    DWORD      cbBmiSrc;  
    DWORD      offBitsSrc;  
    DWORD      cbBitsSrc;  
    LONG       cxSrc;  
    LONG       cySrc;  
} EMRTRANSPARENTBLT, *PEMRTRANSPARENTBLT;
```

Members

emr

The base structure for all record types.

rclBounds

Inclusive bounds, in device units.

xDest

Logical x coordinate of the upper-left corner of the destination rectangle.

yDest

Logical y coordinate of the upper-left corner of the destination rectangle.

cxDest

Logical width of the destination rectangle.

cyDest

Logical height of the destination rectangle.

dwRop

Stores the transparent color.

xSrc

Logical x coordinate of the upper-left corner of the source rectangle.

ySrc

Logical y coordinate of the upper-left corner of the source rectangle.

xformSrc

World-space to page-space transformation of the source device context.

crBkColorSrc

Background color (the RGB value) of the source device context. To make a [COLORREF](#) value, use the [RGB](#) macro.

iUsageSrc

Source bitmap information color table usage (DIB_RGB_COLORS).

offBmiSrc

Offset to the source [BITMAPINFO](#) structure.

cbBmiSrc

Size of the source [BITMAPINFO](#) structure.

offBitsSrc

Offset to the source bitmap bits.

cbBitsSrc

Size of the source bitmap bits.

cxSrc

Width of the source rectangle, in logical units.

cySrc

Height of the source rectangle, in logical units.

Remarks

This structure is to be used during metafile playback.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[COLORREF](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RGB](#)

[TransparentBLT](#)

EndPath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EndPath** function closes a path bracket and selects the path defined by the bracket into the specified device context.

Syntax

```
BOOL EndPath(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

A handle to the device context into which the new path is selected.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[Path Functions](#)

[Paths Overview](#)

ENHMETAHEADER structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ENHMETAHEADER** structure contains enhanced-metafile data such as the dimensions of the picture stored in the enhanced metafile, the count of records in the enhanced metafile, the resolution of the device on which the picture was created, and so on.

This structure is always the first record in an enhanced metafile.

Syntax

```
typedef struct tagENHMETAHEADER {
    DWORD iType;
    DWORD nSize;
    RECTL rclBounds;
    RECTL rclFrame;
    DWORD dSignature;
    DWORD nVersion;
    DWORD nBytes;
    DWORD nRecords;
    WORD nHandles;
    WORD sReserved;
    DWORD nDescription;
    DWORD offDescription;
    DWORD nPalEntries;
    SIZEL szlDevice;
    SIZEL szlMillimeters;
    DWORD cbPixelFormat;
    DWORD offPixelFormat;
    DWORD bOpenGL;
    SIZEL szlMicrometers;
} ENHMETAHEADER, *PENHMETAHEADER, *LPENHMETAHEADER;
```

Members

iType

The record type. This member must specify the value assigned to the EMR_HEADER constant.

nSize

The structure size, in bytes.

rclBounds

The dimensions, in device units, of the smallest rectangle that can be drawn around the picture stored in the metafile. This rectangle is supplied by graphics device interface (GDI). Its dimensions include the right and bottom edges.

rclFrame

The dimensions, in .01 millimeter units, of a rectangle that surrounds the picture stored in the metafile. This rectangle must be supplied by the application that creates the metafile. Its dimensions include the right and bottom edges.

dSignature

A signature. This member must specify the value assigned to the ENHMETA_SIGNATURE constant.

`nVersion`

The metafile version. The current version value is 0x10000.

`nBytes`

The size of the enhanced metafile, in bytes.

`nRecords`

The number of records in the enhanced metafile.

`nHandles`

The number of handles in the enhanced-metafile handle table. (Index zero in this table is reserved.)

`sReserved`

Reserved; must be zero.

`nDescription`

The number of characters in the array that contains the description of the enhanced metafile's contents. This member should be set to zero if the enhanced metafile does not contain a description string.

`offDescription`

The offset from the beginning of the **ENHMETAHEADER** structure to the array that contains the description of the enhanced metafile's contents. This member should be set to zero if the enhanced metafile does not contain a description string.

`nPalEntries`

The number of entries in the enhanced metafile's palette.

`szlDevice`

The resolution of the reference device, in pixels.

`szlMillimeters`

The resolution of the reference device, in millimeters.

`cbPixelFormat`

The size of the last recorded pixel format in a metafile. If a pixel format is set in a reference DC at the start of recording, `cbPixelFormat` is set to the size of the **PIXELFORMATDESCRIPTOR**. When no pixel format is set when a metafile is recorded, this member is set to zero. If more than a single pixel format is set, the header points to the last pixel format.

`offPixelFormat`

The offset of pixel format used when recording a metafile. If a pixel format is set in a reference DC at the start of recording or during recording, `offPixelFormat` is set to the offset of the **PIXELFORMATDESCRIPTOR** in the metafile. If no pixel format is set when a metafile is recorded, this member is set to zero. If more than a single pixel format is set, the header points to the last pixel format.

`bOpenGL`

Indicates whether any OpenGL records are present in a metafile. `bOpenGL` is a simple Boolean flag that you can use to determine whether an enhanced metafile requires OpenGL handling. When a metafile contains OpenGL

records, *bOpenGL* is TRUE; otherwise it is FALSE.

`szlMicrometers`

The size of the reference device, in micrometers.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[ENHMETARECORD](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[RECTL](#)

ENHMETARECORD structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ENHMETARECORD** structure contains data that describes a graphics device interface (GDI) function used to create part of a picture in an enhanced-format metafile.

Syntax

```
typedef struct tagENHMETARECORD {  
    DWORD iType;  
    DWORD nSize;  
    DWORD dParm[1];  
} ENHMETARECORD, *PENHMETARECORD, *L彭HMETARECORD;
```

Members

iType

The record type.

nSize

The size of the record, in bytes.

dParm

An array of parameters passed to the GDI function identified by the record.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[ENHMETAHEADER](#)

[Metafile Structures](#)

[Metafiles Overview](#)

ENHMFENUMPROC callback function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnhMetaFileProc** function is an application-defined callback function used with the [EnumEnhMetaFile](#) function. The **ENHMFENUMPROC** type defines a pointer to this callback function. **EnhMetaFileProc** is a placeholder for the application-defined function name.

Syntax

```
ENHMFENUMPROC Enhmfenumproc;

int Enhmfenumproc(
    HDC hdc,
    HANDLETABLE *lphf,
    const ENHMETARECORD *lpmr,
    int nHandles,
    LPARAM data
)
{...}
```

Parameters

hdc

lphf

lpmr

nHandles

data

Return value

This function must return a nonzero value to continue enumeration; to stop enumeration, it must return zero.

Remarks

An application must register the callback function by passing its address to the [EnumEnhMetaFile](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[ENHMETARECORD](#)

[EnumEnhMetaFile](#)

[HANDLETABLE](#)

[Metafile Functions](#)

[Metafiles Overview](#)

EnumEnhMetaFile function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumEnhMetaFile** function enumerates the records within an enhanced-format metafile by retrieving each record and passing it to the specified callback function. The application-supplied callback function processes each record as required. The enumeration continues until the last record is processed or when the callback function returns zero.

Syntax

```
BOOL EnumEnhMetaFile(
    [in] HDC         hdc,
    [in] HENHMETAFILE hmf,
    [in] ENHMFENUMPROC proc,
    [in] LPVOID      param,
    [in] const RECT   *lpRect
);
```

Parameters

[in] `hdc`

A handle to a device context. This handle is passed to the callback function.

[in] `hmf`

A handle to an enhanced metafile.

[in] `proc`

A pointer to the application-supplied callback function. For more information, see the [EnhMetaFileProc](#) function.

[in] `param`

A pointer to optional callback-function data.

[in] `lpRect`

A pointer to a [RECT](#) structure that specifies the coordinates, in logical units, of the picture's upper-left and lower-right corners.

Return value

If the callback function successfully enumerates all the records in the enhanced metafile, the return value is nonzero.

If the callback function does not successfully enumerate all the records in the enhanced metafile, the return value is zero.

Remarks

Points along the edge of the rectangle pointed to by the `lpRect` parameter are included in the picture. If the `hdc` parameter is **NULL**, the system ignores `lpRect`.

If the callback function calls the [PlayEnhMetaFileRecord](#) function, *hdc* must identify a valid device context. The system uses the device context's transformation and mapping mode to transform the picture displayed by the [PlayEnhMetaFileRecord](#) function.

You can use the [EnumEnhMetaFile](#) function to embed one enhanced-metafile within another.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnhMetaFileProc](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayEnhMetaFile](#)

[PlayEnhMetaFileRecord](#)

[RECT](#)

EnumFontFamiliesA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumFontFamilies** function enumerates the fonts in a specified font family that are available on a specified device.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the [EnumFontFamiliesEx](#) function.

Syntax

```
int EnumFontFamiliesA(
    [in] HDC         hdc,
    [in] LPCSTR     lpLogfont,
    [in] FONTENUMPROCA lpProc,
    [in] LPARAM     lParam
);
```

Parameters

[in] *hdc*

A handle to the device context from which to enumerate the fonts.

[in] *lpLogfont*

A pointer to a null-terminated string that specifies the family name of the desired fonts. If *lpszFamily* is **NULL**, **EnumFontFamilies** selects and enumerates one font of each available type family.

[in] *lpProc*

A pointer to the application defined callback function. For information, see [EnumFontFamProc](#).

[in] *lParam*

A pointer to application-supplied data. The data is passed to the callback function along with the font information.

Return value

The return value is the last value returned by the callback function. Its meaning is implementation specific.

Remarks

For each font having the typeface name specified by the *lpszFamily* parameter, the **EnumFontFamilies** function retrieves information about that font and passes it to the function pointed to by the *lpEnumFontFamProc* parameter. The application defined callback function can process the font information as desired. Enumeration continues until there are no more fonts or the callback function returns zero.

When the graphics mode on the device context is set to GM_ADVANCED using the [SetGraphicsMode](#) function

and the DEVICE_FONTPTYPE flag is passed to the FontType parameter, this function returns a list of type 1 and OpenType fonts on the system. When the graphics mode is not set to GM_ADVANCED, this function returns a list of type 1, OpenType, and TrueType fonts on the system.

The fonts for many East Asian languages have two typeface names: an English name and a localized name. [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) return the English typeface name if the system locale does not match the language of the font.

Examples

For examples, see [Enumerating the Installed Fonts](#).

NOTE

The wingdi.h header defines EnumFontFamilies as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnumFontFamProc](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

EnumFontFamiliesExA function (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **EnumFontFamiliesEx** function enumerates all uniquely-named fonts in the system that match the font characteristics specified by the [LOGFONT](#) structure. **EnumFontFamiliesEx** enumerates fonts based on typeface name, character set, or both.

Syntax

```
int EnumFontFamiliesExA(
    [in] HDC             hdc,
    [in] LPLOGFONTA     lpLogfont,
    [in] FONTENUMPROCA  lpProc,
    [in] LPARAM          lParam,
    DWORD              dwFlags
);
```

Parameters

[in] **hdc**

A handle to the device context from which to enumerate the fonts.

[in] **lpLogfont**

A pointer to a [LOGFONT](#) structure that contains information about the fonts to enumerate. The function examines the following members.

MEMBER	DESCRIPTION
lfCharSet	If set to DEFAULT_CHARSET , the function enumerates all uniquely-named fonts in all character sets. (If there are two fonts with the same name, only one is enumerated.) If set to a valid character set value, the function enumerates only fonts in the specified character set.
lfFaceName	If set to an empty string, the function enumerates one font in each available typeface name. If set to a valid typeface name, the function enumerates all fonts with the specified name.
lfPitchAndFamily	Must be set to zero for all language versions of the operating system.

[in] **lpProc**

A pointer to the application defined callback function. For more information, see the [EnumFontFamExProc](#) function.

[in] **lParam**

An application defined value. The function passes this value to the callback function along with font information.

dwFlags

This parameter is not used and must be zero.

Return value

The return value is the last value returned by the callback function. This value depends on which font families are available for the specified device.

Remarks

The **EnumFontFamiliesEx** function does not use tagged typeface names to identify character sets. Instead, it always passes the correct typeface name and a separate character set value to the callback function. The function enumerates fonts based on the values of the **IfCharSet** and **IfFaceName** members in the **LOGFONT** structure.

As with [EnumFontFamilies](#), **EnumFontFamiliesEx** enumerates all font styles. Not all styles of a font cover the same character sets. For example, Fontorama Bold might contain ANSI, Greek, and Cyrillic characters, but Fontorama Italic might contain only ANSI characters. For this reason, it's best not to assume that a specified font covers a specific character set, even if it is the ANSI character set. The following table shows the results of various combinations of values for **IfCharSet** and **IfFaceName**.

VALUES	MEANING
IfCharSet = DEFAULT_CHARSET IfFaceName = '\0'	Enumerates all uniquely-named fonts within all character sets. If there are two fonts with the same name, only one is enumerated.
IfCharSet = DEFAULT_CHARSET IfFaceName = a specific font	Enumerates all character sets and styles in a specific font.
IfCharSet = a specific character set IfFaceName = '\0'	Enumerates all styles of all fonts in the specific character set.
IfCharSet = a specific character set IfFaceName = a specific font	Enumerates all styles of a font in a specific character set.

The following code sample shows how these values are used.

```
// To enumerate all styles and charsets of all fonts:  
lf.lfFaceName[0] = '\0';  
lf.lfCharSet = DEFAULT_CHARSET;  
HRESULT hr;  
  
// To enumerate all styles and character sets of the Arial font:  
hr = StringCchCopy( (LPSTR)lf.lfFaceName, LF_FACESIZE, "Arial" );  
if (FAILED(hr))  
{  
    // TODO: write error handler  
}  
  
lf.lfCharSet = DEFAULT_CHARSET;
```

```

// To enumerate all styles of all fonts for the ANSI character set
lf.lfFaceName[0] = '\0';
lf.lfCharSet = ANSI_CHARSET;

// To enumerate all styles of Arial font that cover the ANSI charset
hr = StringCchCopy( (LPSTR)lf.lfFaceName, LF_FACESIZE, "Arial" );
if (FAILED(hr))
{
// TODO: write error handler
}

lf.lfCharSet = ANSI_CHARSET;

```

The callback functions for [EnumFontFamilies](#) and [EnumFontFamiliesEx](#) are very similar. The main difference is that the [ENUMLOGFONTEX](#) structure includes a script field.

Note, based on the values of [lfCharSet](#) and [lfFaceName](#), [EnumFontFamiliesEx](#) will enumerate the same font as many times as there are distinct character sets in the font. This can create an extensive list of fonts which can be burdensome to a user. For example, the Century Schoolbook font can appear for the Baltic, Western, Greek, Turkish, and Cyrillic character sets. To avoid this, an application should filter the list of fonts.

The fonts for many East Asian languages have two typeface names: an English name and a localized name. [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) return the English typeface name if the system locale does not match the language of the font.

When the graphics mode on the device context is set to GM_ADVANCED using the SetGraphicsMode function and the DEVICE_FONTPTYPE flag is passed to the FontType parameter, this function returns a list of type 1 and OpenType fonts on the system. When the graphics mode is not set to GM_ADVANCED, this function returns a list of type 1, OpenType, and TrueType fonts on the system.

NOTE

The wingdi.h header defines [EnumFontFamiliesEx](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnumFontFamExProc](#)

[EnumFontFamilies](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

EnumFontFamiliesExW function (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **EnumFontFamiliesEx** function enumerates all uniquely-named fonts in the system that match the font characteristics specified by the [LOGFONT](#) structure. **EnumFontFamiliesEx** enumerates fonts based on typeface name, character set, or both.

Syntax

```
int EnumFontFamiliesExW(
    [in] HDC             hdc,
    [in] LPLOGFONTW     lpLogfont,
    [in] FONTENUMPROCW lpProc,
    [in] LPARAM          lParam,
    DWORD              dwFlags
);
```

Parameters

[in] **hdc**

A handle to the device context from which to enumerate the fonts.

[in] **lpLogfont**

A pointer to a [LOGFONT](#) structure that contains information about the fonts to enumerate. The function examines the following members.

MEMBER	DESCRIPTION
IfCharSet	If set to DEFAULT_CHARSET , the function enumerates all uniquely-named fonts in all character sets. (If there are two fonts with the same name, only one is enumerated.) If set to a valid character set value, the function enumerates only fonts in the specified character set.
IfFaceName	If set to an empty string, the function enumerates one font in each available typeface name. If set to a valid typeface name, the function enumerates all fonts with the specified name.
IfPitchAndFamily	Must be set to zero for all language versions of the operating system.

[in] **lpProc**

A pointer to the application defined callback function. For more information, see the [EnumFontFamExProc](#) function.

[in] **lParam**

An application defined value. The function passes this value to the callback function along with font information.

dwFlags

This parameter is not used and must be zero.

Return value

The return value is the last value returned by the callback function. This value depends on which font families are available for the specified device.

Remarks

The **EnumFontFamiliesEx** function does not use tagged typeface names to identify character sets. Instead, it always passes the correct typeface name and a separate character set value to the callback function. The function enumerates fonts based on the values of the **IfCharSet** and **IfFaceName** members in the **LOGFONT** structure.

As with [EnumFontFamilies](#), **EnumFontFamiliesEx** enumerates all font styles. Not all styles of a font cover the same character sets. For example, Fontorama Bold might contain ANSI, Greek, and Cyrillic characters, but Fontorama Italic might contain only ANSI characters. For this reason, it's best not to assume that a specified font covers a specific character set, even if it is the ANSI character set. The following table shows the results of various combinations of values for **IfCharSet** and **IfFaceName**.

VALUES	MEANING
IfCharSet = DEFAULT_CHARSET IfFaceName = '\0'	Enumerates all uniquely-named fonts within all character sets. If there are two fonts with the same name, only one is enumerated.
IfCharSet = DEFAULT_CHARSET IfFaceName = a specific font	Enumerates all character sets and styles in a specific font.
IfCharSet = a specific character set IfFaceName = '\0'	Enumerates all styles of all fonts in the specific character set.
IfCharSet = a specific character set IfFaceName = a specific font	Enumerates all styles of a font in a specific character set.

The following code sample shows how these values are used.

```
// To enumerate all styles and charsets of all fonts:  
lf.lfFaceName[0] = '\0';  
lf.lfCharSet = DEFAULT_CHARSET;  
HRESULT hr;  
  
// To enumerate all styles and character sets of the Arial font:  
hr = StringCchCopy( (LPSTR)lf.lfFaceName, LF_FACESIZE, "Arial" );  
if (FAILED(hr))  
{  
    // TODO: write error handler  
}  
  
lf.lfCharSet = DEFAULT_CHARSET;
```

```

// To enumerate all styles of all fonts for the ANSI character set
lf.lfFaceName[0] = '\0';
lf.lfCharSet = ANSI_CHARSET;

// To enumerate all styles of Arial font that cover the ANSI charset
hr = StringCchCopy( (LPSTR)lf.lfFaceName, LF_FACESIZE, "Arial" );
if (FAILED(hr))
{
// TODO: write error handler
}

lf.lfCharSet = ANSI_CHARSET;

```

The callback functions for [EnumFontFamilies](#) and [EnumFontFamiliesEx](#) are very similar. The main difference is that the [ENUMLOGFONTEX](#) structure includes a script field.

Note, based on the values of [lfCharSet](#) and [lfFaceName](#), [EnumFontFamiliesEx](#) will enumerate the same font as many times as there are distinct character sets in the font. This can create an extensive list of fonts which can be burdensome to a user. For example, the Century Schoolbook font can appear for the Baltic, Western, Greek, Turkish, and Cyrillic character sets. To avoid this, an application should filter the list of fonts.

The fonts for many East Asian languages have two typeface names: an English name and a localized name. [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) return the English typeface name if the system locale does not match the language of the font.

When the graphics mode on the device context is set to GM_ADVANCED using the [SetGraphicsMode](#) function and the DEVICE_FONTPTYPE flag is passed to the [FontType](#) parameter, this function returns a list of type 1 and OpenType fonts on the system. When the graphics mode is not set to GM_ADVANCED, this function returns a list of type 1, OpenType, and TrueType fonts on the system.

NOTE

The wingdi.h header defines [EnumFontFamiliesEx](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the [UNICODE](#) preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnumFontFamExProc](#)

[EnumFontFamilies](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

EnumFontFamiliesW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumFontFamilies** function enumerates the fonts in a specified font family that are available on a specified device.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the [EnumFontFamiliesEx](#) function.

Syntax

```
int EnumFontFamiliesW(
    [in] HDC         hdc,
    [in] LPCWSTR     lpLogfont,
    [in] FONTENUMPROCW lpProc,
    [in] LPARAM      lParam
);
```

Parameters

[in] *hdc*

A handle to the device context from which to enumerate the fonts.

[in] *lpLogfont*

A pointer to a null-terminated string that specifies the family name of the desired fonts. If *lpszFamily* is **NULL**, **EnumFontFamilies** selects and enumerates one font of each available type family.

[in] *lpProc*

A pointer to the application defined callback function. For information, see [EnumFontFamProc](#).

[in] *lParam*

A pointer to application-supplied data. The data is passed to the callback function along with the font information.

Return value

The return value is the last value returned by the callback function. Its meaning is implementation specific.

Remarks

For each font having the typeface name specified by the *lpszFamily* parameter, the **EnumFontFamilies** function retrieves information about that font and passes it to the function pointed to by the *lpEnumFontFamProc* parameter. The application defined callback function can process the font information as desired. Enumeration continues until there are no more fonts or the callback function returns zero.

When the graphics mode on the device context is set to GM_ADVANCED using the [SetGraphicsMode](#) function

and the DEVICE_FONTPTYPE flag is passed to the FontType parameter, this function returns a list of type 1 and OpenType fonts on the system. When the graphics mode is not set to GM_ADVANCED, this function returns a list of type 1, OpenType, and TrueType fonts on the system.

The fonts for many East Asian languages have two typeface names: an English name and a localized name. [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) return the English typeface name if the system locale does not match the language of the font.

Examples

For examples, see [Enumerating the Installed Fonts](#).

NOTE

The wingdi.h header defines EnumFontFamilies as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnumFontFamProc](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

EnumFontsA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumFonts** function enumerates the fonts available on a specified device. For each font with the specified typeface name, the **EnumFonts** function retrieves information about that font and passes it to the application defined callback function. This callback function can process the font information as desired. Enumeration continues until there are no more fonts or the callback function returns zero.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the [EnumFontFamiliesEx](#) function.

Syntax

```
int EnumFontsA(
    [in] HDC         hdc,
    [in] LPCSTR     lpLogfont,
    [in] FONTENUMPROCA lpProc,
    [in] LPARAM      lParam
);
```

Parameters

[in] `hdc`

A handle to the device context from which to enumerate the fonts.

[in] `lpLogfont`

A pointer to a null-terminated string that specifies the typeface name of the desired fonts. If *lpFaceName* is `NULL`, **EnumFonts** randomly selects and enumerates one font of each available typeface.

[in] `lpProc`

A pointer to the application definedcallback function. For more information, see [EnumFontsProc](#).

[in] `lParam`

A pointer to any application-defined data. The data is passed to the callback function along with the font information.

Return value

The return value is the last value returned by the callback function. Its meaning is defined by the application.

Remarks

Use [EnumFontFamiliesEx](#) instead of **EnumFonts**. The [EnumFontFamiliesEx](#) function differs from the **EnumFonts** function in that it retrieves the style names associated with a TrueType font. With [EnumFontFamiliesEx](#), you can retrieve information about font styles that cannot be enumerated using the **EnumFonts** function.

The fonts for many East Asian languages have two typeface names: an English name and a localized name. [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) return the English typeface name if the system locale does not match the language of the font.

NOTE

The wingdi.h header defines EnumFonts as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFontsProc](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetDeviceCaps](#)

EnumFontsW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumFonts** function enumerates the fonts available on a specified device. For each font with the specified typeface name, the **EnumFonts** function retrieves information about that font and passes it to the application defined callback function. This callback function can process the font information as desired. Enumeration continues until there are no more fonts or the callback function returns zero.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the [EnumFontFamiliesEx](#) function.

Syntax

```
int EnumFontsW(
    [in] HDC         hdc,
    [in] LPCWSTR     lpLogfont,
    [in] FONTENUMPROCW lpProc,
    [in] LPARAM      lParam
);
```

Parameters

[in] `hdc`

A handle to the device context from which to enumerate the fonts.

[in] `lpLogfont`

A pointer to a null-terminated string that specifies the typeface name of the desired fonts. If *lpFaceName* is `NULL`, **EnumFonts** randomly selects and enumerates one font of each available typeface.

[in] `lpProc`

A pointer to the application definedcallback function. For more information, see [EnumFontsProc](#).

[in] `lParam`

A pointer to any application-defined data. The data is passed to the callback function along with the font information.

Return value

The return value is the last value returned by the callback function. Its meaning is defined by the application.

Remarks

Use [EnumFontFamiliesEx](#) instead of **EnumFonts**. The [EnumFontFamiliesEx](#) function differs from the **EnumFonts** function in that it retrieves the style names associated with a TrueType font. With [EnumFontFamiliesEx](#), you can retrieve information about font styles that cannot be enumerated using the **EnumFonts** function.

The fonts for many East Asian languages have two typeface names: an English name and a localized name. [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) return the English typeface name if the system locale does not match the language of the font.

NOTE

The wingdi.h header defines EnumFonts as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFontsProc](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetDeviceCaps](#)

ENUMLOGFONTA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ENUMLOGFONT** structure defines the attributes of a font, the complete name of a font, and the style of a font.

Syntax

```
typedef struct tagENUMLOGFONTA {
    LOGFONTA elfLogFont;
    BYTE      elfFullName[LF_FULLFACESIZE];
    BYTE      elfStyle[LF_FACESIZE];
} ENUMLOGFONTA, *LPENUMLOGFONTA;
```

Members

elfLogFont

A **LOGFONT** structure that defines the attributes of a font.

elfFullName

A unique name for the font. For example, ABCD Font Company TrueType Bold Italic Sans Serif.

elfStyle

The style of the font. For example, Bold Italic.

Remarks

NOTE

The wingdi.h header defines **ENUMLOGFONT** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EnumFontFamProc](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

ENUMLOGFONTEXA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ENUMLOGFONTEX** structure contains information about an enumerated font.

Syntax

```
typedef struct tagENUMLOGFONTEXA {
    LOGFONTA elfLogFont;
    BYTE     elfFullName[LF_FULLFACESIZE];
    BYTE     elfStyle[LF_FACESIZE];
    BYTE     elfScript[LF_FACESIZE];
} ENUMLOGFONTEXA, *LPENUMLOGFONTEXA;
```

Members

`elfLogFont`

A [LOGFONT](#) structure that contains values defining the font attributes.

`elfFullName`

The unique name of the font. For example, ABC Font Company TrueType Bold Italic Sans Serif.

`elfStyle`

The style of the font. For example, Bold Italic.

`elfScript`

The script, that is, the character set, of the font. For example, Cyrillic.

Remarks

NOTE

The wingdi.h header defines **ENUMLOGFONTEX** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EnumFontFamExProc](#)

[EnumFontFamiliesEx](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

ENUMLOGFONTEXDVA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ENUMLOGFONTEXDV** structure contains the information used to create a font.

Syntax

```
typedef struct tagENUMLOGFONTEXDVA {  
    ENUMLOGFONTEXA elfEnumLogfontEx;  
    DESIGNVECTOR    elfDesignVector;  
} ENUMLOGFONTEXDVA, *PENUMLOGFONTEXDVA, *LPENUMLOGFONTEXDVA;
```

Members

`elfEnumLogfontEx`

An [ENUMLOGFONTEX](#) structure that contains information about the logical attributes of the font.

`elfDesignVector`

A [DESIGNVECTOR](#) structure. This is zero-filled unless the font described is a multiple master OpenType font.

Remarks

The actual size of **ENUMLOGFONTEXDV** depends on that of [DESIGNVECTOR](#), which, in turn depends on its `dvNumAxes` member.

The [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) functions have been modified to return pointers to [ENUMTEXTMETRIC](#) and **ENUMLOGFONTEXDV** to the callback function.

NOTE

The wingdi.h header defines **ENUMLOGFONTEXDV** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[CreateFontIndirectEx](#)

[DESIGNVECTOR](#)

[ENUMTEXTMETRIC](#)

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

ENUMLOGFONTEXDVW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ENUMLOGFONTEXDV** structure contains the information used to create a font.

Syntax

```
typedef struct tagENUMLOGFONTEXDVW {  
    ENUMLOGFONTEXW elfEnumLogfontEx;  
    DESIGNVECTOR    elfDesignVector;  
} ENUMLOGFONTEXDVW, *PENUMLOGFONTEXDVW, *LPENUMLOGFONTEXDVW;
```

Members

`elfEnumLogfontEx`

An [ENUMLOGFONTEX](#) structure that contains information about the logical attributes of the font.

`elfDesignVector`

A [DESIGNVECTOR](#) structure. This is zero-filled unless the font described is a multiple master OpenType font.

Remarks

The actual size of **ENUMLOGFONTEXDV** depends on that of [DESIGNVECTOR](#), which, in turn depends on its `dvNumAxes` member.

The [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) functions have been modified to return pointers to [ENUMTEXTMETRIC](#) and **ENUMLOGFONTEXDV** to the callback function.

NOTE

The wingdi.h header defines **ENUMLOGFONTEXDV** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[CreateFontIndirectEx](#)

[DESIGNVECTOR](#)

[ENUMTEXTMETRIC](#)

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

ENUMLOGFONTEXW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ENUMLOGFONTEX** structure contains information about an enumerated font.

Syntax

```
typedef struct tagENUMLOGFONTEXW {
    LOGFONTW elfLogFont;
    WCHAR     elfFullName[LF_FULLFACESIZE];
    WCHAR     elfStyle[LF_FACESIZE];
    WCHAR     elfScript[LF_FACESIZE];
} ENUMLOGFONTEXW, *LPENUMLOGFONTEXW;
```

Members

`elfLogFont`

A [LOGFONT](#) structure that contains values defining the font attributes.

`elfFullName`

The unique name of the font. For example, ABC Font Company TrueType Bold Italic Sans Serif.

`elfStyle`

The style of the font. For example, Bold Italic.

`elfScript`

The script, that is, the character set, of the font. For example, Cyrillic.

Remarks

NOTE

The wingdi.h header defines **ENUMLOGFONTEX** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EnumFontFamExProc](#)

[EnumFontFamiliesEx](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

ENUMLOGFONTW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ENUMLOGFONT** structure defines the attributes of a font, the complete name of a font, and the style of a font.

Syntax

```
typedef struct tagENUMLOGFONTW {  
    LOGFONTW elfLogFont;  
    WCHAR     elfFullName[LF_FULLFACESIZE];  
    WCHAR     elfStyle[LF_FACESIZE];  
} ENUMLOGFONTW, *LPENUMLOGFONTW;
```

Members

elfLogFont

A [LOGFONT](#) structure that defines the attributes of a font.

elfFullName

A unique name for the font. For example, ABCD Font Company TrueType Bold Italic Sans Serif.

elfStyle

The style of the font. For example, Bold Italic.

Remarks

NOTE

The wingdi.h header defines **ENUMLOGFONT** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EnumFontFamProc](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

EnumMetaFile function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumMetaFile** function enumerates the records within a Windows-format metafile by retrieving each record and passing it to the specified callback function. The application-supplied callback function processes each record as required. The enumeration continues until the last record is processed or when the callback function returns zero.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [EnumEnhMetaFile](#).

Syntax

```
BOOL EnumMetaFile(
    [in] HDC      hdc,
    [in] HMETAFILE  hmf,
    [in] MFENUMPROC proc,
    [in] LPARAM    param
);
```

Parameters

[in] hdc

Handle to a device context. This handle is passed to the callback function.

[in] hmf

Handle to a Windows-format metafile.

[in] proc

Pointer to an application-supplied callback function. For more information, see [EnumMetaFileProc](#).

[in] param

Pointer to optional data.

Return value

If the callback function successfully enumerates all the records in the Windows-format metafile, the return value is nonzero.

If the callback function does not successfully enumerate all the records in the Windows-format metafile, the return value is zero.

Remarks

To convert a Windows-format metafile into an enhanced-format metafile, use the [SetWinMetaFileBits](#) function.

You can use the **EnumMetaFile** function to embed one Windows-format metafile within another.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnumEnhMetaFile](#)

[EnumMetaFileProc](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayMetaFile](#)

[PlayMetaFileRecord](#)

[SetWinMetaFileBits](#)

EnumObjects function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumObjects** function enumerates the pens or brushes available for the specified device context (DC). This function calls the application-defined callback function once for each available object, supplying data describing that object. **EnumObjects** continues calling the callback function until the callback function returns zero or until all of the objects have been enumerated.

Syntax

```
int EnumObjects(
    [in] HDC         hdc,
    [in] int          nType,
    [in] GOBJENUMPROC lpFunc,
    [in] LPARAM       lParam
);
```

Parameters

[in] hdc

A handle to the DC.

[in] nType

The object type. This parameter can be OBJ_BRUSH or OBJ_PEN.

[in] lpFunc

A pointer to the application-defined callback function. For more information about the callback function, see the [EnumObjectsProc](#) function.

[in] lParam

A pointer to the application-defined data. The data is passed to the callback function along with the object information.

Return value

If the function succeeds, the return value is the last value returned by the callback function. Its meaning is user-defined.

If the objects cannot be enumerated (for example, there are too many objects), the function returns zero without calling the callback function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumObjectsProc](#)

[GetObject](#)

ENUMTEXTMETRICA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef struct tagENUMTEXTMETRICA {  
    NEWTEXTMETRICEXA etmNewTextMetricEx;  
    AXESLISTA       etmAxesList;  
} ENUMTEXTMETRICA, *PENUMTEXTMETRICA, *LPENUMTEXTMETRICA;
```

Members

etmNewTextMetricEx

A [NEWTEXTMETRICEX](#) structure, containing information about a physical font.

etmAxesList

An [AXESLIST](#) structure, containing information about the axes for the font. This is only used for multiple master fonts.

Remarks

ENUMTEXTMETRIC is an extension of [NEWTEXTMETRICEX](#) that includes the axis information for a multiple master font.

The [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) functions have been modified to return pointers to the **ENUMTEXTMETRIC** and [ENUMLOGFONTEXDV](#) structures.

NOTE

The wingdi.h header defines **ENUMTEXTMETRIC** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[AXESLIST](#)

[ENUMLOGFONTEX](#)

[ENUMLOGFONTEXDV](#)

[ENUMTEXTMETRIC](#)

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[NEWTEXTMETRICEX](#)

ENUMTEXTMETRICW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef struct tagENUMTEXTMETRICW {  
    NEWTEXTMETRICEXW etmNewTextMetricEx;  
    AXESLISTW       etmAxesList;  
} ENUMTEXTMETRICW, *PENUMTEXTMETRICW, *LPENUMTEXTMETRICW;
```

Members

etmNewTextMetricEx

A [NEWTEXTMETRICEX](#) structure, containing information about a physical font.

etmAxesList

An [AXESLIST](#) structure, containing information about the axes for the font. This is only used for multiple master fonts.

Remarks

ENUMTEXTMETRIC is an extension of [NEWTEXTMETRICEX](#) that includes the axis information for a multiple master font.

The [EnumFonts](#), [EnumFontFamilies](#), and [EnumFontFamiliesEx](#) functions have been modified to return pointers to the **ENUMTEXTMETRIC** and [ENUMLOGFONTEXDV](#) structures.

NOTE

The wingdi.h header defines **ENUMTEXTMETRIC** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[AXESLIST](#)

[ENUMLOGFONTEX](#)

[ENUMLOGFONTEXDV](#)

[ENUMTEXTMETRIC](#)

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFonts](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[NEWTEXTMETRICEX](#)

EqualRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EqualRgn** function checks the two specified regions to determine whether they are identical. The function considers two regions identical if they are equal in size and shape.

Syntax

```
BOOL EqualRgn(
    [in] HRGN hrgn1,
    [in] HRGN hrgn2
);
```

Parameters

[in] `hrgn1`

Handle to a region.

[in] `hrgn2`

Handle to a region.

Return value

If the two regions are equal, the return value is nonzero.

If the two regions are not equal, the return value is zero. A return value of ERROR means at least one of the region handles is invalid.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateRectRgn](#)

[CreateRectRgnIndirect](#)

[Region Functions](#)

[Regions Overview](#)

ExcludeClipRect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ExcludeClipRect** function creates a new clipping region that consists of the existing clipping region minus the specified rectangle.

Syntax

```
int ExcludeClipRect(
    [in] HDC hdc,
    [in] int left,
    [in] int top,
    [in] int right,
    [in] int bottom
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `left`

The x-coordinate, in logical units, of the upper-left corner of the rectangle.

[in] `top`

The y-coordinate, in logical units, of the upper-left corner of the rectangle.

[in] `right`

The x-coordinate, in logical units, of the lower-right corner of the rectangle.

[in] `bottom`

The y-coordinate, in logical units, of the lower-right corner of the rectangle.

Return value

The return value specifies the new clipping region's complexity; it can be one of the following values.

RETURN CODE	DESCRIPTION
<code>NULLREGION</code>	Region is empty.
<code>SIMPLEREGION</code>	Region is a single rectangle.
<code>COMPLEXREGION</code>	Region is more than one rectangle.

ERROR	No region was created.
-------	------------------------

Remarks

The lower and right edges of the specified rectangle are not excluded from the clipping region.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Clipping Functions](#)

[Clipping Overview](#)

[IntersectClipRect](#)

ExtCreatePen function (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **ExtCreatePen** function creates a logical cosmetic or geometric pen that has the specified style, width, and brush attributes.

Syntax

```
HPEN ExtCreatePen(
    [in] DWORD         iPenStyle,
    [in] DWORD         cWidth,
    [in] const LOGBRUSH *plbrush,
    [in] DWORD         cStyle,
    [in] const DWORD   *pstyle
);
```

Parameters

[in] **iPenStyle**

A combination of type, style, end cap, and join attributes. The values from each category are combined by using the bitwise OR operator (|).

The pen type can be one of the following values.

VALUE	MEANING
PS_GEOMETRIC	The pen is geometric.
PS_COSMETIC	The pen is cosmetic.

The pen style can be one of the following values.

VALUE	MEANING
PS_ALTERNATE	The pen sets every other pixel. (This style is applicable only for cosmetic pens.)
PS_SOLID	The pen is solid.
PS_DASH	The pen is dashed.
PS_DOT	The pen is dotted.

PS_DASHDOT	The pen has alternating dashes and dots.
PS_DASHDOTDOT	The pen has alternating dashes and double dots.
PS_NULL	The pen is invisible.
PS_USERSTYLE	The pen uses a styling array supplied by the user.
PS_INSIDEFRAME	The pen is solid. When this pen is used in any GDI drawing function that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to geometric pens.

The end cap is only specified for geometric pens. The end cap can be one of the following values.

VALUE	MEANING
PS_ENDCAP_ROUND	End caps are round.
PS_ENDCAP_SQUARE	End caps are square.
PS_ENDCAP_FLAT	End caps are flat.

The join is only specified for geometric pens. The join can be one of the following values.

VALUE	MEANING
PS_JOIN_BEVEL	Joins are beveled.
PS_JOIN_MITER	Joins are mitered when they are within the current limit set by the SetMiterLimit function. If it exceeds this limit, the join is beveled.
PS_JOIN_ROUND	Joins are round.

[in] cWidth

The width of the pen. If the *dwPenStyle* parameter is PS_GEOMETRIC, the width is given in logical units. If *dwPenStyle* is PS_COSMETIC, the width must be set to 1.

[in] pBrush

A pointer to a [LOGBRUSH](#) structure. If *dwPenStyle* is PS_COSMETIC, the **IbColor** member specifies the color of the pen and the **IpStyle** member must be set to BS_SOLID. If *dwPenStyle* is PS_GEOMETRIC, all members must be used to specify the brush attributes of the pen.

[in] *cStyle*

The length, in **DWORD** units, of the *lpStyle* array. This value must be zero if *dwPenStyle* is not PS_USERSTYLE.

The style count is limited to 16.

[in] *pstyle*

A pointer to an array. The first value specifies the length of the first dash in a user-defined style, the second value specifies the length of the first space, and so on. This pointer must be **NULL** if *dwPenStyle* is not PS_USERSTYLE.

If the *lpStyle* array is exceeded during line drawing, the pointer is reset to the beginning of the array. When this happens and *dwStyleCount* is an even number, the pattern of dashes and spaces repeats. However, if *dwStyleCount* is odd, the pattern reverses when the pointer is reset -- the first element of *lpStyle* now refers to spaces, the second refers to dashes, and so forth.

Return value

If the function succeeds, the return value is a handle that identifies a logical pen.

If the function fails, the return value is zero.

Remarks

A geometric pen can have any width and can have any of the attributes of a brush, such as dithers and patterns. A cosmetic pen can only be a single pixel wide and must be a solid color, but cosmetic pens are generally faster than geometric pens.

The width of a geometric pen is always specified in world units. The width of a cosmetic pen is always 1.

End caps and joins are only specified for geometric pens.

After an application creates a logical pen, it can select that pen into a device context by calling the [SelectObject](#) function. After a pen is selected into a device context, it can be used to draw lines and curves.

If *dwPenStyle* is PS_COSMETIC and PS_USERSTYLE, the entries in the *lpStyle* array specify lengths of dashes and spaces in style units. A style unit is defined by the device where the pen is used to draw a line.

If *dwPenStyle* is PS_GEOMETRIC and PS_USERSTYLE, the entries in the *lpStyle* array specify lengths of dashes and spaces in logical units.

If *dwPenStyle* is PS_ALTERNATE, the style unit is ignored and every other pixel is set.

If the **IbStyle** member of the [LOGBRUSH](#) structure pointed to by *lpIb* is BS_PATTERN, the bitmap pointed to by the **IbHatch** member of that structure cannot be a DIB section. A DIB section is a bitmap created by [CreateDIBSection](#). If that bitmap is a DIB section, the [ExtCreatePen](#) function fails.

When an application no longer requires a specified pen, it should call the [DeleteObject](#) function to delete the pen.

ICM: No color management is done at pen creation. However, color management is performed when the pen is selected into an ICM-enabled device context.

Examples

For an example, see [Using Pens](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateDIBSection](#)

[CreatePen](#)

[CreatePenIndirect](#)

[DeleteObject](#)

[GetObject](#)

[LOGBRUSH](#)

[Pen Functions](#)

[Pens Overview](#)

[SelectObject](#)

[SetMiterLimit](#)

ExtCreateRegion function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ExtCreateRegion** function creates a region from the specified region and transformation data.

Syntax

```
HRGN ExtCreateRegion(
    [in] const XFORM    *lpx,
    [in] DWORD         nCount,
    [in] const RGNDATA *lpData
);
```

Parameters

[in] *lpx*

A pointer to an [XFORM](#) structure that defines the transformation to be performed on the region. If this pointer is **NULL**, the identity transformation is used.

[in] *nCount*

The number of bytes pointed to by */pRgnData*.

[in] *lpData*

A pointer to a [RGNDATA](#) structure that contains the region data in logical units.

Return value

If the function succeeds, the return value is the value of the region.

If the function fails, the return value is **NULL**.

Remarks

Region coordinates are represented as 27-bit signed integers.

An application can retrieve data for a region by calling the [GetRegionData](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePolyPolygonRgn](#)

[CreatePolygonRgn](#)

[CreateRectRgn](#)

[CreateRectRgnIndirect](#)

[CreateRoundRectRgn](#)

[GetRegionData](#)

[RGNDATA](#)

[Region Functions](#)

[Regions Overview](#)

[XFORM](#)

ExtFloodFill function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ExtFloodFill** function fills an area of the display surface with the current brush.

Syntax

```
BOOL ExtFloodFill(
    [in] HDC      hdc,
    [in] int      x,
    [in] int      y,
    [in] COLORREF color,
    [in] UINT     type
);
```

Parameters

[in] `hdc`

A handle to a device context.

[in] `x`

The x-coordinate, in logical units, of the point where filling is to start.

[in] `y`

The y-coordinate, in logical units, of the point where filling is to start.

[in] `color`

The color of the boundary or of the area to be filled. The interpretation of `color` depends on the value of the `fuFillType` parameter. To create a `COLORREF` color value, use the `RGB` macro.

[in] `type`

The type of fill operation to be performed. This parameter must be one of the following values.

VALUE	MEANING
<code>FLOODFILLBORDER</code>	The fill area is bounded by the color specified by the <code>color</code> parameter. This style is identical to the filling performed by the <code>FloodFill</code> function.
<code>FLOODFILLSURFACE</code>	The fill area is defined by the color that is specified by <code>color</code> . Filling continues outward in all directions as long as the color is encountered. This style is useful for filling areas with multicolored boundaries.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The following are some of the reasons this function might fail:

- The filling could not be completed.
- The specified point has the boundary color specified by the *color* parameter (if FLOODFILLBORDER was requested).
- The specified point does not have the color specified by *color* (if FLOODFILLSURFACE was requested).
- The point is outside the clipping region, that is, it is not visible on the device.

If the *fuFillType* parameter is FLOODFILLBORDER, the system assumes that the area to be filled is completely bounded by the color specified by the *color* parameter. The function begins filling at the point specified by the *nXStart* and *nYStart* parameters and continues in all directions until it reaches the boundary.

If *fuFillType* is FLOODFILLSURFACE, the system assumes that the area to be filled is a single color. The function begins to fill the area at the point specified by *nXStart* and *nYStart* and continues in all directions, filling all adjacent regions containing the color specified by *color*.

Only memory device contexts and devices that support raster-display operations support the **ExtFloodFill** function. To determine whether a device supports this technology, use the [GetDeviceCaps](#) function.

Examples

For an example, see "Adding Lines and Graphs to a Menu" in [Using Menus](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[COLORREF](#)

[FloodFill](#)

[GetDeviceCaps](#)

[RGB](#)

EXTLOGFONTA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The EXTLOGFONT structure defines the attributes of a font.

Syntax

```
typedef struct tagEXTLOGFONTA {
    LOGFONTA elfLogFont;
    BYTE      elfFullName[LF_FULLFACESIZE];
    BYTE      elfStyle[LF_FACESIZE];
    DWORD     elfVersion;
    DWORD     elfStyleSize;
    DWORD     elfMatch;
    DWORD     elfReserved;
    BYTE      elfVendorId[ELF_VENDOR_SIZE];
    DWORD     elfCulture;
    PANOSE    elfPanose;
} EXTLOGFONTA, *PEXTLOGFONTA, *NPEXTLOGFONTA, *LPEXTLOGFONTA;
```

Members

elfLogFont

Specifies some of the attributes of the specified font. This member is a [LOGFONT](#) structure.

elfFullName

A unique name for the font (for example, ABCD Font Company TrueType Bold Italic Sans Serif).

elfStyle

The style of the font (for example, Bold Italic).

elfVersion

Reserved. Must be zero.

elfStyleSize

This member only has meaning for hinted fonts. It specifies the point size at which the font is hinted. If set to zero, which is its default value, the font is hinted at the point size corresponding to the **lfHeight** member of the [LOGFONT](#) structure specified by **elfLogFont**.

elfMatch

A unique identifier for an enumerated font. This will be filled in by the graphics device interface (GDI) upon font enumeration.

elfReserved

Reserved; must be zero.

elfVendorId

A 4-byte identifier of the font vendor.

elfCulture

Reserved; must be zero.

elfPanose

A [PANOSE](#) structure that specifies the shape of the font. If all members of this structure are set to zero, the **elfPanose** member is ignored by the font mapper.

Remarks

NOTE

The wingdi.h header defines EXTLOGFONT as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

[PANOSE](#)

EXTLOGFONTW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The EXTLOGFONT structure defines the attributes of a font.

Syntax

```
typedef struct tagEXTLOGFONTW {
    LOGFONTW elfLogFont;
    WCHAR     elfFullName[LF_FULLFACESIZE];
    WCHAR     elfStyle[LF_FACESIZE];
    DWORD     elfVersion;
    DWORD     elfStyleSize;
    DWORD     elfMatch;
    DWORD     elfReserved;
    BYTE      elfVendorId[ELF_VENDOR_SIZE];
    DWORD     elfCulture;
    PANOSE    elfPanose;
} EXTLOGFONTW, *PEXTLOGFONTW, *NPEXTLOGFONTW, *LPEXTLOGFONTW;
```

Members

`elfLogFont`

Specifies some of the attributes of the specified font. This member is a [LOGFONT](#) structure.

`elfFullName`

A unique name for the font (for example, ABCD Font Company TrueType Bold Italic Sans Serif).

`elfStyle`

The style of the font (for example, Bold Italic).

`elfVersion`

Reserved. Must be zero.

`elfStyleSize`

This member only has meaning for hinted fonts. It specifies the point size at which the font is hinted. If set to zero, which is its default value, the font is hinted at the point size corresponding to the `lfHeight` member of the [LOGFONT](#) structure specified by `elfLogFont`.

`elfMatch`

A unique identifier for an enumerated font. This will be filled in by the graphics device interface (GDI) upon font enumeration.

`elfReserved`

Reserved; must be zero.

`elfVendorId`

A 4-byte identifier of the font vendor.

elfCulture

Reserved; must be zero.

elfPanose

A [PANOSE](#) structure that specifies the shape of the font. If all members of this structure are set to zero, the **elfPanose** member is ignored by the font mapper.

Remarks

NOTE

The wingdi.h header defines EXTLOGFONT as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

[PANOSE](#)

EXTLOGOPEN structure (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **EXTLOGOPEN** structure defines the pen style, width, and brush attributes for an extended pen. This structure is used by the [GetObject](#) function when it retrieves a description of a pen that was created when an application called the [ExtCreatePen](#) function.

Syntax

```
typedef struct tagEXTLOGOPEN {  
    DWORD     elpPenStyle;  
    DWORD     elpWidth;  
    UINT      elpBrushStyle;  
    COLORREF  elpColor;  
    ULONG_PTR elpHatch;  
    DWORD     elpNumEntries;  
    DWORD     elpStyleEntry[1];  
} EXTLOGOPEN, *PEXTLOGOPEN, *NPEXTLOGOPEN, *LPEXTLOGOPEN;
```

Members

elpPenStyle

A combination of pen type, style, end cap style, and join style. The values from each category can be retrieved by using a bitwise AND operator with the appropriate mask.

The **elpPenStyle** member masked with **PS_TYPE_MASK** has one of the following pen type values.

VALUE	MEANING
PS_GEOMETRIC	The pen is geometric.
PS_COSMETIC	The pen is cosmetic.

The **elpPenStyle** member masked with **PS_STYLE_MASK** has one of the following pen styles values:

VALUE	MEANING
PS_DASH	The pen is dashed.
PS_DASHDOT	The pen has alternating dashes and dots.
PS_DASHDOTDOT	The pen has alternating dashes and double dots.
PS_DOT	The pen is dotted.

PS_INSIDEFRAME	The pen is solid. When this pen is used in any GDI drawing function that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to PS_GEOMETRIC pens.
PS_NULL	The pen is invisible.
PS_SOLID	The pen is solid.
PS_USERSTYLE	The pen uses a styling array supplied by the user.

The following category applies only to PS_GEOMETRIC pens. The **elpPenStyle** member masked with PS_ENDCAP_MASK has one of the following end cap values.

VALUE	MEANING
PS_ENDCAP_FLAT	Line end caps are flat.
PS_ENDCAP_ROUND	Line end caps are round.
PS_ENDCAP_SQUARE	Line end caps are square.

The following category applies only to PS_GEOMETRIC pens. The **elpPenStyle** member masked with PS_JOIN_MASK has one of the following join values.

VALUE	MEANING
PS_JOIN_BEVEL	Line joins are beveled.
PS_JOIN_MITER	Line joins are mitered when they are within the current limit set by the SetMiterLimit function. A join is beveled when it would exceed the limit.
PS_JOIN_ROUND	Line joins are round.

elpWidth

The width of the pen. If the **elpPenStyle** member is PS_GEOMETRIC, this value is the width of the line in logical units. Otherwise, the lines are cosmetic and this value is 1, which indicates a line with a width of one pixel.

elpBrushStyle

The brush style of the pen. The **elpBrushStyle** member value can be one of the following.

VALUE	MEANING
BS_DIBPATTERN	Specifies a pattern brush defined by a DIB specification. If elpBrushStyle is BS_DIBPATTERN, the elpHatch member contains a handle to a packed DIB. For more information, see discussion in elpHatch

BS_DIBPATTERNPT	Specifies a pattern brush defined by a DIB specification. If elpBrushStyle is BS_DIBPATTERNPT, the elpHatch member contains a pointer to a packed DIB. For more information, see discussion in elpHatch .
BS_HATCHED	Specifies a hatched brush.
BS_HOLLOW	Specifies a hollow or NULL brush.
BS_PATTERN	Specifies a pattern brush defined by a memory bitmap.
BS_SOLID	Specifies a solid brush.

elpColor

If **elpBrushStyle** is BS_SOLID or BS_HATCHED, **elpColor** specifies the color in which the pen is to be drawn. For BS_HATCHED, the [SetBkMode](#) and [SetBkColor](#) functions determine the background color.

If **elpBrushStyle** is BS_HOLLOW or BS_PATTERN, **elpColor** is ignored.

If **elpBrushStyle** is BS_DIBPATTERN or BS_DIBPATTERNPT, the low-order word of **elpColor** specifies whether the **bmiColors** member of the [BITMAPINFO](#) structure contain explicit RGB values or indices into the currently realized logical palette. The **elpColor** value must be one of the following.

VALUE	MEANING
DIB_PAL_COLORS	The color table consists of an array of 16-bit indices into the currently realized logical palette.
DIB_RGB_COLORS	The color table contains literal RGB values.

The [RGB](#) macro is used to generate a [COLORREF](#) structure.

elpHatch

If **elpBrushStyle** is BS_PATTERN, **elpHatch** is a handle to the bitmap that defines the pattern.

If **elpBrushStyle** is BS_SOLID or BS_HOLLOW, **elpHatch** is ignored.

If **elpBrushStyle** is BS_DIBPATTERN, the **elpHatch** member is a handle to a packed DIB. To obtain this handle, an application calls the [GlobalAlloc](#) function with GMEM_MOVEABLE (or [LocalAlloc](#) with LMEM_MOVEABLE) to allocate a block of memory and then fills the memory with the packed DIB. A packed DIB consists of a [BITMAPINFO](#) structure immediately followed by the array of bytes that define the pixels of the bitmap.

If **elpBrushStyle** is BS_DIBPATTERNPT, the **elpHatch** member is a pointer to a packed DIB. The pointer derives from the memory block created by [LocalAlloc](#) with LMEM_FIXED set or by [GlobalAlloc](#) with GMEM_FIXED set, or it is the pointer returned by a call like [LocalLock](#) (handle_to_the_dib). A packed DIB consists of a [BITMAPINFO](#) structure immediately followed by the array of bytes that define the pixels of the bitmap.

If **elpBrushStyle** is BS_HATCHED, the **elpHatch** member specifies the orientation of the lines used to create the hatch. It can be one of the following values.

VALUE	MEANING
HS_BDIAGONAL	45-degree upward hatch (left to right)

HS_CROSS	Horizontal and vertical crosshatch
HS_DIAGCROSS	45-degree crosshatch
HS_FDIAGONAL	45-degree downward hatch (left to right)
HS_HORIZONTAL	Horizontal hatch
HS_VERTICAL	Vertical hatch

elpNumEntries

The number of entries in the style array in the `elpStyleEntry` member. This value is zero if `elpPenStyle` does not specify PS_USERSTYLE.

elpStyleEntry

A user-supplied style array. The array is specified with a finite length, but it is used as if it repeated indefinitely. The first entry in the array specifies the length of the first dash. The second entry specifies the length of the first gap. Thereafter, lengths of dashes and gaps alternate.

If `elpWidth` specifies geometric lines, the lengths are in logical units. Otherwise, the lines are cosmetic and lengths are in device units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[COLORREF](#)

[ExtCreatePen](#)

[GetObject](#)

[Pen Structures](#)

[Pens Overview](#)

[RGB](#)

[SetBkColor](#)

[SetBkMode](#)

ExtSelectClipRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ExtSelectClipRgn** function combines the specified region with the current clipping region using the specified mode.

Syntax

```
int ExtSelectClipRgn(
    [in] HDC hdc,
    [in] HRGN hrgn,
    [in] int mode
);
```

Parameters

[in] *hdc*

A handle to the device context.

[in] *hrgn*

A handle to the region to be selected. This handle must not be **NULL** unless the **RGN_COPY** mode is specified.

[in] *mode*

The operation to be performed. It must be one of the following values.

VALUE	MEANING
RGN_AND	The new clipping region combines the overlapping areas of the current clipping region and the region identified by <i>hrgn</i> .
RGN_COPY	The new clipping region is a copy of the region identified by <i>hrgn</i> . This is identical to SelectClipRgn . If the region identified by <i>hrgn</i> is NULL , the new clipping region is the default clipping region (the default clipping region is a null region).
RGN_DIFF	The new clipping region combines the areas of the current clipping region with those areas excluded from the region identified by <i>hrgn</i> .
RGN_OR	The new clipping region combines the current clipping region and the region identified by <i>hrgn</i> .
RGN_XOR	The new clipping region combines the current clipping region and the region identified by <i>hrgn</i> but excludes any overlapping areas.

Return value

The return value specifies the new clipping region's complexity; it can be one of the following values.

RETURN CODE	DESCRIPTION
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred.

Remarks

If an error occurs when this function is called, the previous clipping region for the specified device context is not affected.

The `ExtSelectClipRgn` function assumes that the coordinates for the specified region are specified in device units.

Only a copy of the region identified by the *hrgn* parameter is used. The region itself can be reused after this call or it can be deleted.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Clipping Functions](#)

[Clipping Overview](#)

[SelectClipRgn](#)

ExtTextOutA function (wingdi.h)

4/21/2022 • 6 minutes to read • [Edit Online](#)

The **ExtTextOut** function draws text using the currently selected font, background color, and text color. You can optionally provide dimensions to be used for clipping, opaquing, or both.

Syntax

```
BOOL ExtTextOutA(
    [in] HDC      hdc,
    [in] int      x,
    [in] int      y,
    [in] UINT     options,
    [in] const RECT *lprect,
    [in] LPCSTR   lpString,
    [in] UINT     c,
    [in] const INT *lpDx
);
```

Parameters

[in] hdc

A handle to the device context.

[in] x

The x-coordinate, in logical coordinates, of the reference point used to position the string.

[in] y

The y-coordinate, in logical coordinates, of the reference point used to position the string.

[in] options

Specifies how to use the application-defined rectangle. This parameter can be one or more of the following values.

VALUE	MEANING
ETO_CLIPPED	The text will be clipped to the rectangle.

ETO_GLYPH_INDEX	The <i>lpString</i> array refers to an array returned from GetCharacterPlacement and should be parsed directly by GDI as no further language-specific processing is required. Glyph indexing only applies to TrueType fonts, but the flag can be used for bitmap and vector fonts to indicate that no further language processing is necessary and GDI should process the string directly. Note that all glyph indexes are 16-bit values even though the string is assumed to be an array of 8-bit values for raster fonts. For ExtTextOutW, the glyph indexes are saved to a metafile. However, to display the correct characters the metafile must be played back using the same font. For ExtTextOutA, the glyph indexes are not saved.
ETO_IGNORELANGUAGE	Reserved for system use. If an application sets this flag, it loses international scripting support and in some cases it may display no text at all.
ETO_NUMERICSLATIN	To display numbers, use European digits.
ETO_NUMERICSLOCAL	To display numbers, use digits appropriate to the locale.
ETO_OPAQUE	The current background color should be used to fill the rectangle.
ETO_PDY	When this is set, the array pointed to by <i>lpDx</i> contains pairs of values. The first value of each pair is, as usual, the distance between origins of adjacent character cells, but the second value is the displacement along the vertical direction of the font.
ETO_RTLREADING	Middle East language edition of Windows: If this value is specified and a Hebrew or Arabic font is selected into the device context, the string is output using right-to-left reading order. If this value is not specified, the string is output in left-to-right order. The same effect can be achieved by setting the TA_RTLREADING value in SetTextAlign . This value is preserved for backward compatibility.

The ETO_GLYPH_INDEX and ETO_RTLREADING values cannot be used together. Because ETO_GLYPH_INDEX implies that all language processing has been completed, the function ignores the ETO_RTLREADING flag if also specified.

[in] *lprect*

A pointer to an optional [RECT](#) structure that specifies the dimensions, in logical coordinates, of a rectangle that is used for clipping, opaquing, or both.

[in] *lpString*

A pointer to a string that specifies the text to be drawn. The string does not need to be zero-terminated, since *cbCount* specifies the length of the string.

[in] *c*

The length of the string pointed to by *lpString*.

This value may not exceed 8192.

[in] *lpDx*

A pointer to an optional array of values that indicate the distance between origins of adjacent character cells. For example, *lpDx[i]* logical units separate the origins of character cell *i* and character cell *i* + 1.

Return value

If the string is drawn, the return value is nonzero. However, if the ANSI version of **ExtTextOut** is called with **ETO_GLYPH_INDEX**, the function returns **TRUE** even though the function does nothing.

If the function fails, the return value is zero.

Remarks

The current text-alignment settings for the specified device context determine how the reference point is used to position the text. The text-alignment settings are retrieved by calling the [GetTextAlign](#) function. The text-alignment settings are altered by calling the [SetTextAlign](#) function. You can use the following values for text alignment. Only one flag can be chosen from those that affect horizontal and vertical alignment. In addition, only one of the two flags that alter the current position can be chosen.

TERM	DESCRIPTION
TA_BASELINE	The reference point will be on the base line of the text.
TA_BOTTOM	The reference point will be on the bottom edge of the bounding rectangle.
TA_TOP	The reference point will be on the top edge of the bounding rectangle.
TA_CENTER	The reference point will be aligned horizontally with the center of the bounding rectangle.
TA_LEFT	The reference point will be on the left edge of the bounding rectangle.
TA_RIGHT	The reference point will be on the right edge of the bounding rectangle.
TA_NOUPDATECP	The current position is not updated after each text output call. The reference point is passed to the text output function.
TA_RTLREADING	Middle East language edition of Windows: The text is laid out in right to left reading order, as opposed to the default left to right order. This applies only when the font selected into the device context is either Hebrew or Arabic.
TA_UPDATECP	The current position is updated after each text output call. The current position is used as the reference point.

If the *lpDx* parameter is **NULL**, the **ExtTextOut** function uses the default spacing between characters. The

character-cell origins and the contents of the array pointed to by the *lpDx* parameter are specified in logical units. A character-cell origin is defined as the upper-left corner of the character cell.

By default, the current position is not used or updated by this function. However, an application can call the [SetTextAlign](#) function with the *fMode* parameter set to TA_UPDATECP to permit the system to use and update the current position each time the application calls [ExtTextOut](#) for a specified device context. When this flag is set, the system ignores the *X* and *Y* parameters on subsequent [ExtTextOut](#) calls.

For the ANSI version of [ExtTextOut](#), the *lpDx* array has the same number of INT values as there are bytes in *lpString*. For DBCS characters, you can apportion the dx in the *lpDx* entries between the lead byte and the trail byte, as long as the sum of the two bytes adds up to the desired dx. For DBCS characters with the Unicode version of [ExtTextOut](#), each Unicode glyph gets a single *pdx* entry.

Note, the *alpDx* values from [GetTextExtentExPoint](#) are not the same as the *lpDx* values for [ExtTextOut](#). To use the *alpDx* values in *lpDx*, you must first process them.

[ExtTextOut](#) will use [Uniscribe](#) when necessary resulting in font fallback. The ETO_IGNORELANGUAGE flag will inhibit this behavior and should not be passed.

Additionally, [ExtTextOut](#) will perform internal batching of calls before transitioning to kernel mode, mitigating some of the performance concerns when weighing usage of [PolyTextOut](#) versus [ExtTextOut](#).

TIP

[ExtTextOut](#) is strongly recommended over [PolyTextOut](#) for modern development due to its ability to handle display of different languages.

Examples

For an example, see "Setting Fonts for Menu-Item Text Strings" in [Using Menus](#).

NOTE

The wingdi.h header defines ExtTextOut as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[RECT](#)

[SelectObject](#)

[SetBkColor](#)

[SetTextAlign](#)

[SetTextColor](#)

ExtTextOutW function (wingdi.h)

4/21/2022 • 6 minutes to read • [Edit Online](#)

The **ExtTextOut** function draws text using the currently selected font, background color, and text color. You can optionally provide dimensions to be used for clipping, opaquing, or both.

Syntax

```
BOOL ExtTextOutW(
    [in] HDC      hdc,
    [in] int      x,
    [in] int      y,
    [in] UINT     options,
    [in] const RECT *lprect,
    [in] LPCWSTR   lpString,
    [in] UINT      c,
    [in] const INT *lpDx
);
```

Parameters

[in] hdc

A handle to the device context.

[in] x

The x-coordinate, in logical coordinates, of the reference point used to position the string.

[in] y

The y-coordinate, in logical coordinates, of the reference point used to position the string.

[in] options

Specifies how to use the application-defined rectangle. This parameter can be one or more of the following values.

VALUE	MEANING
ETO_CLIPPED	The text will be clipped to the rectangle.

ETO_GLYPH_INDEX	The <i>lpString</i> array refers to an array returned from GetCharacterPlacement and should be parsed directly by GDI as no further language-specific processing is required. Glyph indexing only applies to TrueType fonts, but the flag can be used for bitmap and vector fonts to indicate that no further language processing is necessary and GDI should process the string directly. Note that all glyph indexes are 16-bit values even though the string is assumed to be an array of 8-bit values for raster fonts. For ExtTextOutW, the glyph indexes are saved to a metafile. However, to display the correct characters the metafile must be played back using the same font. For ExtTextOutA, the glyph indexes are not saved.
ETO_IGNORELANGUAGE	Reserved for system use. If an application sets this flag, it loses international scripting support and in some cases it may display no text at all.
ETO_NUMERICSLATIN	To display numbers, use European digits.
ETO_NUMERICSLOCAL	To display numbers, use digits appropriate to the locale.
ETO_OPAQUE	The current background color should be used to fill the rectangle.
ETO_PDY	When this is set, the array pointed to by <i>lpDx</i> contains pairs of values. The first value of each pair is, as usual, the distance between origins of adjacent character cells, but the second value is the displacement along the vertical direction of the font.
ETO_RTLREADING	Middle East language edition of Windows: If this value is specified and a Hebrew or Arabic font is selected into the device context, the string is output using right-to-left reading order. If this value is not specified, the string is output in left-to-right order. The same effect can be achieved by setting the TA_RTLREADING value in SetTextAlign . This value is preserved for backward compatibility.

The ETO_GLYPH_INDEX and ETO_RTLREADING values cannot be used together. Because ETO_GLYPH_INDEX implies that all language processing has been completed, the function ignores the ETO_RTLREADING flag if also specified.

[in] *lprect*

A pointer to an optional [RECT](#) structure that specifies the dimensions, in logical coordinates, of a rectangle that is used for clipping, opaquing, or both.

[in] *lpString*

A pointer to a string that specifies the text to be drawn. The string does not need to be zero-terminated, since *cbCount* specifies the length of the string.

[in] *c*

The length of the string pointed to by *lpString*.

This value may not exceed 8192.

[in] *lpDx*

A pointer to an optional array of values that indicate the distance between origins of adjacent character cells. For example, *lpDx[i]* logical units separate the origins of character cell *i* and character cell *i* + 1.

Return value

If the string is drawn, the return value is nonzero. However, if the ANSI version of **ExtTextOut** is called with **ETO_GLYPH_INDEX**, the function returns **TRUE** even though the function does nothing.

If the function fails, the return value is zero.

Remarks

The current text-alignment settings for the specified device context determine how the reference point is used to position the text. The text-alignment settings are retrieved by calling the [GetTextAlign](#) function. The text-alignment settings are altered by calling the [SetTextAlign](#) function. You can use the following values for text alignment. Only one flag can be chosen from those that affect horizontal and vertical alignment. In addition, only one of the two flags that alter the current position can be chosen.

TERM	DESCRIPTION
TA_BASELINE	The reference point will be on the base line of the text.
TA_BOTTOM	The reference point will be on the bottom edge of the bounding rectangle.
TA_TOP	The reference point will be on the top edge of the bounding rectangle.
TA_CENTER	The reference point will be aligned horizontally with the center of the bounding rectangle.
TA_LEFT	The reference point will be on the left edge of the bounding rectangle.
TA_RIGHT	The reference point will be on the right edge of the bounding rectangle.
TA_NOUPDATECP	The current position is not updated after each text output call. The reference point is passed to the text output function.
TA_RTLREADING	Middle East language edition of Windows: The text is laid out in right to left reading order, as opposed to the default left to right order. This applies only when the font selected into the device context is either Hebrew or Arabic.
TA_UPDATECP	The current position is updated after each text output call. The current position is used as the reference point.

If the *lpDx* parameter is **NULL**, the **ExtTextOut** function uses the default spacing between characters. The

character-cell origins and the contents of the array pointed to by the *lpDx* parameter are specified in logical units. A character-cell origin is defined as the upper-left corner of the character cell.

By default, the current position is not used or updated by this function. However, an application can call the [SetTextAlign](#) function with the *fMode* parameter set to TA_UPDATECP to permit the system to use and update the current position each time the application calls [ExtTextOut](#) for a specified device context. When this flag is set, the system ignores the *X* and *Y* parameters on subsequent [ExtTextOut](#) calls.

For the ANSI version of [ExtTextOut](#), the *lpDx* array has the same number of INT values as there are bytes in *lpString*. For DBCS characters, you can apportion the dx in the *lpDx* entries between the lead byte and the trail byte, as long as the sum of the two bytes adds up to the desired dx. For DBCS characters with the Unicode version of [ExtTextOut](#), each Unicode glyph gets a single *pdx* entry.

Note, the *alpDx* values from [GetTextExtentExPoint](#) are not the same as the *lpDx* values for [ExtTextOut](#). To use the *alpDx* values in *lpDx*, you must first process them.

[ExtTextOut](#) will use [Uniscribe](#) when necessary resulting in font fallback. The ETO_IGNORELANGUAGE flag will inhibit this behavior and should not be passed.

Additionally, [ExtTextOut](#) will perform internal batching of calls before transitioning to kernel mode, mitigating some of the performance concerns when weighing usage of [PolyTextOut](#) versus [ExtTextOut](#).

TIP

[ExtTextOut](#) is strongly recommended over [PolyTextOut](#) for modern development due to its ability to handle display of different languages.

Examples

For an example, see "Setting Fonts for Menu-Item Text Strings" in [Using Menus](#).

NOTE

The wingdi.h header defines ExtTextOut as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[RECT](#)

[SelectObject](#)

[SetBkColor](#)

[SetTextAlign](#)

[SetTextColor](#)

FillPath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **FillPath** function closes any open figures in the current path and fills the path's interior by using the current brush and polygon-filling mode.

Syntax

```
BOOL FillPath(  
    [in] HDC hdc  
>;
```

Parameters

[in] *hdc*

A handle to a device context that contains a valid path.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

After its interior is filled, the path is discarded from the DC identified by the *hdc* parameter.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[Path Functions](#)

[Paths Overview](#)

[SetPolyFillMode](#)

[StrokeAndFillPath](#)

[StrokePath](#)

FillRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **FillRgn** function fills a region by using the specified brush.

Syntax

```
BOOL FillRgn(
    [in] HDC     hdc,
    [in] HRGN    hrgn,
    [in] HBRUSH  hbr
);
```

Parameters

[in] **hdc**

Handle to the device context.

[in] **hrgn**

Handle to the region to be filled. The region's coordinates are presumed to be in logical units.

[in] **hbr**

Handle to the brush to be used to fill the region.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateBrushIndirect](#)

[CreateDIBPatternBrush](#)

[CreateHatchBrush](#)

[CreatePatternBrush](#)

[CreateSolidBrush](#)

[PaintRgn](#)

[Region Functions](#)

[Regions Overview](#)

FIXED structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **FIXED** structure contains the integral and fractional parts of a fixed-point real number.

Syntax

```
typedef struct _FIXED {
    #if ...
        WORD fract;
    #if ...
        short value;
    #else
        short value;
    #endif
    #else
        WORD fract;
    #endif
} FIXED;
```

Members

`fract`

The fractional part of the number.

`value`

The integer part of the number.

Remarks

The **FIXED** structure is used to describe the elements of the [MAT2](#) structure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[MAT2](#)

FlattenPath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **FlattenPath** function transforms any curves in the path that is selected into the current device context (DC), turning each curve into a sequence of lines.

Syntax

```
BOOL FlattenPath(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

A handle to a DC that contains a valid path.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Path Functions](#)

[Paths Overview](#)

[WidenPath](#)

FloodFill function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **FloodFill** function fills an area of the display surface with the current brush. The area is assumed to be bounded as specified by the *color* parameter.

Note The **FloodFill** function is included only for compatibility with 16-bit versions of Windows. Applications should use the [ExtFloodFill](#) function with FLOODFILLBORDER specified.

Syntax

```
BOOL FloodFill(
    [in] HDC      hdc,
    [in] int      x,
    [in] int      y,
    [in] COLORREF color
);
```

Parameters

[in] *hdc*

A handle to a device context.

[in] *x*

The x-coordinate, in logical units, of the point where filling is to start.

[in] *y*

The y-coordinate, in logical units, of the point where filling is to start.

[in] *color*

The color of the boundary or the area to be filled. To create a **COLORREF** color value, use the **RGB** macro.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The following are reasons this function might fail:

- The fill could not be completed.
- The given point has the boundary color specified by the *color* parameter.
- The given point lies outside the current clipping region, that is, it is not visible on the device.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[COLORREF](#)

[ExtFloodFill](#)

[RGB](#)

FrameRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **FrameRgn** function draws a border around the specified region by using the specified brush.

Syntax

```
BOOL FrameRgn(
    [in] HDC     hdc,
    [in] HRGN   hrgn,
    [in] HBRUSH hbr,
    [in] int     w,
    [in] int     h
);
```

Parameters

[in] **hdc**

Handle to the device context.

[in] **hrgn**

Handle to the region to be enclosed in a border. The region's coordinates are presumed to be in logical units.

[in] **hbr**

Handle to the brush to be used to draw the border.

[in] **w**

Specifies the width, in logical units, of vertical brush strokes.

[in] **h**

Specifies the height, in logical units, of horizontal brush strokes.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[FillRgn](#)

[PaintRgn](#)

[Region Functions](#)

[Regions Overview](#)

GCP_RESULTS structure (wingdi.h)

4/21/2022 • 8 minutes to read • [Edit Online](#)

The **GCP_RESULTS** structure contains information about characters in a string. This structure receives the results of the [GetCharacterPlacement](#) function. For some languages, the first element in the arrays may contain more, language-dependent information.

Syntax

```
typedef struct tagGCP_RESULTS {
    DWORD  lStructSize;
    LPSTR  lpOutString;
    UINT   *lpOrder;
    int    *lpDx;
    int    *lpCaretPos;
    LPSTR  lpClass;
    LPWSTR lpGlyphs;
    UINT   nGlyphs;
    int    nMaxFit;
} GCP_RESULTS, *LPGCP_RESULTS;
```

Members

lStructSize

The size, in bytes, of the structure.

lpOutString

A pointer to the buffer that receives the output string or is **NULL** if the output string is not needed. The output string is a version of the original string that is in the order that will be displayed on a specified device. Typically the output string is identical to the original string, but may be different if the string needs reordering and the **GCP_REORDER** flag is set or if the original string exceeds the maximum extent and the **GCP_MAXEXTENT** flag is set.

lpOrder

A pointer to the array that receives ordering indexes or is **NULL** if the ordering indexes are not needed. However, its meaning depends on the other elements of **GCP_RESULTS**. If glyph indexes are to be returned, the indexes are for the **lpGlyphs** array; if glyphs indexes are not returned and **lpOrder** is requested, the indexes are for **lpOutString**. For example, in the latter case the value of **lpOrder[i]** is the position of **lpString[i]** in the output string **lpOutString**.

This is typically used when [GetFontLanguageInfo](#) returns the **GCP_REORDER** flag, which indicates that the original string needs reordering. For example, in Hebrew, in which the text runs from right to left, the **lpOrder** array gives the exact locations of each element in the original string.

lpDx

A pointer to the array that receives the distances between adjacent character cells or is **NULL** if these distances are not needed. If glyph rendering is done, the distances are for the glyphs not the characters, so the resulting array can be used with the [ExtTextOut](#) function.

The distances in this array are in display order. To find the distance for the *i*th character in the original string, use

the **lpOrder** array as follows:

```
width = lpDx[lpOrder[i]];
```

lpCaretPos

A pointer to the array that receives the caret position values or is **NULL** if caret positions are not needed. Each value specifies the caret position immediately before the corresponding character. In some languages the position of the caret for each character may not be immediately to the left of the character. For example, in Hebrew, in which the text runs from right to left, the caret position is to the right of the character. If glyph ordering is done, **lpCaretPos** matches the original string, not the output string. This means that some adjacent values may be the same.

The values in this array are in input order. To find the caret position value for the *i*th character in the original string, use the array as follows:

```
position = lpCaretPos[i];
```

lpClass

A pointer to the array that contains and/or receives character classifications. The values indicate how to lay out characters in the string and are similar (but not identical) to the **CT_CTYPE2** values returned by the [GetStringTypeEx](#) function. Each element of the array can be set to zero or one of the following values.

VALUE	MEANING
GCPCLASS_ARABIC	Arabic character.
GCPCLASS_HEBREW	Hebrew character.
GCPCLASS_LATIN	Character from a Latin or other single-byte character set for a left-to-right language.
GCPCLASS_LATINNUMBER	Digit from a Latin or other single-byte character set for a left-to-right language.
GCPCLASS_LOCALNUMBER	Digit from the character set associated with the current font.

In addition, the following can be used when supplying values in the **lpClass** array with the **GCP_CLASSIN** flag.

VALUE	MEANING
GCPCLASS_LATINNUMERICSEPARATOR	Input only. Character used to separate Latin digits, such as a comma or decimal point.

GCPCCLASS_LATINNUMERICTERMINATOR	Input only. Character used to terminate Latin digits, such as a plus or minus sign.
GCPCCLASS_NEUTRAL	Input only. Character has no specific classification.
GCPCCLASS_NUMERICSEPARATOR	Input only. Character used to separate digits, such as a comma or decimal point.

For languages that use the GCP_REORDER flag, the following values can also be used with the GCP_CLASSIN flag. Unlike the preceding values, which can be used anywhere in the **IpClass** array, all of the following values are used only in the first location in the array. All combine with other classifications.

Note that GCPCLASS_PREBOUNDLTR and GCPCLASS_PREBOUNDRTL are mutually exclusive, as are GCPCLASSPOSTBOUNDLTR and GCPCLASSPOSTBOUNDRTL.

VALUE	MEANING
GCPCLASS_PREBOUNDLTR	Set IpClass[0] to GCPCLASS_PREBOUNDLTR to bind the string to left-to-right reading order before the string.
GCPCLASS_PREBOUNDRTL	Set IpClass[0] to GCPCLASS_PREBOUNDRTL to bind the string to right-to-left reading order before the string.
GCPCLASS_POSTBOUNDLTR	Set IpClass[0] to GCPCLASS_POSTBOUNDLTR to bind the string to left-to-right reading order after the string.
GCPCLASS_POSTBOUNDRTL	Set IpClass[0] to GCPCLASS_POSTBOUNDRTL to bind the string to right-to-left reading order after the string.

To force the layout of a character to be carried out in a specific way, preset the classification for the corresponding array element; the function leaves such preset classifications unchanged and computes classifications only for array elements that have been set to zero. Preset classifications are used only if the GCP_CLASSIN flag is set and the **IpClass** array is supplied.

If [GetFontLanguageInfo](#) does not return GCP_REORDER for the current font, only the GCPCLASS_LATIN value is meaningful.

lpGlyphs

A pointer to the array that receives the values identifying the glyphs used for rendering the string or is **NULL** if glyph rendering is not needed. The number of glyphs in the array may be less than the number of characters in the original string if the string contains ligated glyphs. Also if reordering is required, the order of the glyphs may not be sequential.

This array is useful if more than one operation is being done on a string which has any form of ligation, kerning or order-switching. Using the values in this array for subsequent operations saves the time otherwise required to generate the glyph indices each time.

This array always contains glyph indices and the ETO_GLYPH_INDEX value must always be used when this array is used with the [ExtTextOut](#) function.

When GCP_LIGATE is used, you can limit the number of characters that will be ligated together. (In Arabic for example, three-character ligations are common). This is done by setting the maximum required in `IpGcpResults->IpGlyphs[0]`. If no maximum is required, you should set this field to zero.

For languages such as Arabic, where [GetFontLanguageInfo](#) returns the GCP_GLYPHSHAPE flag, the glyphs for a character will be different depending on whether the character is at the beginning, middle, or end of a word. Typically, the first character in the input string will also be the first character in a word, and the last character in the input string will be treated as the last character in a word. However, if the displayed string is a subset of the complete string, such as when displaying a section of scrolled text, this may not be true. In these cases, it is desirable to force the first or last characters to be shaped as not being initial or final forms. To do this, again, the first location in the `IpGlyphs` array is used by performing an OR operation of the ligation value above with the values GCPGLYPH_LINKBEFORE and/or GCPGLYPH_LINKAFTER. For example, a value of GCPGLYPH_LINKBEFORE | 2 means that two-character ligatures are the maximum required, and the first character in the string should be treated as if it is in the middle of a word.

nGlyphs

On input, this member must be set to the size of the arrays pointed to by the array pointer members. On output, this is set to the number of glyphs filled in, in the output arrays. If glyph substitution is not required (that is, each input character maps to exactly one glyph), this member is the same as it is on input.

nMaxFit

The number of characters that fit within the extents specified by the `nMaxExtent` parameter of the [GetCharacterPlacement](#) function. If the GCP_MAXEXTENT or GCP_JUSTIFY value is set, this value may be less than the number of characters in the original string. This member is set regardless of whether the GCP_MAXEXTENT or GCP_JUSTIFY value is specified. Unlike `nGlyphs`, which specifies the number of output glyphs, `nMaxFit` refers to the number of characters from the input string. For Latin SBCS languages, this will be the same.

Remarks

Whether the `IpGlyphs`, `IpOutString`, or neither is required depends on the results of the [GetFontLanguageInfo](#) call.

In the case of a font for a language such as English, in which none of the GCP_DBCS, GCP_REORDER, GCP_GLYPHSHAPE, GCP_LIGATE, GCP_DIACRITIC, or GCP_KASHIDA flags are returned, neither of the arrays is required for proper operation. (Though not required, they can still be used. If the `IpOutString` array is used, it will be exactly the same as the `IpInputString` passed to [GetCharacterPlacement](#).) Note, however, that if GCP_MAXEXTENT is used, then `IpOutString` will contain the truncated string if it is used, NOT an exact copy of the original.

In the case of fonts for languages such as Hebrew, which DO have reordering but do not typically have extra glyph shapes, `IpOutString` should be used. This will give the string on the screen-readable order. However, the `IpGlyphs` array is not typically needed. (Hebrew can have extra glyphs, if the font is a TrueType/Open font.)

In the case of languages such as Thai or Arabic, in which [GetFontLanguageInfo](#) returns the GCP_GLYPHSHAPE flag, the `IpOutString` will give the display-readable order of the string passed to [GetCharacterPlacement](#), but the values will still be the unshaped characters. For proper display, the `IpGlyphs` array must be used.

NOTE

The wingdi.h header defines GCP_RESULTS as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[ExtTextOut](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetCharacterPlacement](#)

[GetFontLanguageInfo](#)

GCP_RESULTS structure (wingdi.h)

4/21/2022 • 8 minutes to read • [Edit Online](#)

The **GCP_RESULTS** structure contains information about characters in a string. This structure receives the results of the [GetCharacterPlacement](#) function. For some languages, the first element in the arrays may contain more, language-dependent information.

Syntax

```
typedef struct tagGCP_RESULTS {
    DWORD lStructSize;
    LPWSTR lpOutString;
    UINT *lpOrder;
    int *lpDx;
    int *lpCaretPos;
    LPSTR lpClass;
    LPWSTR lpGlyphs;
    UINT nGlyphs;
    int nMaxFit;
} GCP_RESULTS, *LPGCP_RESULTS;
```

Members

lStructSize

The size, in bytes, of the structure.

lpOutString

A pointer to the buffer that receives the output string or is **NULL** if the output string is not needed. The output string is a version of the original string that is in the order that will be displayed on a specified device. Typically the output string is identical to the original string, but may be different if the string needs reordering and the **GCP_REORDER** flag is set or if the original string exceeds the maximum extent and the **GCP_MAXEXTENT** flag is set.

lpOrder

A pointer to the array that receives ordering indexes or is **NULL** if the ordering indexes are not needed. However, its meaning depends on the other elements of **GCP_RESULTS**. If glyph indexes are to be returned, the indexes are for the **lpGlyphs** array; if glyphs indexes are not returned and **lpOrder** is requested, the indexes are for **lpOutString**. For example, in the latter case the value of **lpOrder[i]** is the position of **lpString[i]** in the output string **lpOutString**.

This is typically used when [GetFontLanguageInfo](#) returns the **GCP_REORDER** flag, which indicates that the original string needs reordering. For example, in Hebrew, in which the text runs from right to left, the **lpOrder** array gives the exact locations of each element in the original string.

lpDx

A pointer to the array that receives the distances between adjacent character cells or is **NULL** if these distances are not needed. If glyph rendering is done, the distances are for the glyphs not the characters, so the resulting array can be used with the [ExtTextOut](#) function.

The distances in this array are in display order. To find the distance for the *i*th character in the original string, use

the **lpOrder** array as follows:

```
width = lpDx[lpOrder[i]];
```

lpCaretPos

A pointer to the array that receives the caret position values or is **NULL** if caret positions are not needed. Each value specifies the caret position immediately before the corresponding character. In some languages the position of the caret for each character may not be immediately to the left of the character. For example, in Hebrew, in which the text runs from right to left, the caret position is to the right of the character. If glyph ordering is done, **lpCaretPos** matches the original string, not the output string. This means that some adjacent values may be the same.

The values in this array are in input order. To find the caret position value for the *i*th character in the original string, use the array as follows:

```
position = lpCaretPos[i];
```

lpClass

A pointer to the array that contains and/or receives character classifications. The values indicate how to lay out characters in the string and are similar (but not identical) to the **CT_CTYPE2** values returned by the [GetStringTypeEx](#) function. Each element of the array can be set to zero or one of the following values.

VALUE	MEANING
GCPCLASS_ARABIC	Arabic character.
GCPCLASS_HEBREW	Hebrew character.
GCPCLASS_LATIN	Character from a Latin or other single-byte character set for a left-to-right language.
GCPCLASS_LATINNUMBER	Digit from a Latin or other single-byte character set for a left-to-right language.
GCPCLASS_LOCALNUMBER	Digit from the character set associated with the current font.

In addition, the following can be used when supplying values in the **lpClass** array with the **GCP_CLASSIN** flag.

VALUE	MEANING
GCPCLASS_LATINNUMERICSEPARATOR	Input only. Character used to separate Latin digits, such as a comma or decimal point.

GCPCCLASS_LATINNUMERICTERMINATOR	Input only. Character used to terminate Latin digits, such as a plus or minus sign.
GCPCCLASS_NEUTRAL	Input only. Character has no specific classification.
GCPCCLASS_NUMERICSEPARATOR	Input only. Character used to separate digits, such as a comma or decimal point.

For languages that use the GCP_REORDER flag, the following values can also be used with the GCP_CLASSIN flag. Unlike the preceding values, which can be used anywhere in the **IpClass** array, all of the following values are used only in the first location in the array. All combine with other classifications.

Note that GCPCLASS_PREBOUNDLTR and GCPCLASS_PREBOUNDRTL are mutually exclusive, as are GCPCLASSPOSTBOUNDLTR and GCPCLASSPOSTBOUNDRTL.

VALUE	MEANING
GCPCLASS_PREBOUNDLTR	Set IpClass[0] to GCPCLASS_PREBOUNDLTR to bind the string to left-to-right reading order before the string.
GCPCLASS_PREBOUNDRTL	Set IpClass[0] to GCPCLASS_PREBOUNDRTL to bind the string to right-to-left reading order before the string.
GCPCLASS_POSTBOUNDLTR	Set IpClass[0] to GCPCLASS_POSTBOUNDLTR to bind the string to left-to-right reading order after the string.
GCPCLASS_POSTBOUNDRTL	Set IpClass[0] to GCPCLASS_POSTBOUNDRTL to bind the string to right-to-left reading order after the string.

To force the layout of a character to be carried out in a specific way, preset the classification for the corresponding array element; the function leaves such preset classifications unchanged and computes classifications only for array elements that have been set to zero. Preset classifications are used only if the GCP_CLASSIN flag is set and the **IpClass** array is supplied.

If [GetFontLanguageInfo](#) does not return GCP_REORDER for the current font, only the GCPCLASS_LATIN value is meaningful.

lpGlyphs

A pointer to the array that receives the values identifying the glyphs used for rendering the string or is **NULL** if glyph rendering is not needed. The number of glyphs in the array may be less than the number of characters in the original string if the string contains ligated glyphs. Also if reordering is required, the order of the glyphs may not be sequential.

This array is useful if more than one operation is being done on a string which has any form of ligation, kerning or order-switching. Using the values in this array for subsequent operations saves the time otherwise required to generate the glyph indices each time.

This array always contains glyph indices and the ETO_GLYPH_INDEX value must always be used when this array is used with the [ExtTextOut](#) function.

When GCP_LIGATE is used, you can limit the number of characters that will be ligated together. (In Arabic for example, three-character ligations are common). This is done by setting the maximum required in `IpGcpResults->IpGlyphs[0]`. If no maximum is required, you should set this field to zero.

For languages such as Arabic, where [GetFontLanguageInfo](#) returns the GCP_GLYPHSHAPE flag, the glyphs for a character will be different depending on whether the character is at the beginning, middle, or end of a word. Typically, the first character in the input string will also be the first character in a word, and the last character in the input string will be treated as the last character in a word. However, if the displayed string is a subset of the complete string, such as when displaying a section of scrolled text, this may not be true. In these cases, it is desirable to force the first or last characters to be shaped as not being initial or final forms. To do this, again, the first location in the `IpGlyphs` array is used by performing an OR operation of the ligation value above with the values GCPGLYPH_LINKBEFORE and/or GCPGLYPH_LINKAFTER. For example, a value of GCPGLYPH_LINKBEFORE | 2 means that two-character ligatures are the maximum required, and the first character in the string should be treated as if it is in the middle of a word.

nGlyphs

On input, this member must be set to the size of the arrays pointed to by the array pointer members. On output, this is set to the number of glyphs filled in, in the output arrays. If glyph substitution is not required (that is, each input character maps to exactly one glyph), this member is the same as it is on input.

nMaxFit

The number of characters that fit within the extents specified by the `nMaxExtent` parameter of the [GetCharacterPlacement](#) function. If the GCP_MAXEXTENT or GCP_JUSTIFY value is set, this value may be less than the number of characters in the original string. This member is set regardless of whether the GCP_MAXEXTENT or GCP_JUSTIFY value is specified. Unlike `nGlyphs`, which specifies the number of output glyphs, `nMaxFit` refers to the number of characters from the input string. For Latin SBCS languages, this will be the same.

Remarks

Whether the `IpGlyphs`, `IpOutString`, or neither is required depends on the results of the [GetFontLanguageInfo](#) call.

In the case of a font for a language such as English, in which none of the GCP_DBCS, GCP_REORDER, GCP_GLYPHSHAPE, GCP_LIGATE, GCP_DIACRITIC, or GCP_KASHIDA flags are returned, neither of the arrays is required for proper operation. (Though not required, they can still be used. If the `IpOutString` array is used, it will be exactly the same as the `IpInputString` passed to [GetCharacterPlacement](#).) Note, however, that if GCP_MAXEXTENT is used, then `IpOutString` will contain the truncated string if it is used, NOT an exact copy of the original.

In the case of fonts for languages such as Hebrew, which DO have reordering but do not typically have extra glyph shapes, `IpOutString` should be used. This will give the string on the screen-readable order. However, the `IpGlyphs` array is not typically needed. (Hebrew can have extra glyphs, if the font is a TrueType/Open font.)

In the case of languages such as Thai or Arabic, in which [GetFontLanguageInfo](#) returns the GCP_GLYPHSHAPE flag, the `IpOutString` will give the display-readable order of the string passed to [GetCharacterPlacement](#), but the values will still be the unshaped characters. For proper display, the `IpGlyphs` array must be used.

NOTE

The wingdi.h header defines GCP_RESULTS as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[ExtTextOut](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetCharacterPlacement](#)

[GetFontLanguageInfo](#)

GdiAlphaBlend function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **GdiAlphaBlend** function displays bitmaps that have transparent or semitransparent pixels.

Syntax

```
BOOL GdiAlphaBlend(
    [in] HDC         hdcDest,
    [in] int          xoriginDest,
    [in] int          yoriginDest,
    [in] int          wDest,
    [in] int          hDest,
    [in] HDC         hdcSrc,
    [in] int          xoriginSrc,
    [in] int          yoriginSrc,
    [in] int          wSrc,
    [in] int          hSrc,
    [in] BLENDFUNCTION ftn
);
```

Parameters

[in] hdcDest

A handle to the destination device context.

[in] xoriginDest

The x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] yoriginDest

The y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] wDest

The width, in logical units, of the destination rectangle.

[in] hDest

The height, in logical units, of the destination rectangle.

[in] hdcSrc

A handle to the source device context.

[in] xoriginSrc

The x-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] yoriginSrc

The y-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] wSrc

The width, in logical units, of the source rectangle.

[in] hSrc

The height, in logical units, of the source rectangle.

[in] ftn

The alpha-blending function for source and destination bitmaps, a global alpha value to be applied to the entire source bitmap, and format information for the source bitmap. The source and destination blend functions are currently limited to AC_SRC_OVER. See the [BLENDFUNCTION](#) and [EMRALPHABLEND](#) structures.

Return value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE.

This function can return the following value.

RETURN CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	One or more of the input parameters is invalid.

Remarks

Note This function is the same as [AlphaBlend](#).

If the source rectangle and destination rectangle are not the same size, the source bitmap is stretched to match the destination rectangle. If the [SetStretchBltMode](#) function is used, the *iStretchMode* value is automatically converted to COLORONCOLOR for this function (that is, BLACKONWHITE, WHITEONBLACK, and HALFTONE are changed to COLORONCOLOR).

The destination coordinates are transformed by using the transformation currently specified for the destination device context. The source coordinates are transformed by using the transformation currently specified for the source device context.

An error occurs (and the function returns FALSE) if the source device context identifies an enhanced metafile device context.

If destination and source bitmaps do not have the same color format, **GdiAlphaBlend** converts the source bitmap to match the destination bitmap.

GdiAlphaBlend does not support mirroring. If either the width or height of the source or destination is negative, this call will fail.

When rendering to a printer, first call [GetDeviceCaps](#) with SHADEBLENDCAPS to determine if the printer supports blending with **GdiAlphaBlend**. Note that, for a display DC, all blending operations are supported and these flags represent whether the operations are accelerated.

If the source and destination are the same surface, that is, they are both the screen or the same memory bitmap and the source and destination rectangles overlap, an error occurs and the function returns FALSE.

The source rectangle must lie completely within the source surface, otherwise an error occurs and the function returns FALSE.

GdiAlphaBlend fails if the width or height of the source or destination is negative.

The **SourceConstantAlpha** member of [BLENDFUNCTION](#) specifies an alpha transparency value to be used on the entire source bitmap. The **SourceConstantAlpha** value is combined with any per-pixel alpha values. If **SourceConstantAlpha** is 0, it is assumed that the image is transparent. Set the **SourceConstantAlpha** value to 255 (which indicates that the image is opaque) when you only want to use per-pixel alpha values.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BLENDFUNCTION](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[EMRALPHABLEND](#)

[GetDeviceCaps](#)

[SetStretchBltMode](#)

GdiComment function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GdiComment** function copies a comment from a buffer into a specified enhanced-format metafile.

Syntax

```
BOOL GdiComment(
    [in] HDC      hdc,
    [in] UINT     nSize,
    [in] const BYTE *lpData
);
```

Parameters

[in] **hdc**

A handle to an enhanced-metafile device context.

[in] **nSize**

The length of the comment buffer, in bytes.

[in] **lpData**

A pointer to the buffer that contains the comment.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

A comment can include any kind of private information, for example, the source of a picture and the date it was created. A comment should begin with an application signature, followed by the data.

Comments should not contain application-specific or position-specific data. Position-specific data specifies the location of a record, and it should not be included because one metafile may be embedded within another metafile.

A public comment is a comment that begins with the comment signature identifier **GDICOMMENT_IDENTIFIER**. The following public comments are defined.

GDICOMMENT_WINDOWS_METAFILE

The **GDICOMMENT_WINDOWS_METAFILE** public comment contains a Windows-format metafile that is equivalent to an enhanced-format metafile. This comment is written only by the [SetWinMetaFileBits](#) function. The comment record, if given, follows the [ENHMETAHEADER](#) metafile record. The comment has the following form:

```

DWORD ident;           // This contains GDICOMMENT_IDENTIFIER.
DWORD iComment;        // This contains GDICOMMENT_WINDOWS_METAFILE.
DWORD nVersion;        // This contains the version number of the
                      // Windows-format metafile.
DWORD nChecksum;       // This is the additive DWORD checksum for
                      // the enhanced metafile. The checksum
                      // for the enhanced metafile data including
                      // this comment record must be zero.
                      // Otherwise, the enhanced metafile has been
                      // modified and the Windows-format
                      // metafile is no longer valid.
DWORD fFlags;          // This must be zero.
DWORD cbWinMetaFile;   // This is the size, in bytes. of the
                      // Windows-format metafile data that follows.

```

GDICOMMENT_BEGINGROUP

The GDICOMMENT_BEGINGROUP public comment identifies the beginning of a group of drawing records. It identifies an object within an enhanced metafile. The comment has the following form:

```

DWORD ident;           // This contains GDICOMMENT_IDENTIFIER.
DWORD iComment;        // This contains GDICOMMENT_BEGINGROUP.
RECTL rclOutput;       // This is the bounding rectangle for the
                      // object in logical coordinates.
DWORD nDescription;   // This is the number of characters in the
                      // optional Unicode description string that
                      // follows. This is zero if there is no
                      // description string.

```

GDICOMMENT_ENDDGROUP

The GDICOMMENT_ENDDGROUP public comment identifies the end of a group of drawing records. The GDICOMMENT_BEGINGROUP comment and the GDICOMMENT_ENDDGROUP comment must be included in a pair and may be nested. The comment has the following form:

```

DWORD ident;           // This contains GDICOMMENT_IDENTIFIER.
DWORD iComment;        // This contains GDICOMMENT_ENDDGROUP.

```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SetWinMetaFileBits](#)

GdiFlush function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GdiFlush** function flushes the calling thread's current batch.

Syntax

```
BOOL GdiFlush();
```

Return value

If all functions in the current batch succeed, the return value is nonzero.

If not all functions in the current batch succeed, the return value is zero, indicating that at least one function returned an error.

Remarks

Batching enhances drawing performance by minimizing the amount of time needed to call GDI drawing functions that return Boolean values. The system accumulates the parameters for calls to these functions in the current batch and then calls the functions when the batch is flushed by any of the following means:

- Calling the **GdiFlush** function.
- Reaching or exceeding the batch limit set by the [GdiSetBatchLimit](#) function.
- Filling the batching buffers.
- Calling any GDI function that does not return a Boolean value.

The return value for **GdiFlush** applies only to the functions in the batch at the time **GdiFlush** is called. Errors that occur when the batch is flushed by any other means are never reported.

The [GdiGetBatchLimit](#) function returns the batch limit.

Note The batch limit is maintained for each thread separately. In order to completely disable batching, call [GdiSetBatchLimit](#)(1) during the initialization of each thread.

An application should call **GdiFlush** before a thread goes away if there is a possibility that there are pending function calls in the graphics batch queue. The system does not execute such batched functions when a thread goes away.

A multithreaded application that serializes access to GDI objects with a mutex must ensure flushing the GDI batch queue by calling **GdiFlush** as each thread releases ownership of the GDI object. This prevents collisions of the GDI objects (device contexts, metafiles, and so on).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GdiGetBatchLimit](#)

[GdiSetBatchLimit](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

GdiGetBatchLimit function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GdiGetBatchLimit** function returns the maximum number of function calls that can be accumulated in the calling thread's current batch. The system flushes the current batch whenever this limit is exceeded.

Syntax

```
DWORD GdiGetBatchLimit();
```

Return value

If the function succeeds, the return value is the batch limit.

If the function fails, the return value is zero.

Remarks

The batch limit is set by using the [GdiSetBatchLimit](#) function. Setting the limit to 1 effectively disables batching.

Only GDI drawing functions that return Boolean values can be batched; calls to any other GDI functions immediately flush the current batch. Exceeding the batch limit or calling the [GdiFlush](#) function also flushes the current batch.

When the system batches a function call, the function returns TRUE. The actual return value for the function is reported only if [GdiFlush](#) is used to flush the batch.

Note The batch limit is maintained for each thread separately. In order to completely disable batching, call [GdiSetBatchLimit](#) (1) during the initialization of each thread.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GdiFlush](#)

[GdiSetBatchLimit](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

GdiGradientFill function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GdiGradientFill** function fills rectangle and triangle structures.

Syntax

```
BOOL GdiGradientFill(
    [in] HDC         hdc,
    [in] PTRIVERTEX pVertex,
    [in] ULONG       nVertex,
    [in] PVOID       pMesh,
    [in] ULONG       nCount,
    [in] ULONG       ulMode
);
```

Parameters

[in] *hdc*

A handle to the destination device context.

[in] *pVertex*

A pointer to an array of [TRIVERTEX](#) structures that each define a triangle vertex.

[in] *nVertex*

The number of vertices in *pVertex*.

[in] *pMesh*

An array of [GRADIENT_TRIANGLE](#) structures in triangle mode, or an array of [GRADIENT_RECT](#) structures in rectangle mode.

[in] *nCount*

The number of elements (triangles or rectangles) in *pMesh*.

[in] *ulMode*

The gradient fill mode. This parameter can be one of the following values.

VALUE	MEANING
GRADIENT_FILL_RECT_H	In this mode, two endpoints describe a rectangle. The rectangle is defined to have a constant color (specified by the TRIVERTEX structure) for the left and right edges. GDI interpolates the color from the left to right edge and fills the interior.

GRADIENT_FILL_RECT_V	In this mode, two endpoints describe a rectangle. The rectangle is defined to have a constant color (specified by the TRIVERTEX structure) for the top and bottom edges. GDI interpolates the color from the top to bottom edge and fills the interior.
GRADIENT_FILL_TRIANGLE	In this mode, an array of TRIVERTEX structures is passed to GDI along with a list of array indexes that describe separate triangles. GDI performs linear interpolation between triangle vertices and fills the interior. Drawing is done directly in 24- and 32-bpp modes. Dithering is performed in 16-, 8-, 4-, and 1-bpp mode.

Return value

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**.

Remarks

Note This function is the same as [GradientFill](#).

To add smooth shading to a triangle, call the [GdiGradientFill](#) function with the three triangle endpoints. GDI will linearly interpolate and fill the triangle. Here is the drawing output of a shaded triangle.



To add smooth shading to a rectangle, call [GdiGradientFill](#) with the upper-left and lower-right coordinates of the rectangle. There are two shading modes used when drawing a rectangle. In horizontal mode, the rectangle is shaded from left-to-right. In vertical mode, the rectangle is shaded from top-to-bottom. Here is the drawing output of two shaded rectangles - one in horizontal mode, the other in



vertical mode. The

[GdiGradientFill](#) function uses a mesh method to specify the endpoints of the object to draw. All vertices are passed to [GdiGradientFill](#) in the *pVertex* array. The *pMesh* parameter specifies how these vertices are connected to form an object. When filling a rectangle, *pMesh* points to an array of [GRADIENT_RECT](#) structures. Each [GRADIENT_RECT](#) structure specifies the index of two vertices in the *pVertex* array. These two vertices form the upper-left and lower-right boundary of one rectangle.

In the case of filling a triangle, *pMesh* points to an array of [GRADIENT_TRIANGLE](#) structures. Each [GRADIENT_TRIANGLE](#) structure specifies the index of three vertices in the *pVertex* array. These three vertices form one triangle.

To simplify hardware acceleration, this routine is not required to be pixel-perfect in the triangle interior.

Note that [GdiGradientFill](#) does not use the Alpha member of the [TRIVERTEX](#) structure. To use [GdiGradientFill](#) with transparency, call [GdiGradientFill](#) and then call [GdiAlphaBlend](#) with the desired values for the alpha

channel of each vertex.

For more information, see [Smooth Shading](#), [Drawing a Shaded Triangle](#), and [Drawing a Shaded Rectangle](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[EMRGRADIENTFILL](#)

[GRADIENT_RECT](#)

[GRADIENT_TRIANGLE](#)

[TRIVERTEX](#)

GdiSetBatchLimit function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GdiSetBatchLimit** function sets the maximum number of function calls that can be accumulated in the calling thread's current batch. The system flushes the current batch whenever this limit is exceeded.

Syntax

```
DWORD GdiSetBatchLimit(  
    [in] DWORD dw  
) ;
```

Parameters

[in] dw

Specifies the batch limit to be set. A value of 0 sets the default limit. A value of 1 disables batching.

Return value

If the function succeeds, the return value is the previous batch limit.

If the function fails, the return value is zero.

Remarks

Only GDI drawing functions that return Boolean values can be accumulated in the current batch; calls to any other GDI functions immediately flush the current batch. Exceeding the batch limit or calling the [GdiFlush](#) function also flushes the current batch.

When the system accumulates a function call, the function returns **TRUE** to indicate it is in the batch. When the system flushes the current batch and executes the function for the second time, the return value is either **TRUE** or **FALSE**, depending on whether the function succeeds. This second return value is reported only if [GdiFlush](#) is used to flush the batch.

Note The batch limit is maintained for each thread separately. In order to completely disable batching, call [GdiSetBatchLimit](#) (1) during the initialization of each thread.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GdiFlush](#)

[GdiGetBatchLimit](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

GdiTransparentBlt function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GdiTransparentBlt** function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

Note This function is the same as [TransparentBlt](#).

Syntax

```
BOOL GdiTransparentBlt(
    [in] HDC hdcDest,
    [in] int xoriginDest,
    [in] int yoriginDest,
    [in] int wDest,
    [in] int hDest,
    [in] HDC hdcSrc,
    [in] int xoriginSrc,
    [in] int yoriginSrc,
    [in] int wSrc,
    [in] int hSrc,
    [in] UINT crTransparent
);
```

Parameters

[in] hdcDest

A handle to the destination device context.

[in] xoriginDest

The x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] yoriginDest

The y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] wDest

The width, in logical units, of the destination rectangle.

[in] hDest

The height, in logical units, of the destination rectangle.

[in] hdcSrc

A handle to the source device context.

[in] xoriginSrc

The x-coordinate, in logical units, of the source rectangle.

[in] *yoriginSrc*

The y-coordinate, in logical units, of the source rectangle.

[in] *wSrc*

The width, in logical units, of the source rectangle.

[in] *hSrc*

The height, in logical units, of the source rectangle.

[in] *crTransparent*

The RGB color in the source bitmap to treat as transparent.

Return value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE.

Remarks

The **GdiTransparentBlt** function works with compatible bitmaps (DDBs).

The [GdiTransparentBlt](#) function supports all formats of source bitmaps. However, for 32 bpp bitmaps, it just copies the alpha value over. Use [AlphaBlend](#) to specify 32 bits-per-pixel bitmaps with transparency.

If the source and destination rectangles are not the same size, the source bitmap is stretched to match the destination rectangle. When the [SetStretchBltMode](#) function is used, the *iStretchMode* modes of BLACKONWHITE and WHITEONBLACK are converted to COLORONCOLOR for the **GdiTransparentBlt** function.

The destination device context specifies the transformation type for the destination coordinates. The source device context specifies the transformation type for the source coordinates.

GdiTransparentBlt does not mirror a bitmap if either the width or height, of either the source or destination, is negative.

When used in a multiple monitor system, both *hdcSrc* and *hdcDest* must refer to the same device or the function will fail. To transfer data between DCs for different devices, convert the memory bitmap to a DIB by calling [GetDIBits](#). To display the DIB to the second device, call [SetDIBits](#) or [StretchDIBits](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[AlphaBlend](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[GetDIBits](#)

[SetDIBits](#)

[SetStretchBltMode](#)

[StretchDIBits](#)

GetArcDirection function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetArcDirection** function retrieves the current arc direction for the specified device context. Arc and rectangle functions use the arc direction.

Syntax

```
int GetArcDirection(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

Handle to the device context.

Return value

The return value specifies the current arc direction; it can be any one of the following values:

VALUE	MEANING
AD_COUNTERCLOCKWISE	Arcs and rectangles are drawn counterclockwise.
AD_CLOCKWISE	Arcs and rectangles are drawn clockwise.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

SetArcDirection

GetAspectRatioFilterEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetAspectRatioFilterEx** function retrieves the setting for the current aspect-ratio filter.

Syntax

```
BOOL GetAspectRatioFilterEx(
    [in]  HDC     hdc,
    [out] LPVOID lpsize
);
```

Parameters

[in] hdc

Handle to a device context.

[out] lpsize

Pointer to a **SIZE** structure that receives the current aspect-ratio filter.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The aspect ratio is the ratio formed by the width and height of a pixel on a specified device.

The system provides a special filter, the aspect-ratio filter, to select fonts that were designed for a particular device. An application can specify that the system should only retrieve fonts matching the specified aspect ratio by calling the [SetMapperFlags](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[SIZE](#)

[SetMapperFlags](#)

GetBitmapBits function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetBitmapBits** function copies the bitmap bits of a specified device-dependent bitmap into a buffer.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the [GetDIBits](#) function.

Syntax

```
LONG GetBitmapBits(
    [in] HBITMAP hbit,
    [in] LONG      cb,
    [out] LPVOID    lpvBits
);
```

Parameters

[in] `hbit`

A handle to the device-dependent bitmap.

[in] `cb`

The number of bytes to copy from the bitmap into the buffer.

[out] `lpvBits`

A pointer to a buffer to receive the bitmap bits. The bits are stored as an array of byte values.

Return value

If the function succeeds, the return value is the number of bytes copied to the buffer.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

GetBitmapDimensionEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetBitmapDimensionEx** function retrieves the dimensions of a compatible bitmap. The retrieved dimensions must have been set by the [SetBitmapDimensionEx](#) function.

Syntax

```
BOOL GetBitmapDimensionEx(
    [in] HBITMAP hbit,
    [out] LPSIZE lpsize
);
```

Parameters

[in] **hbit**

A handle to a compatible bitmap (DDB).

[out] **lpsize**

A pointer to a [SIZE](#) structure to receive the bitmap dimensions. For more information, see Remarks.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The function returns a data structure that contains fields for the height and width of the bitmap, in .01-mm units. If those dimensions have not yet been set, the structure that is returned will have zeros in those fields.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[SIZE](#)

[SetBitmapDimensionEx](#)

GetBkColor function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetBkColor** function returns the current background color for the specified device context.

Syntax

```
COLORREF GetBkColor(  
    [in] HDC hdc  
>);
```

Parameters

`[in] hdc`

Handle to the device context whose background color is to be returned.

Return value

If the function succeeds, the return value is a [COLORREF](#) value for the current background color.

If the function fails, the return value is `CLR_INVALID`.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORREF](#)

[GetBkMode](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[SetBkColor](#)

GetBkMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetBkMode** function returns the current background mix mode for a specified device context. The background mix mode of a device context affects text, hatched brushes, and pen styles that are not solid lines.

Syntax

```
int GetBkMode(  
    [in] HDC hdc  
>;
```

Parameters

[in] `hdc`

Handle to the device context whose background mode is to be returned.

Return value

If the function succeeds, the return value specifies the current background mix mode, either OPAQUE or TRANSPARENT.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GetBkColor](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[SetBkMode](#)

GetBoundsRect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetBoundsRect** function obtains the current accumulated bounding rectangle for a specified device context.

The system maintains an accumulated bounding rectangle for each application. An application can retrieve and set this rectangle.

Syntax

```
UINT GetBoundsRect(
    [in]    HDC      hdc,
    [out]   LPRECT   lprect,
    [in]    UINT     flags
);
```

Parameters

[in] `hdc`

A handle to the device context whose bounding rectangle the function will return.

[out] `lprect`

A pointer to the **RECT** structure that will receive the current bounding rectangle. The application's rectangle is returned in logical coordinates, and the bounding rectangle is returned in screen coordinates.

[in] `flags`

Specifies how the **GetBoundsRect** function will behave. This parameter can be the following value.

VALUE	MEANING
<code>DCB_RESET</code>	Clears the bounding rectangle after returning it. If this flag is not set, the bounding rectangle will not be cleared.

Return value

The return value specifies the state of the accumulated bounding rectangle; it can be one of the following values.

VALUE	MEANING
0	An error occurred. The specified device context handle is invalid.
<code>DCB_DISABLE</code>	Boundary accumulation is off.
<code>DCB_ENABLE</code>	Boundary accumulation is on.
<code>DCB_RESET</code>	The bounding rectangle is empty.

DCB_SET	The bounding rectangle is not empty.
---------	--------------------------------------

Remarks

The DCB_SET value is a combination of the bit values DCB_ACCUMULATE and DCB_RESET. Applications that check the DCB_RESET bit to determine whether the bounding rectangle is empty must also check the DCB_ACCUMULATE bit. The bounding rectangle is empty only if the DCB_RESET bit is 1 and the DCB_ACCUMULATE bit is 0.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[SetBoundsRect](#)

GetBrushOrgEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetBrushOrgEx** function retrieves the current brush origin for the specified device context. This function replaces the **GetBrushOrg** function.

Syntax

```
BOOL GetBrushOrgEx(
    [in]  HDC      hdc,
    [out] LPPOINT lppt
);
```

Parameters

[in] `hdc`

A handle to the device context.

[out] `lppt`

A pointer to a [POINT](#) structure that receives the brush origin, in device coordinates.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

A brush is a bitmap that the system uses to paint the interiors of filled shapes.

The brush origin is a set of coordinates with values between 0 and 7, specifying the location of one pixel in the bitmap. The default brush origin coordinates are (0,0). For horizontal coordinates, the value 0 corresponds to the leftmost column of pixels; the value 7 corresponds to the rightmost column. For vertical coordinates, the value 0 corresponds to the uppermost row of pixels; the value 7 corresponds to the lowermost row. When the system positions the brush at the start of any painting operation, it maps the origin of the brush to the location in the window's client area specified by the brush origin. For example, if the origin is set to (2,3), the system maps the origin of the brush (0,0) to the location (2,3) on the window's client area.

If an application uses a brush to fill the backgrounds of both a parent and a child window with matching colors, it may be necessary to set the brush origin after painting the parent window but before painting the child window.

The system automatically tracks the origin of all window-managed device contexts and adjusts their brushes as necessary to maintain an alignment of patterns on the surface.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Brush Functions](#)

[Brushes Overview](#)

[POINT](#)

[SelectObject](#)

[SetBrushOrgEx](#)

[UnrealizeObject](#)

GetBValue macro (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetBValue** macro retrieves an intensity value for the blue component of a red, green, blue (RGB) value.

Syntax

```
void GetBValue(  
    rgb  
) ;
```

Parameters

rgb

Specifies an RGB color value.

Return value

None

Remarks

The intensity value is in the range 0 through 255.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[Color Macros](#)

[Colors Overview](#)

[GetGValue](#)

[GetRValue](#)

[PALETTEINDEX](#)

[PALETTERGB](#)

RGB

GetCharABCWidthsA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharABCWidths** function retrieves the widths, in logical units, of consecutive characters in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.

Syntax

```
BOOL GetCharABCWidthsA(
    [in] HDC    hdc,
    [in] UINT   wFirst,
    [in] UINT   wLast,
    [out] LPABC lpABC
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `wFirst`

The first character in the group of consecutive characters from the current font.

[in] `wLast`

The last character in the group of consecutive characters from the current font.

[out] `lpABC`

A pointer to an array of [ABC](#) structures that receives the character widths, in logical units. This array must contain at least as many [ABC](#) structures as there are characters in the range specified by the *uFirstChar* and *uLastChar* parameters.

Return value

If the function succeeds, the return value is nonzero

If the function fails, the return value is zero.

Remarks

The TrueType rasterizer provides ABC character spacing after a specific point size has been selected. A spacing is the distance added to the current position before placing the glyph. B spacing is the width of the black part of the glyph. C spacing is the distance added to the current position to provide white space to the right of the glyph. The total advanced width is specified by A+B+C.

When the **GetCharABCWidths** function retrieves negative A or C widths for a character, that character includes underhangs or overhangs.

To convert the ABC widths to font design units, an application should use the value stored in the **otmEMSSquare** member of a [OUTLINETEXTMETRIC](#) structure. This value can be retrieved by calling the [GetOutlineTextMetrics](#)

function.

The ABC widths of the default character are used for characters outside the range of the currently selected font.

To retrieve the widths of characters in non-TrueType fonts, applications should use the [GetCharWidth](#) function.

NOTE

The wingdi.h header defines GetCharABCWidths as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ABC](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharWidth](#)

[GetOutlineTextMetrics](#)

[OUTLINETEXTMETRIC](#)

GetCharABCWidthsFloatA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharABCWidthsFloat** function retrieves the widths, in logical units, of consecutive characters in a specified range from the current font.

Syntax

```
BOOL GetCharABCWidthsFloatA(
    [in]    HDC      hdc,
    [in]    UINT     iFirst,
    [in]    UINT     iLast,
    [out]   LPABCFLOAT lpABC
);
```

Parameters

[in] hdc

Handle to the device context.

[in] iFirst

Specifies the code point of the first character in the group of consecutive characters where the ABC widths are seeked.

[in] iLast

Specifies the code point of the last character in the group of consecutive characters where the ABC widths are seeked. This range is inclusive. An error is returned if the specified last character precedes the specified first character.

[out] lpABC

Pointer to an array of **ABCFLOAT** structures that receives the character widths, in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Unlike the **GetCharABCWidths** function that returns widths only for TrueType fonts, the **GetCharABCWidthsFloat** function retrieves widths for any font. The widths returned by this function are in the IEEE floating-point format.

If the current world-to-device transformation is not identified, the returned widths may be noninteger values, even if the corresponding values in the device space are integers.

A spacing is the distance added to the current position before placing the glyph. B spacing is the width of the black part of the glyph. C spacing is the distance added to the current position to provide white space to the

right of the glyph. The total advanced width is specified by A+B+C.

The ABC spaces are measured along the character base line of the selected font.

The ABC widths of the default character are used for characters outside the range of the currently selected font.

NOTE

The wingdi.h header defines GetCharABCWidthsFloat as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ABCFLOAT](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharWidth](#)

[GetCharWidthFloat](#)

GetCharABCWidthsFloatW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharABCWidthsFloat** function retrieves the widths, in logical units, of consecutive characters in a specified range from the current font.

Syntax

```
BOOL GetCharABCWidthsFloatW(
    [in]    HDC      hdc,
    [in]    UINT     iFirst,
    [in]    UINT     iLast,
    [out]   LPABCFLOAT lpABC
);
```

Parameters

[in] hdc

Handle to the device context.

[in] iFirst

Specifies the code point of the first character in the group of consecutive characters where the ABC widths are seeked.

[in] iLast

Specifies the code point of the last character in the group of consecutive characters where the ABC widths are seeked. This range is inclusive. An error is returned if the specified last character precedes the specified first character.

[out] lpABC

Pointer to an array of **ABCFLOAT** structures that receives the character widths, in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Unlike the [GetCharABCWidths](#) function that returns widths only for TrueType fonts, the **GetCharABCWidthsFloat** function retrieves widths for any font. The widths returned by this function are in the IEEE floating-point format.

If the current world-to-device transformation is not identified, the returned widths may be noninteger values, even if the corresponding values in the device space are integers.

A spacing is the distance added to the current position before placing the glyph. B spacing is the width of the black part of the glyph. C spacing is the distance added to the current position to provide white space to the

right of the glyph. The total advanced width is specified by A+B+C.

The ABC spaces are measured along the character base line of the selected font.

The ABC widths of the default character are used for characters outside the range of the currently selected font.

NOTE

The wingdi.h header defines GetCharABCWidthsFloat as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ABCFLOAT](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharWidth](#)

[GetCharWidthFloat](#)

GetCharABCWidthsI function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharABCWidthsI** function retrieves the widths, in logical units, of consecutive glyph indices in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.

Syntax

```
BOOL GetCharABCWidthsI(
    [in] HDC     hdc,
    [in] UINT    giFirst,
    [in] UINT    cgi,
    [in] LPWORD  pgi,
    [out] LPABC   pabc
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `giFirst`

The first glyph index in the group of consecutive glyph indices from the current font. This parameter is only used if the `pgi` parameter is **NULL**.

[in] `cgi`

The number of glyph indices.

[in] `pgi`

A pointer to an array that contains glyph indices. If this parameter is **NULL**, the `giFirst` parameter is used instead. The `cgi` parameter specifies the number of glyph indices in this array.

[out] `pabc`

A pointer to an array of **ABC** structures that receives the character widths, in logical units. This array must contain at least as many **ABC** structures as there are glyph indices specified by the `cgi` parameter.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The TrueType rasterizer provides ABC character spacing after a specific point size has been selected. A spacing is the distance added to the current position before placing the glyph. B spacing is the width of the black part of the glyph. C spacing is the distance added to the current position to provide white space to the right of the glyph. The total advanced width is specified by A+B+C.

When the [GetCharABCWidthsI](#) function retrieves negative A or C widths for a character, that character includes underhangs or overhangs.

To convert the ABC widths to font design units, an application should use the value stored in the **otmEMSSquare** member of a [OUTLINETEXTMETRIC](#) structure. This value can be retrieved by calling the [GetOutlineTextMetrics](#) function.

The ABC widths of the default character are used for characters outside the range of the currently selected font.

To retrieve the widths of glyph indices in non-TrueType fonts, applications should use the [GetCharWidthI](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ABC](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharWidth](#)

[GetOutlineTextMetrics](#)

[OUTLINETEXTMETRIC](#)

GetCharABCWidthsW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharABCWidths** function retrieves the widths, in logical units, of consecutive characters in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.

Syntax

```
BOOL GetCharABCWidthsW(
    [in] HDC    hdc,
    [in] UINT   wFirst,
    [in] UINT   wLast,
    [out] LPABC lpABC
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `wFirst`

The first character in the group of consecutive characters from the current font.

[in] `wLast`

The last character in the group of consecutive characters from the current font.

[out] `lpABC`

A pointer to an array of [ABC](#) structures that receives the character widths, in logical units. This array must contain at least as many ABC structures as there are characters in the range specified by the *uFirstChar* and *uLastChar* parameters.

Return value

If the function succeeds, the return value is nonzero

If the function fails, the return value is zero.

Remarks

The TrueType rasterizer provides ABC character spacing after a specific point size has been selected. A spacing is the distance added to the current position before placing the glyph. B spacing is the width of the black part of the glyph. C spacing is the distance added to the current position to provide white space to the right of the glyph. The total advanced width is specified by A+B+C.

When the **GetCharABCWidths** function retrieves negative A or C widths for a character, that character includes underhangs or overhangs.

To convert the ABC widths to font design units, an application should use the value stored in the **otmEMSSquare** member of a [OUTLINETEXTMETRIC](#) structure. This value can be retrieved by calling the [GetOutlineTextMetrics](#)

function.

The ABC widths of the default character are used for characters outside the range of the currently selected font.

To retrieve the widths of characters in non-TrueType fonts, applications should use the [GetCharWidth](#) function.

NOTE

The wingdi.h header defines GetCharABCWidths as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ABC](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharWidth](#)

[GetOutlineTextMetrics](#)

[OUTLINETEXTMETRIC](#)

GetCharacterPlacementA function (wingdi.h)

4/21/2022 • 10 minutes to read • [Edit Online](#)

The **GetCharacterPlacement** function retrieves information about a character string, such as character widths, caret positioning, ordering within the string, and glyph rendering. The type of information returned depends on the *dwFlags* parameter and is based on the currently selected font in the specified display context. The function copies the information to the specified [GCP_RESULTS](#) structure or to one or more arrays specified by the structure.

Although this function was once adequate for working with character strings, a need to work with an increasing number of languages and scripts has rendered it obsolete. It has been superseded by the functionality of the Uniscribe module. For more information, see [Uniscribe](#).

It is recommended that an application use the [GetFontLanguageInfo](#) function to determine whether the GCP_DIACRITIC, GCP_DBCS, GCP_USEKERNING, GCP_LIGATE, GCP_reordered, GCP_GLYPHSHAPE, and GCP_KASHIDA values are valid for the currently selected font. If not valid, **GetCharacterPlacement** ignores the value.

The GCP_NODIACRITICS value is no longer defined and should not be used.

Syntax

```
DWORD GetCharacterPlacementA(
    [in]      HDC          hdc,
    [in]      LPCSTR       lpString,
    [in]      int          nCount,
    [in]      int          nMexExtent,
    [in, out] LPGCP_RESULTS_A lpResults,
    [in]      DWORD         dwFlags
);
```

Parameters

[in] hdc

A handle to the device context.

[in] lpString

A pointer to the character string to process. The string does not need to be zero-terminated, since *nCount* specifies the length of the string.

[in] nCount

The [length of the string](#) pointed to by *lpString*.

[in] nMexExtent

The maximum extent (in logical units) to which the string is processed. Characters that, if processed, would exceed this extent are ignored. Computations for any required ordering or glyph arrays apply only to the included characters. This parameter is used only if the GCP_MAXEXTENT value is specified in the *dwFlags* parameter. As the function processes the input string, each character and its extent is added to the output, extent, and other arrays only if the total extent has not yet exceeded the maximum. Once the limit is reached,

processing will stop.

[in, out] lpResults

A pointer to a [GCP_RESULTS](#) structure that receives the results of the function.

[in] dwFlags

Specifies how to process the string into the required arrays. This parameter can be one or more of the following values.

VALUE	MEANING
GCP_CLASSIN	Specifies that the <i>lpClass</i> array contains preset classifications for characters. The classifications may be the same as on output. If the particular classification for a character is not known, the corresponding location in the array must be set to zero. For more information about the classifications, see GCP_RESULTS . This is useful only if GetFontLanguageInfo returned the GCP_REORDER flag.
GCP_DIACRITIC	Determines how diacritics in the string are handled. If this value is not set, diacritics are treated as zero-width characters. For example, a Hebrew string may contain diacritics, but you may not want to display them. Use GetFontLanguageInfo to determine whether a font supports diacritics. If it does, you can use or not use the GCP_DIACRITIC flag in the call to GetCharacterPlacement , depending on the needs of your application.
GCP_DISPLAYZWG	For languages that need reordering or different glyph shapes depending on the positions of the characters within a word, nondisplayable characters often appear in the code page. For example, in the Hebrew code page, there are Left-To-Right and Right-To-Left markers, to help determine the final positioning of characters within the output strings. Normally these are not displayed and are removed from the <i>lpGlyphs</i> and <i>lpDx</i> arrays. You can use the GCP_DISPLAYZWG flag to display these characters.
GCP_GLYPHSHAPE	Specifies that some or all characters in the string are to be displayed using shapes other than the standard shapes defined in the currently selected font for the current code page. Some languages, such as Arabic, cannot support glyph creation unless this value is specified. As a general rule, if GetFontLanguageInfo returns this value for a string, this value must be used with GetCharacterPlacement .
GCP_JUSTIFY	Adjusts the extents in the <i>lpDx</i> array so that the string length is the same as <i>nMaxExtent</i> . GCP_JUSTIFY may only be used in conjunction with GCP_MAXEXTENT.

GCP_KASHIDA	<p>Use Kashidas as well as, or instead of, adjusted extents to modify the length of the string so that it is equal to the value specified by <i>nMaxExtent</i>. In the <i>lpDx</i> array, a Kashida is indicated by a negative justification index. GCP_KASHIDA may be used only in conjunction with GCP_JUSTIFY and only if the font (and language) support Kashidas. Use GetFontLanguageInfo to determine whether the current font supports Kashidas.</p> <p>Using Kashidas to justify the string can result in the number of glyphs required being greater than the number of characters in the input string. Because of this, when Kashidas are used, the application cannot assume that setting the arrays to be the size of the input string will be sufficient. (The maximum possible will be approximately <i>dxPageWidth/dxAveCharWidth</i>, where <i>dxPageWidth</i> is the width of the document and <i>dxAveCharWidth</i> is the average character width as returned from a GetTextMetrics call).</p> <p>Note that just because GetFontLanguageInfo returns the GCP_KASHIDA flag does not mean that it has to be used in the call to GetCharacterPlacement, just that the option is available.</p>
GCP_LIGATE	<p>Use ligations wherever characters ligate. A ligation occurs where one glyph is used for two or more characters. For example, the letters a and e can ligate to ?. For this to be used, however, both the language support and the font must support the required glyphs (the example will not be processed by default in English).</p> <p>Use GetFontLanguageInfo to determine whether the current font supports ligation. If it does and a specific maximum is required for the number of characters that will ligate, set the number in the first element of the <i>lpGlyphs</i> array. If normal ligation is required, set this value to zero. If GCP_LIGATE is not specified, no ligation will take place. See GCP_RESULTS for more information.</p> <p>If the GCP_REORDER value is usually required for the character set but is not specified, the output will be meaningless unless the string being passed in is already in visual ordering (that is, the result that gets put into <i>lpGcpResults->lpOutString</i> in one call to GetCharacterPlacement is the input string of a second call).</p> <p>Note that just because GetFontLanguageInfo returns the GCP_LIGATE flag does not mean that it has to be used in the call to GetCharacterPlacement, just that the option is available.</p>
GCP_MAXEXTENT	<p>Compute extents of the string only as long as the resulting extent, in logical units, does not exceed the values specified by the <i>nMaxExtent</i> parameter.</p>
GCP_NEUTRALOVERRIDE	<p>Certain languages only. Override the normal handling of neutrals and treat them as strong characters that match the strings reading order. Useful only with the GCP_REORDER flag.</p>

GCP_NUMERICOVERRIDE	Certain languages only. Override the normal handling of numerics and treat them as strong characters that match the strings reading order. Useful only with the GCP_REORDER flag.
GCP_NUMERICSLATIN	Arabic/Thai only. Use standard Latin glyphs for numbers and override the system default. To determine if this option is available in the language of the font, use GetStringTypeEx to see if the language supports more than one number format.
GCP_NUMERICSLOCAL	Arabic/Thai only. Use local glyphs for numeric characters and override the system default. To determine if this option is available in the language of the font, use GetStringTypeEx to see if the language supports more than one number format.
GCP_REORDER	<p>Reorder the string. Use for languages that are not SBCS and left-to-right reading order. If this value is not specified, the string is assumed to be in display order already.</p> <p>If this flag is set for Semitic languages and the <i>lpClass</i> array is used, the first two elements of the array are used to specify the reading order beyond the bounds of the string. GCP_CLASS_PREBOUNDRTL and GCP_CLASS_PREBOUNDLTR can be used to set the order. If no preset order is required, set the values to zero. These values can be combined with other values if the GCPCLASSIN flag is set.</p> <p>If the GCP_REORDER value is not specified, the <i>lpString</i> parameter is taken to be visual ordered for languages where this is used, and the <i>lpOutString</i> and <i>lpOrder</i> fields are ignored.</p> <p>Use GetFontLanguageInfo to determine whether the current font supports reordering.</p>
GCP_SYMSWAPOFF	Semitic languages only. Specifies that swappable characters are not reset. For example, in a right-to-left string, the '(' and ')' are not reversed.
GCP_USEKERNING	<p>Use kerning pairs in the font (if any) when creating the widths arrays. Use GetFontLanguageInfo to determine whether the current font supports kerning pairs.</p> <p>Note that just because GetFontLanguageInfo returns the GCP_USEKERNING flag does not mean that it has to be used in the call to GetCharacterPlacement, just that the option is available. Most TrueType fonts have a kerning table, but you do not have to use it.</p>

It is recommended that an application use the [GetFontLanguageInfo](#) function to determine whether the GCP_DIACRITIC, GCP_DBCS, GCP_USEKERNING, GCP_LIGATE, GCP_REORDER, GCP_GLYPHSHAPE, and GCP_KASHIDA values are valid for the currently selected font. If not valid, [GetCharacterPlacement](#) ignores the value.

The GCP_NODIACRITICS value is no longer defined and should not be used.

Return value

If the function succeeds, the return value is the width and height of the string in logical units. The width is the

low-order word and the height is the high-order word.

If the function fails, the return value is zero.

Remarks

GetCharacterPlacement ensures that an application can correctly process text regardless of the international setting and type of fonts available. Applications use this function before using the [ExtTextOut](#) function and in place of the [GetTextExtentPoint32](#) function (and occasionally in place of the [GetCharWidth32](#) and [GetCharABCWidths](#) functions).

Using **GetCharacterPlacement** to retrieve intercharacter spacing and index arrays is not always necessary unless justification or kerning is required. For non-Latin fonts, applications can improve the speed at which the [ExtTextOut](#) function renders text by using **GetCharacterPlacement** to retrieve the intercharacter spacing and index arrays before calling [ExtTextOut](#). This is especially useful when rendering the same text repeatedly or when using intercharacter spacing to position the caret. If the **IpGlyphs** output array is used in the call to [ExtTextOut](#), the ETO_GLYPH_INDEX flag must be set.

GetCharacterPlacement checks the **IpOrder**, **IpDX**, **IpCaretPos**, **IpOutString**, and **IpGlyphs** members of the [GCP_RESULTS](#) structure and fills the corresponding arrays if these members are not set to **NULL**. If **GetCharacterPlacement** cannot fill an array, it sets the corresponding member to **NULL**. To ensure retrieval of valid information, the application is responsible for setting the member to a valid address before calling the function and for checking the value of the member after the call. If the **GCP_JUSTIFY** or **GCP_USEKERNING** values are specified, the **IpDX** and/or **IpCaretPos** members must have valid addresses.

Note that the glyph indexes returned in **GCP_RESULTS.IpGlyphs** are specific to the current font in the device context and should only be used to draw text in the device context while that font remains selected.

When computing justification, if the trailing characters in the string are spaces, the function reduces the length of the string and removes the spaces prior to computing the justification. If the array consists of only spaces, the function returns an error.

[ExtTextOut](#) expects an **IpDX** entry for each byte of a DBCS string, whereas **GetCharacterPlacement** assigns an **IpDX** entry for each glyph. To correct this mismatch when using this combination of functions, either use [GetGlyphIndices](#) or expand the **IpDX** array with zero-width entries for the corresponding second byte of a DBCS byte pair.

If the logical width is less than the width of the leading character in the input string, **GCP_RESULTS.nMaxFit** returns a bad value. For this case, call **GetCharacterPlacement** for glyph indexes and the **IpDX** array. Then use the **IpDX** array to do the extent calculation using the advance width of each character, where **nMaxFit** is the number of characters whose glyph indexes advance width is less than the width of the leading character.

NOTE

The wingdi.h header defines **GetCharacterPlacement** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GCP_RESULTS](#)

[GetCharABCWidths](#)

[GetCharWidth32](#)

[GetFontLanguageInfo](#)

[GetStringTypeEx](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

GetCharacterPlacementW function (wingdi.h)

4/21/2022 • 10 minutes to read • [Edit Online](#)

The **GetCharacterPlacement** function retrieves information about a character string, such as character widths, caret positioning, ordering within the string, and glyph rendering. The type of information returned depends on the *dwFlags* parameter and is based on the currently selected font in the specified display context. The function copies the information to the specified **GCP_RESULTS** structure or to one or more arrays specified by the structure.

Although this function was once adequate for working with character strings, a need to work with an increasing number of languages and scripts has rendered it obsolete. It has been superseded by the functionality of the Uniscribe module. For more information, see [Uniscribe](#).

It is recommended that an application use the [GetFontLanguageInfo](#) function to determine whether the GCP_DIACRITIC, GCP_DBCS, GCP_USEKERNING, GCP_LIGATE, GCP_REORDER, GCP_GLYPHSHAPE, and GCP_KASHIDA values are valid for the currently selected font. If not valid, **GetCharacterPlacement** ignores the value.

The GCP_NODIACRITICS value is no longer defined and should not be used.

Syntax

```
DWORD GetCharacterPlacementW(
    [in]      HDC          hdc,
    [in]      LPCWSTR     lpString,
    [in]      int          nCount,
    [in]      int          nMexExtent,
    [in, out] LPGCP_RESULTSW lpResults,
    [in]      DWORD        dwFlags
);
```

Parameters

[in] hdc

A handle to the device context.

[in] lpString

A pointer to the character string to process. The string does not need to be zero-terminated, since *nCount* specifies the length of the string.

[in] nCount

The [length of the string](#) pointed to by *lpString*.

[in] nMexExtent

The maximum extent (in logical units) to which the string is processed. Characters that, if processed, would exceed this extent are ignored. Computations for any required ordering or glyph arrays apply only to the included characters. This parameter is used only if the GCP_MAXEXTENT value is specified in the *dwFlags* parameter. As the function processes the input string, each character and its extent is added to the output, extent, and other arrays only if the total extent has not yet exceeded the maximum. Once the limit is reached,

processing will stop.

[in, out] lpResults

A pointer to a [GCP_RESULTS](#) structure that receives the results of the function.

[in] dwFlags

Specifies how to process the string into the required arrays. This parameter can be one or more of the following values.

VALUE	MEANING
GCP_CLASSIN	Specifies that the <i>lpClass</i> array contains preset classifications for characters. The classifications may be the same as on output. If the particular classification for a character is not known, the corresponding location in the array must be set to zero. For more information about the classifications, see GCP_RESULTS . This is useful only if GetFontLanguageInfo returned the GCP_reordered flag.
GCP_DIACRITIC	Determines how diacritics in the string are handled. If this value is not set, diacritics are treated as zero-width characters. For example, a Hebrew string may contain diacritics, but you may not want to display them. Use GetFontLanguageInfo to determine whether a font supports diacritics. If it does, you can use or not use the GCP_DIACRITIC flag in the call to GetCharacterPlacement , depending on the needs of your application.
GCP_DISPLAYZWG	For languages that need reordering or different glyph shapes depending on the positions of the characters within a word, nondisplayable characters often appear in the code page. For example, in the Hebrew code page, there are Left-To-Right and Right-To-Left markers, to help determine the final positioning of characters within the output strings. Normally these are not displayed and are removed from the <i>lpGlyphs</i> and <i>lpDx</i> arrays. You can use the GCP_DISPLAYZWG flag to display these characters.
GCP_GLYPHSHAPE	Specifies that some or all characters in the string are to be displayed using shapes other than the standard shapes defined in the currently selected font for the current code page. Some languages, such as Arabic, cannot support glyph creation unless this value is specified. As a general rule, if GetFontLanguageInfo returns this value for a string, this value must be used with GetCharacterPlacement .
GCP_JUSTIFY	Adjusts the extents in the <i>lpDx</i> array so that the string length is the same as <i>nMaxExtent</i> . GCP_JUSTIFY may only be used in conjunction with GCP_MAXEXTENT.

GCP_KASHIDA	<p>Use Kashidas as well as, or instead of, adjusted extents to modify the length of the string so that it is equal to the value specified by <i>nMaxExtent</i>. In the <i>lpDx</i> array, a Kashida is indicated by a negative justification index. GCP_KASHIDA may be used only in conjunction with GCP_JUSTIFY and only if the font (and language) support Kashidas. Use GetFontLanguageInfo to determine whether the current font supports Kashidas.</p> <p>Using Kashidas to justify the string can result in the number of glyphs required being greater than the number of characters in the input string. Because of this, when Kashidas are used, the application cannot assume that setting the arrays to be the size of the input string will be sufficient. (The maximum possible will be approximately <i>dxPageWidth/dxAveCharWidth</i>, where <i>dxPageWidth</i> is the width of the document and <i>dxAveCharWidth</i> is the average character width as returned from a GetTextMetrics call).</p> <p>Note that just because GetFontLanguageInfo returns the GCP_KASHIDA flag does not mean that it has to be used in the call to GetCharacterPlacement, just that the option is available.</p>
GCP_LIGATE	<p>Use ligations wherever characters ligate. A ligation occurs where one glyph is used for two or more characters. For example, the letters a and e can ligate to ?. For this to be used, however, both the language support and the font must support the required glyphs (the example will not be processed by default in English).</p> <p>Use GetFontLanguageInfo to determine whether the current font supports ligation. If it does and a specific maximum is required for the number of characters that will ligate, set the number in the first element of the <i>lpGlyphs</i> array. If normal ligation is required, set this value to zero. If GCP_LIGATE is not specified, no ligation will take place. See GCP_RESULTS for more information.</p> <p>If the GCP_REORDER value is usually required for the character set but is not specified, the output will be meaningless unless the string being passed in is already in visual ordering (that is, the result that gets put into <i>lpGcpResults->lpOutString</i> in one call to GetCharacterPlacement is the input string of a second call).</p> <p>Note that just because GetFontLanguageInfo returns the GCP_LIGATE flag does not mean that it has to be used in the call to GetCharacterPlacement, just that the option is available.</p>
GCP_MAXEXTENT	<p>Compute extents of the string only as long as the resulting extent, in logical units, does not exceed the values specified by the <i>nMaxExtent</i> parameter.</p>
GCP_NEUTRAL OVERRIDE	<p>Certain languages only. Override the normal handling of neutrals and treat them as strong characters that match the strings reading order. Useful only with the GCP_REORDER flag.</p>

GCP_NUMERICOVERRIDE	Certain languages only. Override the normal handling of numerics and treat them as strong characters that match the strings reading order. Useful only with the GCP_REORDER flag.
GCP_NUMERICSLATIN	Arabic/Thai only. Use standard Latin glyphs for numbers and override the system default. To determine if this option is available in the language of the font, use GetStringTypeEx to see if the language supports more than one number format.
GCP_NUMERICSLOCAL	Arabic/Thai only. Use local glyphs for numeric characters and override the system default. To determine if this option is available in the language of the font, use GetStringTypeEx to see if the language supports more than one number format.
GCP_REORDER	<p>Reorder the string. Use for languages that are not SBCS and left-to-right reading order. If this value is not specified, the string is assumed to be in display order already.</p> <p>If this flag is set for Semitic languages and the <i>lpClass</i> array is used, the first two elements of the array are used to specify the reading order beyond the bounds of the string. GCP_CLASS_PREBOUNDRTL and GCP_CLASS_PREBOUNDLTR can be used to set the order. If no preset order is required, set the values to zero. These values can be combined with other values if the GCPCLASSIN flag is set.</p> <p>If the GCP_REORDER value is not specified, the <i>lpString</i> parameter is taken to be visual ordered for languages where this is used, and the <i>lpOutString</i> and <i>lpOrder</i> fields are ignored.</p> <p>Use GetFontLanguageInfo to determine whether the current font supports reordering.</p>
GCP_SYMSWAPOFF	Semitic languages only. Specifies that swappable characters are not reset. For example, in a right-to-left string, the '(' and ')' are not reversed.
GCP_USEKERNING	<p>Use kerning pairs in the font (if any) when creating the widths arrays. Use GetFontLanguageInfo to determine whether the current font supports kerning pairs.</p> <p>Note that just because GetFontLanguageInfo returns the GCP_USEKERNING flag does not mean that it has to be used in the call to GetCharacterPlacement, just that the option is available. Most TrueType fonts have a kerning table, but you do not have to use it.</p>

It is recommended that an application use the [GetFontLanguageInfo](#) function to determine whether the GCP_DIACRITIC, GCP_DBCS, GCP_USEKERNING, GCP_LIGATE, GCP_REORDER, GCP_GLYPHSHAPE, and GCP_KASHIDA values are valid for the currently selected font. If not valid, [GetCharacterPlacement](#) ignores the value.

The GCP_NODIACRITICS value is no longer defined and should not be used.

Return value

If the function succeeds, the return value is the width and height of the string in logical units. The width is the

low-order word and the height is the high-order word.

If the function fails, the return value is zero.

Remarks

GetCharacterPlacement ensures that an application can correctly process text regardless of the international setting and type of fonts available. Applications use this function before using the [ExtTextOut](#) function and in place of the [GetTextExtentPoint32](#) function (and occasionally in place of the [GetCharWidth32](#) and [GetCharABCWidths](#) functions).

Using **GetCharacterPlacement** to retrieve intercharacter spacing and index arrays is not always necessary unless justification or kerning is required. For non-Latin fonts, applications can improve the speed at which the [ExtTextOut](#) function renders text by using **GetCharacterPlacement** to retrieve the intercharacter spacing and index arrays before calling [ExtTextOut](#). This is especially useful when rendering the same text repeatedly or when using intercharacter spacing to position the caret. If the **IpGlyphs** output array is used in the call to [ExtTextOut](#), the ETO_GLYPH_INDEX flag must be set.

GetCharacterPlacement checks the **IpOrder**, **IpDX**, **IpCaretPos**, **IpOutString**, and **IpGlyphs** members of the [GCP_RESULTS](#) structure and fills the corresponding arrays if these members are not set to **NULL**. If **GetCharacterPlacement** cannot fill an array, it sets the corresponding member to **NULL**. To ensure retrieval of valid information, the application is responsible for setting the member to a valid address before calling the function and for checking the value of the member after the call. If the **GCP_JUSTIFY** or **GCP_USEKERNING** values are specified, the **IpDX** and/or **IpCaretPos** members must have valid addresses.

Note that the glyph indexes returned in **GCP_RESULTS.IpGlyphs** are specific to the current font in the device context and should only be used to draw text in the device context while that font remains selected.

When computing justification, if the trailing characters in the string are spaces, the function reduces the length of the string and removes the spaces prior to computing the justification. If the array consists of only spaces, the function returns an error.

[ExtTextOut](#) expects an **IpDX** entry for each byte of a DBCS string, whereas **GetCharacterPlacement** assigns an **IpDX** entry for each glyph. To correct this mismatch when using this combination of functions, either use [GetGlyphIndices](#) or expand the **IpDX** array with zero-width entries for the corresponding second byte of a DBCS byte pair.

If the logical width is less than the width of the leading character in the input string, **GCP_RESULTS.nMaxFit** returns a bad value. For this case, call **GetCharacterPlacement** for glyph indexes and the **IpDX** array. Then use the **IpDX** array to do the extent calculation using the advance width of each character, where **nMaxFit** is the number of characters whose glyph indexes advance width is less than the width of the leading character.

NOTE

The wingdi.h header defines **GetCharacterPlacement** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GCP_RESULTS](#)

[GetCharABCWidths](#)

[GetCharWidth32](#)

[GetFontLanguageInfo](#)

[GetStringTypeEx](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

GetCharWidth32A function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharWidth32** function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

Syntax

```
BOOL GetCharWidth32A(
    [in]  HDC   hdc,
    [in]  UINT  iFirst,
    [in]  UINT  iLast,
    [out] LPINT lpBuffer
);
```

Parameters

[in] hdc

A handle to the device context.

[in] iFirst

The first character in the group of consecutive characters.

[in] iLast

The last character in the group of consecutive characters, which must not precede the specified first character.

[out] lpBuffer

A pointer to a buffer that receives the character widths, in logical coordinates.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

GetCharWidth32 cannot be used on TrueType fonts. To retrieve character widths for TrueType fonts, use [GetCharABCWidths](#).

The range is inclusive; that is, the returned widths include the widths of the characters specified by the *iFirstChar* and *iLastChar* parameters.

If a character does not exist in the current font, it is assigned the width of the default character.

Examples

For an example, see "Displaying Keyboard Input" in [Using Keyboard Input](#).

NOTE

The wingdi.h header defines GetCharWidth32 as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharABCWidthsFloat](#)

[GetCharWidthFloat](#)

GetCharWidth32W function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharWidth32** function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

Syntax

```
BOOL GetCharWidth32W(
    [in]  HDC   hdc,
    [in]  UINT  iFirst,
    [in]  UINT  iLast,
    [out] LPINT lpBuffer
);
```

Parameters

[in] hdc

A handle to the device context.

[in] iFirst

The first character in the group of consecutive characters.

[in] iLast

The last character in the group of consecutive characters, which must not precede the specified first character.

[out] lpBuffer

A pointer to a buffer that receives the character widths, in logical coordinates.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

GetCharWidth32 cannot be used on TrueType fonts. To retrieve character widths for TrueType fonts, use [GetCharABCWidths](#).

The range is inclusive; that is, the returned widths include the widths of the characters specified by the *iFirstChar* and *iLastChar* parameters.

If a character does not exist in the current font, it is assigned the width of the default character.

Examples

For an example, see "Displaying Keyboard Input" in [Using Keyboard Input](#).

NOTE

The wingdi.h header defines GetCharWidth32 as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharABCWidthsFloat](#)

[GetCharWidthFloat](#)

GetCharWidthA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharWidth** function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should call the [GetCharWidth32](#) function, which provides more accurate results.

Syntax

```
BOOL GetCharWidthA(
    [in]  HDC    hdc,
    [in]  UINT   iFirst,
    [in]  UINT   iLast,
    [out] LPINT  lpBuffer
);
```

Parameters

[in] hdc

A handle to the device context.

[in] iFirst

The first character in the group of consecutive characters.

[in] iLast

The last character in the group of consecutive characters, which must not precede the specified first character.

[out] lpBuffer

A pointer to a buffer that receives the character widths, in logical coordinates.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

GetCharWidth cannot be used on TrueType fonts. To retrieve character widths for TrueType fonts, use [GetCharABCWidths](#).

The range is inclusive; that is, the returned widths include the widths of the characters specified by the *iFirstChar* and *iLastChar* parameters.

If a character does not exist in the current font, it is assigned the width of the default character.

NOTE

The wingdi.h header defines GetCharWidth as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharABCWidthsFloat](#)

[GetCharWidth32](#)

[GetCharWidthFloat](#)

GetCharWidthFloatA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharWidthFloat** function retrieves the fractional widths of consecutive characters in a specified range from the current font.

Syntax

```
BOOL GetCharWidthFloatA(
    [in]  HDC      hdc,
    [in]  UINT     iFirst,
    [in]  UINT     iLast,
    [out] PFLOAT   lpBuffer
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `iFirst`

The code point of the first character in the group of consecutive characters.

[in] `iLast`

The code point of the last character in the group of consecutive characters.

[out] `lpBuffer`

A pointer to a buffer that receives the character widths, in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The returned widths are in the 32-bit IEEE floating-point format. (The widths are measured along the base line of the characters.)

If the *iFirstChar* parameter specifies the letter a and the *iLastChar* parameter specifies the letter z, **GetCharWidthFloat** retrieves the widths of all lowercase characters.

If a character does not exist in the current font, it is assigned the width of the default character.

NOTE

The wingdi.h header defines GetCharWidthFloat as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharABCWidthsFloat](#)

[GetCharWidth32](#)

GetCharWidthFloatW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharWidthFloat** function retrieves the fractional widths of consecutive characters in a specified range from the current font.

Syntax

```
BOOL GetCharWidthFloatW(
    [in]    HDC      hdc,
    [in]    UINT     iFirst,
    [in]    UINT     iLast,
    [out]   PFLOAT   lpBuffer
);
```

Parameters

[in] hdc

A handle to the device context.

[in] iFirst

The code point of the first character in the group of consecutive characters.

[in] iLast

The code point of the last character in the group of consecutive characters.

[out] lpBuffer

A pointer to a buffer that receives the character widths, in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The returned widths are in the 32-bit IEEE floating-point format. (The widths are measured along the base line of the characters.)

If the *iFirstChar* parameter specifies the letter a and the *iLastChar* parameter specifies the letter z, **GetCharWidthFloat** retrieves the widths of all lowercase characters.

If a character does not exist in the current font, it is assigned the width of the default character.

NOTE

The wingdi.h header defines GetCharWidthFloat as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharABCWidthsFloat](#)

[GetCharWidth32](#)

GetCharWidthI function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharWidthI** function retrieves the widths, in logical coordinates, of consecutive glyph indices in a specified range from the current font.

Syntax

```
BOOL GetCharWidthI(
    [in]  HDC      hdc,
    [in]  UINT     giFirst,
    [in]  UINT     cgi,
    [in]  LPWORD   pgi,
    [out] LPINT    piWidths
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `giFirst`

The first glyph index in the group of consecutive glyph indices.

[in] `cgi`

The number of glyph indices.

[in] `pgi`

A pointer to an array of glyph indices. If this parameter is not **NULL**, it is used instead of the *giFirst* parameter.

[out] `piWidths`

A pointer to a buffer that receives the widths, in logical coordinates.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **GetCharWidthI** function processes a consecutive glyph indices if the *pgi* parameter is **NULL** with the *giFirst* parameter indicating the first glyph index to process and the *cgi* parameter indicating how many glyph indices to process. Otherwise the **GetCharWidthI** function processes the array of glyph indices pointed to by the *pgi* parameter with the *cgi* parameter indicating how many glyph indices to process.

If a character does not exist in the current font, it is assigned the width of the default character.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharABCWidthsFloat](#)

[GetCharWidth32](#)

[GetCharWidthFloat](#)

GetCharWidthW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCharWidth** function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should call the [GetCharWidth32](#) function, which provides more accurate results.

Syntax

```
BOOL GetCharWidthW(
    [in]  HDC    hdc,
    [in]  UINT   iFirst,
    [in]  UINT   iLast,
    [out] LPINT  lpBuffer
);
```

Parameters

[in] hdc

A handle to the device context.

[in] iFirst

The first character in the group of consecutive characters.

[in] iLast

The last character in the group of consecutive characters, which must not precede the specified first character.

[out] lpBuffer

A pointer to a buffer that receives the character widths, in logical coordinates.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

GetCharWidth cannot be used on TrueType fonts. To retrieve character widths for TrueType fonts, use [GetCharABCWidths](#).

The range is inclusive; that is, the returned widths include the widths of the characters specified by the *iFirstChar* and *iLastChar* parameters.

If a character does not exist in the current font, it is assigned the width of the default character.

NOTE

The wingdi.h header defines GetCharWidth as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharABCWidthsFloat](#)

[GetCharWidth32](#)

[GetCharWidthFloat](#)

GetClipBox function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetClipBox** function retrieves the dimensions of the tightest bounding rectangle that can be drawn around the current visible area on the device. The visible area is defined by the current clipping region or clip path, as well as any overlapping windows.

Syntax

```
int GetClipBox(
    [in]  HDC      hdc,
    [out] LPRECT  lprect
);
```

Parameters

[in] `hdc`

A handle to the device context.

[out] `lprect`

A pointer to a [RECT](#) structure that is to receive the rectangle dimensions, in logical units.

Return value

If the function succeeds, the return value specifies the clipping box's complexity and can be one of the following values.

RETURN CODE	DESCRIPTION
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred.

GetClipBox returns logical coordinates based on the given device context.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Clipping Functions](#)

[Clipping Overview](#)

[RECT](#)

GetClipRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetClipRgn** function retrieves a handle identifying the current application-defined clipping region for the specified device context.

Syntax

```
int GetClipRgn(  
    [in] HDC hdc,  
    [in] HRGN hrgn  
) ;
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `hrgn`

A handle to an existing region before the function is called. After the function returns, this parameter is a handle to a copy of the current clipping region.

Return value

If the function succeeds and there is no clipping region for the given device context, the return value is zero. If the function succeeds and there is a clipping region for the given device context, the return value is 1. If an error occurs, the return value is -1.

Remarks

An application-defined clipping region is a clipping region identified by the [SelectClipRgn](#) function. It is not a clipping region created when the application calls the [BeginPaint](#) function.

If the function succeeds, the `hrgn` parameter is a handle to a copy of the current clipping region. Subsequent changes to this copy will not affect the current clipping region.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPaint](#)

[Clipping Functions](#)

[Clipping Overview](#)

[SelectClipRgn](#)

GetColorAdjustment function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetColorAdjustment** function retrieves the color adjustment values for the specified device context (DC).

Syntax

```
BOOL GetColorAdjustment(
    [in]    HDC           hdc,
    [out]   LP COLORADJUSTMENT lPCA
);
```

Parameters

[in] hdc

A handle to the device context.

[out] lPCA

A pointer to a [COLORADJUSTMENT](#) structure that receives the color adjustment values.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORADJUSTMENT](#)

[Color Functions](#)

[Colors Overview](#)

`SetColorAdjustment`

GetCurrentObject function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCurrentObject** function retrieves a handle to an object of the specified type that has been selected into the specified device context (DC).

Syntax

```
HGDIOBJ GetCurrentObject(  
    [in] HDC hdc,  
    [in] UINT type  
)
```

Parameters

[in] hdc

A handle to the DC.

[in] type

The object type to be queried. This parameter can be one of the following values.

VALUE	MEANING
OBJ_BITMAP	Returns the current selected bitmap.
OBJ_BRUSH	Returns the current selected brush.
OBJ_COLORSPACE	Returns the current color space.
OBJ_FONT	Returns the current selected font.
OBJ_PAL	Returns the current selected palette.
OBJ_PEN	Returns the current selected pen.

Return value

If the function succeeds, the return value is a handle to the specified object.

If the function fails, the return value is **NULL**.

Remarks

An application can use the [GetCurrentObject](#) and [GetObject](#) functions to retrieve descriptions of the graphic objects currently selected into the specified DC.

Examples

For an example, see [Retrieving Graphic-Object Attributes and Selecting New Graphic Objects](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateColorSpace](#)

[DeleteObject](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetObject](#)

[SelectObject](#)

GetCurrentPositionEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetCurrentPositionEx** function retrieves the current position in logical coordinates.

Syntax

```
BOOL GetCurrentPositionEx(
    [in]    HDC      hdc,
    [out]   LPPOINT lppt
);
```

Parameters

[in] hdc

A handle to the device context.

[out] lppt

A pointer to a [POINT](#) structure that receives the logical coordinates of the current position.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[MoveToEx](#)

POINT

GetDCBrushColor function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetDCBrushColor** function retrieves the current brush color for the specified device context (DC).

Syntax

```
COLORREF GetDCBrushColor(  
    [in] HDC hdc  
>);
```

Parameters

[in] hdc

A handle to the DC whose brush color is to be returned.

Return value

If the function succeeds, the return value is the [COLORREF](#) value for the current DC brush color.

If the function fails, the return value is [CLR_INVALID](#).

Remarks

For information on setting the brush color, see [SetDCBrushColor](#).

ICM: Color management is performed if ICM is enabled.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORREF](#)

[Device Context Functions](#)

Device Contexts Overview

[SetDCBrushColor](#)

GetDCOrgEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetDCOrgEx** function retrieves the final translation origin for a specified device context (DC). The final translation origin specifies an offset that the system uses to translate device coordinates into client coordinates (for coordinates in an application's window).

Syntax

```
BOOL GetDCOrgEx(
    [in]  HDC      hdc,
    [out] LPPOINT lppt
);
```

Parameters

[in] `hdc`

A handle to the DC whose final translation origin is to be retrieved.

[out] `lppt`

A pointer to a [POINT](#) structure that receives the final translation origin, in device coordinates.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The final translation origin is relative to the physical origin of the screen.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateIC](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[POINT](#)

GetDCPenColor function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetDCPenColor** function retrieves the current pen color for the specified device context (DC).

Syntax

```
COLORREF GetDCPenColor(  
    [in] HDC hdc  
>);
```

Parameters

[in] hdc

A handle to the DC whose brush color is to be returned.

Return value

If the function succeeds, the return value is a [COLORREF](#) value for the current DC pen color.

If the function fails, the return value is CLR_INVALID.

Remarks

For information on setting the pen color, see [SetDCPenColor](#).

ICM: Color management is performed if ICM is enabled.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORREF](#)

[Device Context Functions](#)

Device Contexts Overview

[SetDCPenColor](#)

GetDeviceCaps function (wingdi.h)

4/21/2022 • 8 minutes to read • [Edit Online](#)

The **GetDeviceCaps** function retrieves device-specific information for the specified device.

Syntax

```
int GetDeviceCaps(
    [in] HDC hdc,
    [in] int index
);
```

Parameters

[in] hdc

A handle to the DC.

[in] index

The item to be returned. This parameter can be one of the following values.

INDEX	MEANING														
DRIVERVERSION	The device driver version.														
TECHNOLOGY	Device technology. It can be any one of the following values. <table border="1"><tbody><tr><td>DT_PLOTTER</td><td>Vector plotter</td></tr><tr><td>DT_RASDISPLAY</td><td>Raster display</td></tr><tr><td>DT_RASPRINTER</td><td>Raster printer</td></tr><tr><td>DT_RASCAMERA</td><td>Raster camera</td></tr><tr><td>DT_CHARSTREAM</td><td>Character stream</td></tr><tr><td>DT_METAFILE</td><td>Metafile</td></tr><tr><td>DT_DISPFILE</td><td>Display file</td></tr></tbody></table>	DT_PLOTTER	Vector plotter	DT_RASDISPLAY	Raster display	DT_RASPRINTER	Raster printer	DT_RASCAMERA	Raster camera	DT_CHARSTREAM	Character stream	DT_METAFILE	Metafile	DT_DISPFILE	Display file
DT_PLOTTER	Vector plotter														
DT_RASDISPLAY	Raster display														
DT_RASPRINTER	Raster printer														
DT_RASCAMERA	Raster camera														
DT_CHARSTREAM	Character stream														
DT_METAFILE	Metafile														
DT_DISPFILE	Display file														
HORZSIZE	If the <i>hdc</i> parameter is a handle to the DC of an enhanced metafile, the device technology is that of the referenced device as specified to the CreateEnhMetaFile function. To determine whether it is an enhanced metafile DC, use the GetObjectType function. Width, in millimeters, of the physical screen.														

VERTSIZE	Height, in millimeters, of the physical screen.
HORZRES	Width, in pixels, of the screen; or for printers, the width, in pixels, of the printable area of the page.
VERTRES	Height, in raster lines, of the screen; or for printers, the height, in pixels, of the printable area of the page.
LOGPIXELSX	Number of pixels per logical inch along the screen width. In a system with multiple display monitors, this value is the same for all monitors.
LOGPIXELSY	Number of pixels per logical inch along the screen height. In a system with multiple display monitors, this value is the same for all monitors.
BITSPIXEL	Number of adjacent color bits for each pixel.
PLANES	Number of color planes.
NUMBRUSHES	Number of device-specific brushes.
NUMPENS	Number of device-specific pens.
NUMFONTS	Number of device-specific fonts.
NUMCOLORS	Number of entries in the device's color table, if the device has a color depth of no more than 8 bits per pixel. For devices with greater color depths, 1 is returned.
ASPECTX	Relative width of a device pixel used for line drawing.
ASPECTY	Relative height of a device pixel used for line drawing.
ASPECTXY	Diagonal width of the device pixel used for line drawing.
PDEVICESIZE	Reserved.
CLIPCAPS	Flag that indicates the clipping capabilities of the device. If the device can clip to a rectangle, it is 1. Otherwise, it is 0.

SIZEPALETTE	Number of entries in the system palette. This index is valid only if the device driver sets the RC_PALETTE bit in the RASTERCAPS index and is available only if the driver is compatible with 16-bit Windows.
NUMRESERVED	Number of reserved entries in the system palette. This index is valid only if the device driver sets the RC_PALETTE bit in the RASTERCAPS index and is available only if the driver is compatible with 16-bit Windows.
COLORRES	Actual color resolution of the device, in bits per pixel. This index is valid only if the device driver sets the RC_PALETTE bit in the RASTERCAPS index and is available only if the driver is compatible with 16-bit Windows.
PHYSICALWIDTH	For printing devices: the width of the physical page, in device units. For example, a printer set to print at 600 dpi on 8.5-x11-inch paper has a physical width value of 5100 device units. Note that the physical page is almost always greater than the printable area of the page, and never smaller.
PHYSICALHEIGHT	For printing devices: the height of the physical page, in device units. For example, a printer set to print at 600 dpi on 8.5-by-11-inch paper has a physical height value of 6600 device units. Note that the physical page is almost always greater than the printable area of the page, and never smaller.
PHYSICALOFFSETX	For printing devices: the distance from the left edge of the physical page to the left edge of the printable area, in device units. For example, a printer set to print at 600 dpi on 8.5-by-11-inch paper, that cannot print on the leftmost 0.25-inch of paper, has a horizontal physical offset of 150 device units.
PHYSICALOFFSETY	For printing devices: the distance from the top edge of the physical page to the top edge of the printable area, in device units. For example, a printer set to print at 600 dpi on 8.5-by-11-inch paper, that cannot print on the topmost 0.5-inch of paper, has a vertical physical offset of 300 device units.
VREFRESH	For display devices: the current vertical refresh rate of the device, in cycles per second (Hz). A vertical refresh rate value of 0 or 1 represents the display hardware's default refresh rate. This default rate is typically set by switches on a display card or computer motherboard, or by a configuration program that does not use display functions such as ChangeDisplaySettings .
SCALINGFACTORX	Scaling factor for the x-axis of the printer.
SCALINGFACTORY	Scaling factor for the y-axis of the printer.

BLTALIGNMENT	<p>Preferred horizontal drawing alignment, expressed as a multiple of pixels. For best drawing performance, windows should be horizontally aligned to a multiple of this value. A value of zero indicates that the device is accelerated, and any alignment may be used.</p>												
SHADEBLENDCAPS	<p>Value that indicates the shading and blending capabilities of the device. See Remarks for further comments.</p> <table border="1" data-bbox="809 393 1404 1156"> <tr> <td data-bbox="809 393 1110 617">SB_CONST_ALPHA</td><td data-bbox="1110 393 1404 617"> <p>Handles the SourceConstantAlpha member of the BLENDFUNCTION structure, which is referenced by the blendFunction parameter of the AlphaBlend function.</p> </td></tr> <tr> <td data-bbox="809 617 1110 729">SB_GRAD_RECT</td><td data-bbox="1110 617 1404 729"> <p>Capable of doing GradientFill rectangles.</p> </td></tr> <tr> <td data-bbox="809 729 1110 842">SB_GRAD_TRI</td><td data-bbox="1110 729 1404 842"> <p>Capable of doing GradientFill triangles.</p> </td></tr> <tr> <td data-bbox="809 842 1110 932">SB_NONE</td><td data-bbox="1110 842 1404 932"> <p>Device does not support any of these capabilities.</p> </td></tr> <tr> <td data-bbox="809 932 1110 1044">SB_PIXEL_ALPHA</td><td data-bbox="1110 932 1404 1044"> <p>Capable of handling per-pixel alpha in AlphaBlend.</p> </td></tr> <tr> <td data-bbox="809 1044 1110 1156">SB_PREMULT_ALPHA</td><td data-bbox="1110 1044 1404 1156"> <p>Capable of handling premultiplied alpha in AlphaBlend.</p> </td></tr> </table>	SB_CONST_ALPHA	<p>Handles the SourceConstantAlpha member of the BLENDFUNCTION structure, which is referenced by the blendFunction parameter of the AlphaBlend function.</p>	SB_GRAD_RECT	<p>Capable of doing GradientFill rectangles.</p>	SB_GRAD_TRI	<p>Capable of doing GradientFill triangles.</p>	SB_NONE	<p>Device does not support any of these capabilities.</p>	SB_PIXEL_ALPHA	<p>Capable of handling per-pixel alpha in AlphaBlend.</p>	SB_PREMULT_ALPHA	<p>Capable of handling premultiplied alpha in AlphaBlend.</p>
SB_CONST_ALPHA	<p>Handles the SourceConstantAlpha member of the BLENDFUNCTION structure, which is referenced by the blendFunction parameter of the AlphaBlend function.</p>												
SB_GRAD_RECT	<p>Capable of doing GradientFill rectangles.</p>												
SB_GRAD_TRI	<p>Capable of doing GradientFill triangles.</p>												
SB_NONE	<p>Device does not support any of these capabilities.</p>												
SB_PIXEL_ALPHA	<p>Capable of handling per-pixel alpha in AlphaBlend.</p>												
SB_PREMULT_ALPHA	<p>Capable of handling premultiplied alpha in AlphaBlend.</p>												

RASTERCAPS

Value that indicates the raster capabilities of the device, as shown in the following table.

RC_BANDING	Requires banding support.
RC_BITBLT	Capable of transferring bitmaps.
RC_BITMAP64	Capable of supporting bitmaps larger than 64 KB.
RC_DI_BITMAP	Capable of supporting the SetDIBits and GetDIBits functions.
RC_DIBTODEV	Capable of supporting the SetDIBitsToDevice function.
RC_FLOODFILL	Capable of performing flood fills.
RC_PALETTE	Specifies a palette-based device.
RC_SCALING	Capable of scaling.
RC_STRETCHBLT	Capable of performing the StretchBlt function.
RC_STRETCHDIB	Capable of performing the StretchDIBits function.

CURVECAPS	<p>Value that indicates the curve capabilities of the device, as shown in the following table.</p> <table border="1"> <tbody> <tr> <td data-bbox="810 181 1112 271">CC_NONE</td><td data-bbox="1112 181 1414 271">Device does not support curves.</td></tr> <tr> <td data-bbox="810 271 1112 361">CC_CHORD</td><td data-bbox="1112 271 1414 361">Device can draw chord arcs.</td></tr> <tr> <td data-bbox="810 361 1112 440">CC_CIRCLES</td><td data-bbox="1112 361 1414 440">Device can draw circles.</td></tr> <tr> <td data-bbox="810 440 1112 518">CC_ELLIPSES</td><td data-bbox="1112 440 1414 518">Device can draw ellipses.</td></tr> <tr> <td data-bbox="810 518 1112 597">CC_INTERIORS</td><td data-bbox="1112 518 1414 597">Device can draw interiors.</td></tr> <tr> <td data-bbox="810 597 1112 676">CC_PIE</td><td data-bbox="1112 597 1414 676">Device can draw pie wedges.</td></tr> <tr> <td data-bbox="810 676 1112 754">CC_ROUNDRECT</td><td data-bbox="1112 676 1414 754">Device can draw rounded rectangles.</td></tr> <tr> <td data-bbox="810 754 1112 833">CC_STYLED</td><td data-bbox="1112 754 1414 833">Device can draw styled borders.</td></tr> <tr> <td data-bbox="810 833 1112 911">CC_WIDE</td><td data-bbox="1112 833 1414 911">Device can draw wide borders.</td></tr> <tr> <td data-bbox="810 911 1112 1057">CC_WIDESTYLED</td><td data-bbox="1112 911 1414 1057">Device can draw borders that are wide and styled.</td></tr> </tbody> </table>	CC_NONE	Device does not support curves.	CC_CHORD	Device can draw chord arcs.	CC_CIRCLES	Device can draw circles.	CC_ELLIPSES	Device can draw ellipses.	CC_INTERIORS	Device can draw interiors.	CC_PIE	Device can draw pie wedges.	CC_ROUNDRECT	Device can draw rounded rectangles.	CC_STYLED	Device can draw styled borders.	CC_WIDE	Device can draw wide borders.	CC_WIDESTYLED	Device can draw borders that are wide and styled.
CC_NONE	Device does not support curves.																				
CC_CHORD	Device can draw chord arcs.																				
CC_CIRCLES	Device can draw circles.																				
CC_ELLIPSES	Device can draw ellipses.																				
CC_INTERIORS	Device can draw interiors.																				
CC_PIE	Device can draw pie wedges.																				
CC_ROUNDRECT	Device can draw rounded rectangles.																				
CC_STYLED	Device can draw styled borders.																				
CC_WIDE	Device can draw wide borders.																				
CC_WIDESTYLED	Device can draw borders that are wide and styled.																				
LINECAPS	<p>Value that indicates the line capabilities of the device, as shown in the following table:</p> <table border="1"> <tbody> <tr> <td data-bbox="810 1248 1112 1338">LC_NONE</td><td data-bbox="1112 1248 1414 1338">Device does not support lines.</td></tr> <tr> <td data-bbox="810 1338 1112 1417">LC_INTERIORS</td><td data-bbox="1112 1338 1414 1417">Device can draw interiors.</td></tr> <tr> <td data-bbox="810 1417 1112 1495">LC_MARKER</td><td data-bbox="1112 1417 1414 1495">Device can draw a marker.</td></tr> <tr> <td data-bbox="810 1495 1112 1551">LC_POLYLINE</td><td data-bbox="1112 1495 1414 1551">Device can draw a polyline.</td></tr> <tr> <td data-bbox="810 1551 1112 1630">LC_POLYMARKER</td><td data-bbox="1112 1551 1414 1630">Device can draw multiple markers.</td></tr> <tr> <td data-bbox="810 1630 1112 1709">LC_STYLED</td><td data-bbox="1112 1630 1414 1709">Device can draw styled lines.</td></tr> <tr> <td data-bbox="810 1709 1112 1787">LC_WIDE</td><td data-bbox="1112 1709 1414 1787">Device can draw wide lines.</td></tr> <tr> <td data-bbox="810 1787 1112 1911">LC_WIDESTYLED</td><td data-bbox="1112 1787 1414 1911">Device can draw lines that are wide and styled.</td></tr> </tbody> </table>	LC_NONE	Device does not support lines.	LC_INTERIORS	Device can draw interiors.	LC_MARKER	Device can draw a marker.	LC_POLYLINE	Device can draw a polyline.	LC_POLYMARKER	Device can draw multiple markers.	LC_STYLED	Device can draw styled lines.	LC_WIDE	Device can draw wide lines.	LC_WIDESTYLED	Device can draw lines that are wide and styled.				
LC_NONE	Device does not support lines.																				
LC_INTERIORS	Device can draw interiors.																				
LC_MARKER	Device can draw a marker.																				
LC_POLYLINE	Device can draw a polyline.																				
LC_POLYMARKER	Device can draw multiple markers.																				
LC_STYLED	Device can draw styled lines.																				
LC_WIDE	Device can draw wide lines.																				
LC_WIDESTYLED	Device can draw lines that are wide and styled.																				

POLYGONALCAPS

Value that indicates the polygon capabilities of the device, as shown in the following table.

PC_NONE	Device does not support polygons.
PC_INTERIORS	Device can draw interiors.
PC_POLYGON	Device can draw alternate-fill polygons.
PC_RECTANGLE	Device can draw rectangles.
PC_SCANLINE	Device can draw a single scanline.
PC_STYLED	Device can draw styled borders.
PC_WIDE	Device can draw wide borders.
PC_WIDESTYLED	Device can draw borders that are wide and styled.
PC_WINDPOLYGON	Device can draw winding-fill polygons.

TEXTCAPS

Value that indicates the text capabilities of the device, as shown in the following table.

TC_OP_CHARACTER	Device is capable of character output precision.
TC_OP_STROKE	Device is capable of stroke output precision.
TC_CP_STROKE	Device is capable of stroke clip precision.
TC_CR_90	Device is capable of 90-degree character rotation.
TC_CR_ANY	Device is capable of any character rotation.
TC_SF_X_YINDEP	Device can scale independently in the x- and y-directions.
TC_SA_DOUBLE	Device is capable of doubled character for scaling.
TC_SA_INTEGER	Device uses integer multiples only for character scaling.
TC_SA_CONTIN	Device uses any multiples for exact character scaling.
TC_EA_DOUBLE	Device can draw double-weight characters.
TC_IA_ABLE	Device can italicize.
TC_UA_ABLE	Device can underline.
TC_SO_ABLE	Device can draw strikeouts.
TC_RA_ABLE	Device can draw raster fonts.
TC_VA_ABLE	Device can draw vector fonts.
TC_RESERVED	Reserved; must be zero.
TC_SCROLLBLT	Device cannot scroll using a bit-block transfer. Note that this meaning may be the opposite of what you expect.

COLORMGMTCAPS	Value that indicates the color management capabilities of the device.	
	CM_CMYK_COLOR	Device can accept CMYK color space ICC color profile.
	CM_DEVICE_ICM	Device can perform ICM on either the device driver or the device itself.
	CM_GAMMA_RAMP	Device supports GetDeviceGammaRamp and SetDeviceGammaRamp
	CM_NONE	Device does not support ICM.

Return value

The return value specifies the value of the desired item.

When *nIndex* is BITSPIXEL and the device has 15bpp or 16bpp, the return value is 16.

Remarks

When *nIndex* is SHADEBLENDCAPS:

- For a printer, **GetDeviceCaps** returns whatever the printer reports.
- For a display device, all blending operations are available; besides SB_NONE, the only return values are SB_CONST_ALPHA and SB_PIXEL_ALPHA, which indicate whether these operations are accelerated.

On a multiple monitor system, if *hdc* is the desktop, **GetDeviceCaps** returns the capabilities of the primary monitor. If you want info for other monitors, you must use the [multi-monitor APIs](#) or [CreateDC](#) to get a HDC for the device context (DC) of a specific monitor.

Note Display1 is typically the primary monitor, but not always.

GetDeviceCaps provides the following six indexes in place of printer escapes.

INDEX	PRINTER ESCAPE REPLACED
PHYSICALWIDTH	GETPHYSICALSIZE
PHYSICALHEIGHT	GETPHYSICALSIZE
PHYSICALOFFSETX	GETPRINTINGOFFSET
PHYSICALOFFSETY	GETPHYSICALOFFSET
SCALINGFACTORX	GETSCALINGFACTOR
SCALINGFACTORY	GETSCALINGFACTOR

Note `GetDeviceCaps` reports info that the display driver provides. If the display driver declines to report any info, `GetDeviceCaps` calculates the info based on fixed calculations. If the display driver reports invalid info, `GetDeviceCaps` returns the invalid info. Also, if the display driver declines to report info, `GetDeviceCaps` might calculate incorrect info because it assumes either fixed DPI (96 DPI) or a fixed size (depending on the info that the display driver did and didn't provide). Unfortunately, a display driver that is implemented to the Windows Display Driver Model (WDDM) (introduced in Windows Vista) causes GDI to not get the info, so `GetDeviceCaps` must always calculate the info.

Examples

For an example, see [Preparing to Print](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateEnhMetaFile](#)

[CreateIC](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[DeviceCapabilities](#)

[GetDIBits](#)

[GetObjectType](#)

[SetDIBits](#)

[SetDIBitsToDevice](#)

[StretchBlt](#)

[StretchDIBits](#)

GetDIBColorTable function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetDIBColorTable** function retrieves RGB (red, green, blue) color values from a range of entries in the color table of the DIB section bitmap that is currently selected into a specified device context.

Syntax

```
UINT GetDIBColorTable(
    [in]  HDC      hdc,
    [in]  UINT     iStart,
    [in]  UINT     cEntries,
    [out] RGBQUAD *prgbq
);
```

Parameters

[in] `hdc`

A handle to a device context. A DIB section bitmap must be selected into this device context.

[in] `iStart`

A zero-based color table index that specifies the first color table entry to retrieve.

[in] `cEntries`

The number of color table entries to retrieve.

[out] `prgbq`

A pointer to a buffer that receives an array of **RGBQUAD** data structures containing color information from the DIB color table. The buffer must be large enough to contain as many **RGBQUAD** data structures as the value of *cEntries*.

Return value

If the function succeeds, the return value is the number of color table entries that the function retrieves.

If the function fails, the return value is zero.

Remarks

The **GetDIBColorTable** function should be called to retrieve the color table for DIB section bitmaps that use 1, 4, or 8 bpp. The **biBitCount** member of a bitmap associated **BITMAPINFOHEADER** structure specifies the number of bits-per-pixel. DIB section bitmaps with a **biBitCount** value greater than eight do not have a color table, but they do have associated color masks. Call the **GetObject** function to retrieve those color masks.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPINFOHEADER](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateDIBSection](#)

[DIBSECTION](#)

[GetObject](#)

[RGBQUAD](#)

[SetDIBColorTable](#)

GetDIBits function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **GetDIBits** function retrieves the bits of the specified compatible bitmap and copies them into a buffer as a DIB using the specified format.

Syntax

```
int GetDIBits(
    [in]      HDC      hdc,
    [in]      HBITMAP  hbm,
    [in]      UINT     start,
    [in]      UINT     cLines,
    [out]     LPVOID   lpvBits,
    [in, out] LPBITMAPINFO lpbmi,
    [in]      UINT     usage
);
```

Parameters

[in] hdc

A handle to the device context.

[in] hbm

A handle to the bitmap. This must be a compatible bitmap (DDB).

[in] start

The first scan line to retrieve.

[in] cLines

The number of scan lines to retrieve.

[out] lpvBits

A pointer to a buffer to receive the bitmap data. If this parameter is **NULL**, the function passes the dimensions and format of the bitmap to the **BITMAPINFO** structure pointed to by the *lpbmi* parameter.

[in, out] lpbmi

A pointer to a **BITMAPINFO** structure that specifies the desired format for the DIB data.

[in] usage

The format of the **bmiColors** member of the **BITMAPINFO** structure. It must be one of the following values.

VALUE	MEANING
DIB_PAL_COLORS	The color table should consist of an array of 16-bit indexes into the current logical palette.

DIB_RGB_COLORS	The color table should consist of literal red, green, blue (RGB) values.
----------------	--

Return value

If the */pvBits* parameter is non-NULL and the function succeeds, the return value is the number of scan lines copied from the bitmap.

If the */pvBits* parameter is NULL and **GetDIBits** successfully fills the **BITMAPINFO** structure, the return value is nonzero.

If the function fails, the return value is zero.

This function can return the following value.

RETURN CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	One or more of the input parameters is invalid.

Remarks

If the requested format for the DIB matches its internal format, the RGB values for the bitmap are copied. If the requested format doesn't match the internal format, a color table is synthesized. The following table describes the color table synthesized for each format.

VALUE	MEANING
1_BPP	The color table consists of a black and a white entry.
4_BPP	The color table consists of a mix of colors identical to the standard VGA palette.
8_BPP	The color table consists of a general mix of 256 colors defined by GDI. (Included in these 256 colors are the 20 colors found in the default logical palette.)
24_BPP	No color table is returned.

If the */pvBits* parameter is a valid pointer, the first six members of the **BITMAPINFOHEADER** structure must be initialized to specify the size and format of the DIB. The scan lines must be aligned on a **DWORD** except for RLE compressed bitmaps.

A bottom-up DIB is specified by setting the height to a positive number, while a top-down DIB is specified by setting the height to a negative number. The bitmap color table will be appended to the **BITMAPINFO** structure.

If */pvBits* is NULL, **GetDIBits** examines the first member of the first structure pointed to by */pbi*. This member must specify the size, in bytes, of a **BITMAPCOREHEADER** or a **BITMAPINFOHEADER** structure. The function uses the specified size to determine how the remaining members should be initialized.

If */pvBits* is NULL and the bit count member of **BITMAPINFO** is initialized to zero, **GetDIBits** fills in a **BITMAPINFOHEADER** structure or **BITMAPCOREHEADER** without the color table. This technique can be used to query bitmap attributes.

The bitmap identified by the *hbmp* parameter must not be selected into a device context when the application calls this function.

The origin for a bottom-up DIB is the lower-left corner of the bitmap; the origin for a top-down DIB is the upper-left corner.

Examples

For an example, see [Capturing an Image](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPCOREHEADER](#)

[BITMAPINFO](#)

[BITMAPINFOHEADER](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[SetDIBits](#)

GetEnhMetaFileA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetEnhMetaFile** function creates a handle that identifies the enhanced-format metafile stored in the specified file.

Syntax

```
HENHMETAFILE GetEnhMetaFileA(
    [in] LPCSTR lpName
);
```

Parameters

[in] *lpName*

A pointer to a null-terminated string that specifies the name of an enhanced metafile.

Return value

If the function succeeds, the return value is a handle to the enhanced metafile.

If the function fails, the return value is **NULL**.

Remarks

When the application no longer needs an enhanced-metafile handle, it should delete the handle by calling the [DeleteEnhMetaFile](#) function.

A Windows-format metafile must be converted to the enhanced format before it can be processed by the **GetEnhMetaFile** function. To convert the file, use the [SetWinMetaFileBits](#) function.

Where text arguments must use Unicode characters, use this function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

Examples

For an example, see [Opening an Enhanced Metafile and Displaying Its Contents](#).

NOTE

The wingdi.h header defines **GetEnhMetaFile** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteEnhMetaFile](#)

[GetEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SetWinMetaFileBits](#)

GetEnhMetaFileBits function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetEnhMetaFileBits** function retrieves the contents of the specified enhanced-format metafile and copies them into a buffer.

Syntax

```
UINT GetEnhMetaFileBits(
    [in] HENHMETAFILE hEMF,
    [in] UINT         nSize,
    [out] LPBYTE      lpData
);
```

Parameters

[in] **hEMF**

A handle to the enhanced metafile.

[in] **nSize**

The size, in bytes, of the buffer to receive the data.

[out] **lpData**

A pointer to a buffer that receives the metafile data. The buffer must be sufficiently large to contain the data. If *lpBuffer* is **NULL**, the function returns the size necessary to hold the data.

Return value

If the function succeeds and the buffer pointer is **NULL**, the return value is the size of the enhanced metafile, in bytes.

If the function succeeds and the buffer pointer is a valid pointer, the return value is the number of bytes copied to the buffer.

If the function fails, the return value is zero.

Remarks

After the enhanced-metafile bits are retrieved, they can be used to create a memory-based metafile by calling the [SetEnhMetaFileBits](#) function.

The **GetEnhMetaFileBits** function does not invalidate the enhanced-metafile handle. The application must call the [DeleteEnhMetaFile](#) function to delete the handle when it is no longer needed.

The metafile contents retrieved by this function are in the enhanced format. To retrieve the metafile contents in the Windows format, use the [GetWinMetaFileBits](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteEnhMetaFile](#)

[GetWinMetaFileBits](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SetEnhMetaFileBits](#)

GetEnhMetaFileDescriptionA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetEnhMetaFileDescription** function retrieves an optional text description from an enhanced-format metafile and copies the string to the specified buffer.

Syntax

```
UINT GetEnhMetaFileDescriptionA(
    [in] HENHMETAFILE hemf,
    [in] UINT      cchBuffer,
    [out] LPSTR     lpDescription
);
```

Parameters

[in] hemf

A handle to the enhanced metafile.

[in] cchBuffer

The size, in characters, of the buffer to receive the data. Only this many characters will be copied.

[out] lpDescription

A pointer to a buffer that receives the optional text description.

Return value

If the optional text description exists and the buffer pointer is **NULL**, the return value is the length of the text string, in characters.

If the optional text description exists and the buffer pointer is a valid pointer, the return value is the number of characters copied into the buffer.

If the optional text description does not exist, the return value is zero.

If the function fails, the return value is **GDI_ERROR**.

Remarks

The optional text description contains two strings, the first identifying the application that created the enhanced metafile and the second identifying the picture contained in the metafile. The strings are separated by a null character and terminated with two null characters, for example, "XYZ Graphics Editor\0Bald Eagle\0\0" where \0 represents the null character.

Where text arguments must use Unicode characters, use this function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

NOTE

The wingdi.h header defines GetEnhMetaFileDescription as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

GetEnhMetaFileDescriptionW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetEnhMetaFileDescription** function retrieves an optional text description from an enhanced-format metafile and copies the string to the specified buffer.

Syntax

```
UINT GetEnhMetaFileDescriptionW(
    [in] HENHMETAFILE hemf,
    [in] UINT      cchBuffer,
    [out] LPWSTR    lpDescription
);
```

Parameters

[in] hemf

A handle to the enhanced metafile.

[in] cchBuffer

The size, in characters, of the buffer to receive the data. Only this many characters will be copied.

[out] lpDescription

A pointer to a buffer that receives the optional text description.

Return value

If the optional text description exists and the buffer pointer is **NULL**, the return value is the length of the text string, in characters.

If the optional text description exists and the buffer pointer is a valid pointer, the return value is the number of characters copied into the buffer.

If the optional text description does not exist, the return value is zero.

If the function fails, the return value is **GDI_ERROR**.

Remarks

The optional text description contains two strings, the first identifying the application that created the enhanced metafile and the second identifying the picture contained in the metafile. The strings are separated by a null character and terminated with two null characters, for example, "XYZ Graphics Editor\0Bald Eagle\0\0" where \0 represents the null character.

Where text arguments must use Unicode characters, use this function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

NOTE

The wingdi.h header defines GetEnhMetaFileDescription as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

GetEnhMetaFileHeader function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetEnhMetaFileHeader** function retrieves the record containing the header for the specified enhanced-format metafile.

Syntax

```
UINT GetEnhMetaFileHeader(
    [in] HENHMETAFILE    hemf,
    [in] UINT            nSize,
    [out] LPENHMETAHEADER lpEnhMetaHeader
);
```

Parameters

[in] hemf

A handle to the enhanced metafile for which the header is to be retrieved.

[in] nSize

The size, in bytes, of the buffer to receive the data. Only this many bytes will be copied.

[out] lpEnhMetaHeader

A pointer to an **ENHMETAHEADER** structure that receives the header record. If this parameter is **NULL**, the function returns the size of the header record.

Return value

If the function succeeds and the structure pointer is **NULL**, the return value is the size of the record that contains the header; if the structure pointer is a valid pointer, the return value is the number of bytes copied. Otherwise, it is zero.

Remarks

An enhanced-metafile header contains such information as the metafile's size, in bytes; the dimensions of the picture stored in the metafile; the number of records stored in the metafile; the offset to the optional text description; the size of the optional palette, and the resolution of the device on which the picture was created.

The record that contains the enhanced-metafile header is always the first record in the metafile.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ENHMETAHEADER](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayEnhMetaFile](#)

GetEnhMetaFilePaletteEntries function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetEnhMetaFilePaletteEntries** function retrieves optional palette entries from the specified enhanced metafile.

Syntax

```
UINT GetEnhMetaFilePaletteEntries(
    [in] HENHMETAFILE    hemf,
    [in] UINT            nNumEntries,
    [out] LPPALETTEENTRY lpPaletteEntries
);
```

Parameters

[in] `hemf`

A handle to the enhanced metafile.

[in] `nNumEntries`

The number of entries to be retrieved from the optional palette.

[out] `lpPaletteEntries`

A pointer to an array of **PALETTEENTRY** structures that receives the palette colors. The array must contain at least as many structures as there are entries specified by the *cEntries* parameter.

Return value

If the array pointer is **NULL** and the enhanced metafile contains an optional palette, the return value is the number of entries in the enhanced metafile's palette; if the array pointer is a valid pointer and the enhanced metafile contains an optional palette, the return value is the number of entries copied; if the metafile does not contain an optional palette, the return value is zero. Otherwise, the return value is **GDI_ERROR**.

Remarks

An application can store an optional palette in an enhanced metafile by calling the [CreatePalette](#) and [SetPaletteEntries](#) functions before creating the picture and storing it in the metafile. By doing this, the application can achieve consistent colors when the picture is displayed on a variety of devices.

An application that displays a picture stored in an enhanced metafile can call the **GetEnhMetaFilePaletteEntries** function to determine whether the optional palette exists. If it does, the application can call the **GetEnhMetaFilePaletteEntries** function a second time to retrieve the palette entries and then create a logical palette (by using the [CreatePalette](#) function), select it into its device context (by using the [SelectPalette](#) function), and then realize it (by using the [RealizePalette](#) function). After the logical palette has been realized, calling the [PlayEnhMetaFile](#) function displays the picture using its original colors.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePalette](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PALETTEENTRY](#)

[PlayEnhMetaFile](#)

[RealizePalette](#)

[SelectPalette](#)

GetEnhMetaFileW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetEnhMetaFile** function creates a handle that identifies the enhanced-format metafile stored in the specified file.

Syntax

```
HENHMETAFILE GetEnhMetaFileW(
    [in] LPCWSTR lpName
);
```

Parameters

[in] *lpName*

A pointer to a null-terminated string that specifies the name of an enhanced metafile.

Return value

If the function succeeds, the return value is a handle to the enhanced metafile.

If the function fails, the return value is **NULL**.

Remarks

When the application no longer needs an enhanced-metafile handle, it should delete the handle by calling the [DeleteEnhMetaFile](#) function.

A Windows-format metafile must be converted to the enhanced format before it can be processed by the **GetEnhMetaFile** function. To convert the file, use the [SetWinMetaFileBits](#) function.

Where text arguments must use Unicode characters, use this function as a wide-character function. Where text arguments must use characters from the Windows character set, use this function as an ANSI function.

Examples

For an example, see [Opening an Enhanced Metafile and Displaying Its Contents](#).

NOTE

The wingdi.h header defines **GetEnhMetaFile** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteEnhMetaFile](#)

[GetEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SetWinMetaFileBits](#)

GetFontData function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetFontData** function retrieves font metric data for a TrueType font.

Syntax

```
DWORD GetFontData(
    [in]  HDC    hdc,
    [in]  DWORD  dwTable,
    [in]  DWORD  dwOffset,
    [out] PVOID  pvBuffer,
    [in]  DWORD  cjBuffer
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `dwTable`

The name of a font metric table from which the font data is to be retrieved. This parameter can identify one of the metric tables documented in the TrueType Font Files specification published by Microsoft Corporation. If this parameter is zero, the information is retrieved starting at the beginning of the file for TrueType font files or from the beginning of the data for the currently selected font for TrueType Collection files. To retrieve the data from the beginning of the file for TrueType Collection files specify 'ttcf' (0x66637474).

[in] `dwOffset`

The offset from the beginning of the font metric table to the location where the function should begin retrieving information. If this parameter is zero, the information is retrieved starting at the beginning of the table specified by the `dwTable` parameter. If this value is greater than or equal to the size of the table, an error occurs.

[out] `pvBuffer`

A pointer to a buffer that receives the font information. If this parameter is **NULL**, the function returns the size of the buffer required for the font data.

[in] `cjBuffer`

The length, in bytes, of the information to be retrieved. If this parameter is zero, **GetFontData** returns the size of the data specified in the `dwTable` parameter.

Return value

If the function succeeds, the return value is the number of bytes returned.

If the function fails, the return value is **GDI_ERROR**.

Remarks

This function is intended to be used to retrieve TrueType font information directly from the font file by font-manipulation applications. For information about embedding fonts see the [Font Embedding Reference](#).

An application can sometimes use the **GetFontData** function to save a TrueType font with a document. To do this, the application determines whether the font can be embedded by checking the **otmfsType** member of the [OUTLINETEXTMETRIC](#) structure. If bit 1 of **otmfsType** is set, embedding is not permitted for the font. If bit 1 is clear, the font can be embedded. If bit 2 is set, the embedding is read-only. If embedding is permitted, the application can retrieve the entire font file, specifying zero for the *dwTable*, *dwOffset*, and *cbData* parameters.

If an application attempts to use this function to retrieve information for a non-TrueType font, an error occurs.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextMetrics](#)

[OUTLINETEXTMETRIC](#)

GetFontLanguageInfo function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetFontLanguageInfo** function returns information about the currently selected font for the specified display context. Applications typically use this information and the [GetCharacterPlacement](#) function to prepare a character string for display.

Syntax

```
DWORD GetFontLanguageInfo(  
    [in] HDC hdc  
);
```

Parameters

[in] hdc

Handle to a display device context.

Return value

The return value identifies characteristics of the currently selected font. The function returns 0 if the font is "normalized" and can be treated as a simple Latin font; it returns GCP_ERROR if an error occurs. Otherwise, the function returns a combination of the following values.

VALUE	MEANING
GCP_DBCS	The character set is DBCS.
GCP_DIACRITIC	The font/language contains diacritic glyphs.
FLI_GLYPHS	The font contains extra glyphs not normally accessible using the code page. Use GetCharacterPlacement to access the glyphs. This value is for information only and is not intended to be passed to GetCharacterPlacement .
GCP_GLYPHSHAPE	The font/language contains multiple glyphs per code point or per code point combination (supports shaping and/or ligation), and the font contains advanced glyph tables to provide extra glyphs for the extra shapes. If this value is specified, the lpGlyphs array must be used with the GetCharacterPlacement function and the ETO_GLYPHINDEX value must be passed to the ExtTextOut function when the string is drawn.
GCP_KASHIDA	The font/ language permits Kashidas.
GCP_LIGATE	The font/language contains ligation glyphs which can be substituted for specific character combinations.

GCP_USEKERNING	The font contains a kerning table which can be used to provide better spacing between the characters and glyphs.
GCP_REORDER	The language requires reordering for display for example, Hebrew or Arabic.

The return value, when masked with FLI_MASK, can be passed directly to the [GetCharacterPlacement](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharacterPlacement](#)

GetFontUnicodeRanges function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetFontUnicodeRanges** function returns information about which Unicode characters are supported by a font. The information is returned as a [GLYPHSET](#) structure.

Syntax

```
DWORD GetFontUnicodeRanges(
    [in]    HDC      hdc,
    [out]   LPGLYPHSET lpgs
);
```

Parameters

[in] `hdc`

A handle to the device context.

[out] `lpgs`

A pointer to a [GLYPHSET](#) structure that receives the glyph set information. If this parameter is **NULL**, the function returns the size of the [GLYPHSET](#) structure required to store the information.

Return value

If the function succeeds, it returns number of bytes written to the [GLYPHSET](#) structure or, if the `/pgs` parameter is **NULL**, it returns the size of the [GLYPHSET](#) structure required to store the information.

If the function fails, it returns zero. No extended error information is available.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

Fonts and Text Overview

GLYPHSET

GetGlyphIndicesA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetGlyphIndices** function translates a string into an array of glyph indices. The function can be used to determine whether a glyph exists in a font.

Syntax

```
DWORD GetGlyphIndicesA(
    [in]  HDC      hdc,
    [in]  LPCSTR   lpstr,
    [in]  int       c,
    [out] LPWORD   pgi,
    [in]  DWORD    fl
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpstr`

A pointer to the string to be converted.

[in] `c`

The length of both the [length of the string](#) pointed to by `/pstr` and the size (in WORDs) of the buffer pointed to by `pgi`.

[out] `pgi`

This buffer must be of dimension `c`. On successful return, contains an array of glyph indices corresponding to the characters in the string.

[in] `fl`

Specifies how glyphs should be handled if they are not supported. This parameter can be the following value.

VALUE	MEANING
<code>GGI_MARK_NONEXISTING_GLYPHS</code>	Marks unsupported glyphs with the hexadecimal value <code>0xffff</code> .

Return value

If the function succeeds, it returns the number of bytes (for the ANSI function) or WORDs (for the Unicode function) converted.

If the function fails, the return value is `GDI_ERROR`.

Remarks

This function attempts to identify a single-glyph representation for each character in the string pointed to by *lpstr*. While this is useful for certain low-level purposes (such as manipulating font files), higher-level applications that wish to map a string to glyphs will typically wish to use the [Uniscribe](#) functions.

NOTE

The wingdi.h header defines GetGlyphIndices as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetFontUnicodeRanges](#)

GetGlyphIndicesW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetGlyphIndices** function translates a string into an array of glyph indices. The function can be used to determine whether a glyph exists in a font.

Syntax

```
DWORD GetGlyphIndicesW(
    [in]    HDC      hdc,
    [in]    LPCWSTR  lpstr,
    [in]    int       c,
    [out]   LPWORD   pgi,
    [in]    DWORD    fl
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpstr`

A pointer to the string to be converted.

[in] `c`

The length of both the [length of the string](#) pointed to by `/pstr` and the size (in WORDs) of the buffer pointed to by `pgi`.

[out] `pgi`

This buffer must be of dimension `c`. On successful return, contains an array of glyph indices corresponding to the characters in the string.

[in] `fl`

Specifies how glyphs should be handled if they are not supported. This parameter can be the following value.

VALUE	MEANING
<code>GGI_MARK_NONEXISTING_GLYPHS</code>	Marks unsupported glyphs with the hexadecimal value <code>0xffff</code> .

Return value

If the function succeeds, it returns the number of bytes (for the ANSI function) or WORDs (for the Unicode function) converted.

If the function fails, the return value is `GDI_ERROR`.

Remarks

This function attempts to identify a single-glyph representation for each character in the string pointed to by *lpstr*. While this is useful for certain low-level purposes (such as manipulating font files), higher-level applications that wish to map a string to glyphs will typically wish to use the [Uniscribe](#) functions.

NOTE

The wingdi.h header defines GetGlyphIndices as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetFontUnicodeRanges](#)

GetGlyphOutlineA function (wingdi.h)

4/21/2022 • 5 minutes to read • [Edit Online](#)

The **GetGlyphOutline** function retrieves the outline or bitmap for a character in the TrueType font that is selected into the specified device context.

Syntax

```
DWORD GetGlyphOutlineA(
    [in]    HDC      hdc,
    [in]    UINT     uChar,
    [in]    UINT     fuFormat,
    [out]   LPGLYPHMETRICS lpgm,
    [in]    DWORD    cjBuffer,
    [out]   LPVOID   pvBuffer,
    [in]    const MAT2 *lpmat2
);
```

Parameters

[in] hdc

A handle to the device context.

[in] uChar

The character for which data is to be returned.

[in] fuFormat

The format of the data that the function retrieves. This parameter can be one of the following values.

VALUE	MEANING
GGO_BEZIER	The function retrieves the curve data as a cubic Bézier spline (not in quadratic spline format).
GGO_BITMAP	The function retrieves the glyph bitmap. For information about memory allocation, see the following Remarks section.
GGO_GLYPH_INDEX	Indicates that the <i>uChar</i> parameter is a TrueType Glyph Index rather than a character code. See the ExtTextOut function for additional remarks on Glyph Indexing.
GGO_GRAY2_BITMAP	The function retrieves a glyph bitmap that contains five levels of gray.
GGO_GRAY4_BITMAP	The function retrieves a glyph bitmap that contains 17 levels of gray.

GGO_GRAY8_BITMAP	The function retrieves a glyph bitmap that contains 65 levels of gray.
GGO_METRICS	The function only retrieves the GLYPHMETRICS structure specified by <i>lpgm</i> . The <i>lpvBuffer</i> is ignored. This value affects the meaning of the function's return value upon failure; see the Return Values section.
GGO_NATIVE	The function retrieves the curve data points in the rasterizer's native format and uses the font's design units.
GGO_UNHINTED	The function only returns unhinted outlines. This flag only works in conjunction with GGO_BEZIER and GGO_NATIVE.

Note that, for the GGO_GRAYn_BITMAP values, the function retrieves a glyph bitmap that contains n^2+1 (*n* squared plus one) levels of gray.

[out] *lpgm*

A pointer to the [GLYPHMETRICS](#) structure describing the placement of the glyph in the character cell.

[in] *cjBuffer*

The size, in bytes, of the buffer (**pvBuffer*) where the function is to copy information about the outline character. If this value is zero, the function returns the required size of the buffer.

[out] *pvBuffer*

A pointer to the buffer that receives information about the outline character. If this value is **NULL**, the function returns the required size of the buffer.

[in] *lpmat2*

A pointer to a [MAT2](#) structure specifying a transformation matrix for the character.

Return value

If GGO_BITMAP, GGO_GRAY2_BITMAP, GGO_GRAY4_BITMAP, GGO_GRAY8_BITMAP, or GGO_NATIVE is specified and the function succeeds, the return value is greater than zero; otherwise, the return value is **GDI_ERROR**. If one of these flags is specified and the buffer size or address is zero, the return value specifies the required buffer size, in bytes.

If GGO_METRICS is specified and the function fails, the return value is **GDI_ERROR**.

Remarks

The glyph outline returned by the [GetGlyphOutline](#) function is for a grid-fitted glyph. (A grid-fitted glyph is a glyph that has been modified so that its bitmapped image conforms as closely as possible to the original design of the glyph.) If an application needs an unmodified glyph outline, it can request the glyph outline for a character in a font whose size is equal to the font's em unit. The value for a font's em unit is stored in the **otmEMSSquare** member of the [OUTLINETEXTMETRIC](#) structure.

The glyph bitmap returned by [GetGlyphOutline](#) when GGO_BITMAP is specified is a DWORD-aligned, row-oriented, monochrome bitmap. When GGO_GRAY2_BITMAP is specified, the bitmap returned is a DWORD-aligned, row-oriented array of bytes whose values range from 0 to 4. When GGO_GRAY4_BITMAP is specified,

the bitmap returned is a DWORD-aligned, row-oriented array of bytes whose values range from 0 to 16. When GGO_GRAY8_BITMAP is specified, the bitmap returned is a DWORD-aligned, row-oriented array of bytes whose values range from 0 to 64.

The native buffer returned by [GetGlyphOutline](#) when GGO_NATIVE is specified is a glyph outline. A glyph outline is returned as a series of one or more contours defined by a [TTPOLYGONHEADER](#) structure followed by one or more curves. Each curve in the contour is defined by a [TTPOLYCURVE](#) structure followed by a number of [POINTFX](#) data points. [POINTFX](#) points are absolute positions, not relative moves. The starting point of a contour is given by the [pfxStart](#) member of the [TTPOLYGONHEADER](#) structure. The starting point of each curve is the last point of the previous curve or the starting point of the contour. The count of data points in a curve is stored in the [cpfx](#) member of [TTPOLYCURVE](#) structure. The size of each contour in the buffer, in bytes, is stored in the [cb](#) member of [TTPOLYGONHEADER](#) structure. Additional curve definitions are packed into the buffer following preceding curves and additional contours are packed into the buffer following preceding contours. The buffer contains as many contours as fit within the buffer returned by [GetGlyphOutline](#).

The [GLYPHMETRICS](#) structure specifies the width of the character cell and the location of a glyph within the character cell. The origin of the character cell is located at the left side of the cell at the baseline of the font. The location of the glyph origin is relative to the character cell origin. The height of a character cell, the baseline, and other metrics global to the font are given by the [OUTLINETEXTMETRIC](#) structure.

An application can alter the characters retrieved in bitmap or native format by specifying a 2-by-2 transformation matrix in the *lpMatrix* parameter. For example the glyph can be modified by shear, rotation, scaling, or any combination of the three using matrix multiplication.

Additional information on a glyph outlines is located in the TrueType and the OpenType technical specifications.

NOTE

The wingdi.h header defines GetGlyphOutline as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[FORM_INFO_1](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

GLYPHMETRICS

[GetOutlineTextMetrics](#)

MAT2

[OUTLINETEXTMETRIC](#)

POINT

POINTFX

TTPOLYCURVE

TTPOLYGONHEADER

GetGlyphOutlineW function (wingdi.h)

4/21/2022 • 5 minutes to read • [Edit Online](#)

The **GetGlyphOutline** function retrieves the outline or bitmap for a character in the TrueType font that is selected into the specified device context.

Syntax

```
DWORD GetGlyphOutlineW(
    [in]    HDC      hdc,
    [in]    UINT     uChar,
    [in]    UINT     fuFormat,
    [out]   LPGLYPHMETRICS lpgm,
    [in]    DWORD    cjBuffer,
    [out]   LPVOID   pvBuffer,
    [in]    const MAT2 *lpmat2
);
```

Parameters

[in] hdc

A handle to the device context.

[in] uChar

The character for which data is to be returned.

[in] fuFormat

The format of the data that the function retrieves. This parameter can be one of the following values.

VALUE	MEANING
GGO_BEZIER	The function retrieves the curve data as a cubic Bézier spline (not in quadratic spline format).
GGO_BITMAP	The function retrieves the glyph bitmap. For information about memory allocation, see the following Remarks section.
GGO_GLYPH_INDEX	Indicates that the <i>uChar</i> parameter is a TrueType Glyph Index rather than a character code. See the ExtTextOut function for additional remarks on Glyph Indexing.
GGO_GRAY2_BITMAP	The function retrieves a glyph bitmap that contains five levels of gray.
GGO_GRAY4_BITMAP	The function retrieves a glyph bitmap that contains 17 levels of gray.

GGO_GRAY8_BITMAP	The function retrieves a glyph bitmap that contains 65 levels of gray.
GGO_METRICS	The function only retrieves the GLYPHMETRICS structure specified by <i>lpgm</i> . The <i>lpvBuffer</i> is ignored. This value affects the meaning of the function's return value upon failure; see the Return Values section.
GGO_NATIVE	The function retrieves the curve data points in the rasterizer's native format and uses the font's design units.
GGO_UNHINTED	The function only returns unhinted outlines. This flag only works in conjunction with GGO_BEZIER and GGO_NATIVE.

Note that, for the GGO_GRAYn_BITMAP values, the function retrieves a glyph bitmap that contains n^2+1 (*n* squared plus one) levels of gray.

[out] *lpgm*

A pointer to the [GLYPHMETRICS](#) structure describing the placement of the glyph in the character cell.

[in] *cjBuffer*

The size, in bytes, of the buffer (**pvBuffer*) where the function is to copy information about the outline character. If this value is zero, the function returns the required size of the buffer.

[out] *pvBuffer*

A pointer to the buffer that receives information about the outline character. If this value is **NULL**, the function returns the required size of the buffer.

[in] *lpmat2*

A pointer to a [MAT2](#) structure specifying a transformation matrix for the character.

Return value

If GGO_BITMAP, GGO_GRAY2_BITMAP, GGO_GRAY4_BITMAP, GGO_GRAY8_BITMAP, or GGO_NATIVE is specified and the function succeeds, the return value is greater than zero; otherwise, the return value is **GDI_ERROR**. If one of these flags is specified and the buffer size or address is zero, the return value specifies the required buffer size, in bytes.

If GGO_METRICS is specified and the function fails, the return value is **GDI_ERROR**.

Remarks

The glyph outline returned by the [GetGlyphOutline](#) function is for a grid-fitted glyph. (A grid-fitted glyph is a glyph that has been modified so that its bitmapped image conforms as closely as possible to the original design of the glyph.) If an application needs an unmodified glyph outline, it can request the glyph outline for a character in a font whose size is equal to the font's em unit. The value for a font's em unit is stored in the **otmEMSSquare** member of the [OUTLINETEXTMETRIC](#) structure.

The glyph bitmap returned by [GetGlyphOutline](#) when GGO_BITMAP is specified is a DWORD-aligned, row-oriented, monochrome bitmap. When GGO_GRAY2_BITMAP is specified, the bitmap returned is a DWORD-aligned, row-oriented array of bytes whose values range from 0 to 4. When GGO_GRAY4_BITMAP is specified,

the bitmap returned is a DWORD-aligned, row-oriented array of bytes whose values range from 0 to 16. When GGO_GRAY8_BITMAP is specified, the bitmap returned is a DWORD-aligned, row-oriented array of bytes whose values range from 0 to 64.

The native buffer returned by [GetGlyphOutline](#) when GGO_NATIVE is specified is a glyph outline. A glyph outline is returned as a series of one or more contours defined by a [TTPOLYGONHEADER](#) structure followed by one or more curves. Each curve in the contour is defined by a [TTPOLYCURVE](#) structure followed by a number of [POINTFX](#) data points. [POINTFX](#) points are absolute positions, not relative moves. The starting point of a contour is given by the [pfxStart](#) member of the [TTPOLYGONHEADER](#) structure. The starting point of each curve is the last point of the previous curve or the starting point of the contour. The count of data points in a curve is stored in the [cpfx](#) member of [TTPOLYCURVE](#) structure. The size of each contour in the buffer, in bytes, is stored in the [cb](#) member of [TTPOLYGONHEADER](#) structure. Additional curve definitions are packed into the buffer following preceding curves and additional contours are packed into the buffer following preceding contours. The buffer contains as many contours as fit within the buffer returned by [GetGlyphOutline](#).

The [GLYPHMETRICS](#) structure specifies the width of the character cell and the location of a glyph within the character cell. The origin of the character cell is located at the left side of the cell at the baseline of the font. The location of the glyph origin is relative to the character cell origin. The height of a character cell, the baseline, and other metrics global to the font are given by the [OUTLINETEXTMETRIC](#) structure.

An application can alter the characters retrieved in bitmap or native format by specifying a 2-by-2 transformation matrix in the *lpMatrix* parameter. For example the glyph can be modified by shear, rotation, scaling, or any combination of the three using matrix multiplication.

Additional information on a glyph outlines is located in the TrueType and the OpenType technical specifications.

NOTE

The wingdi.h header defines GetGlyphOutline as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[FORM_INFO_1](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

GLYPHMETRICS

[GetOutlineTextMetrics](#)

MAT2

[OUTLINETEXTMETRIC](#)

POINT

POINTFX

TTPOLYCURVE

TTPOLYGONHEADER

GetGraphicsMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetGraphicsMode** function retrieves the current graphics mode for the specified device context.

Syntax

```
int GetGraphicsMode(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

A handle to the device context.

Return value

If the function succeeds, the return value is the current graphics mode. It can be one of the following values.

VALUE	MEANING
GM_COMPATIBLE	The current graphics mode is the compatible graphics mode, a mode that is compatible with 16-bit Windows. In this graphics mode, an application cannot set or modify the world transformation for the specified device context. The compatible graphics mode is the default graphics mode.
GM_ADVANCED	The current graphics mode is the advanced graphics mode, a mode that allows world transformations. In this graphics mode, an application can set or modify the world transformation for the specified device context.

Otherwise, the return value is zero.

Remarks

An application can set the graphics mode for a device context by calling the [SetGraphicsMode](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[SetGraphicsMode](#)

GetGValue macro (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetGValue** macro retrieves an intensity value for the green component of a red, green, blue (RGB) value.

Syntax

```
void GetGValue(  
    rgb  
) ;
```

Parameters

rgb

Specifies an RGB color value.

Return value

None

Remarks

The intensity value is in the range 0 through 255.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[Color Macros](#)

[Colors Overview](#)

[GetBValue](#)

[GetRValue](#)

[PALETTEINDEX](#)

[PALETTERGB](#)

RGB

GetKerningPairsA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetKerningPairs** function retrieves the character-kerning pairs for the currently selected font for the specified device context.

Syntax

```
DWORD GetKerningPairsA(
    [in]    HDC      hdc,
    [in]    DWORD    nPairs,
    [out]   LPKERNINGPAIR lpKernPair
);
```

Parameters

[in] hdc

A handle to the device context.

[in] nPairs

The number of pairs in the *lpkrnpair* array. If the font has more than *nNumPairs* kerning pairs, the function returns an error.

[out] lpKernPair

A pointer to an array of **KERNINGPAIR** structures that receives the kerning pairs. The array must contain at least as many structures as specified by the *nNumPairs* parameter. If this parameter is **NULL**, the function returns the total number of kerning pairs for the font.

Return value

If the function succeeds, the return value is the number of kerning pairs returned.

If the function fails, the return value is zero.

Remarks

NOTE

The wingdi.h header defines **GetKerningPairs** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[KERNINGPAIR](#)

GetKerningPairsW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetKerningPairs** function retrieves the character-kerning pairs for the currently selected font for the specified device context.

Syntax

```
DWORD GetKerningPairsW(
    [in]    HDC      hdc,
    [in]    DWORD    nPairs,
    [out]   LPKERNINGPAIR lpKernPair
);
```

Parameters

[in] hdc

A handle to the device context.

[in] nPairs

The number of pairs in the *lpkrnpair* array. If the font has more than *nNumPairs* kerning pairs, the function returns an error.

[out] lpKernPair

A pointer to an array of **KERNINGPAIR** structures that receives the kerning pairs. The array must contain at least as many structures as specified by the *nNumPairs* parameter. If this parameter is **NULL**, the function returns the total number of kerning pairs for the font.

Return value

If the function succeeds, the return value is the number of kerning pairs returned.

If the function fails, the return value is zero.

Remarks

NOTE

The wingdi.h header defines **GetKerningPairs** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[KERNINGPAIR](#)

GetLayout function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetLayout** function returns the layout of a device context (DC).

Syntax

```
DWORD GetLayout(  
    [in] HDC hdc  
>;
```

Parameters

`[in] hdc`

A handle to the device context.

Return value

If the function succeeds, it returns the layout flags for the current device context.

If the function fails, it returns GDI_ERROR. For extended error information, call [GetLastError](#).

Remarks

The layout specifies the order in which text and graphics are revealed in a window or device context. The default is left to right. The **GetLayout** function tells you if the default has been changed through a call to **SetLayout**. For more information, see "Window Layout and Mirroring" in [Window Features](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Device Context Functions](#)

Device Contexts Overview

[SetLayout](#)

GetMapMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetMapMode** function retrieves the current mapping mode.

Syntax

```
int GetMapMode(  
    [in] HDC hdc  
>;
```

Parameters

[in] `hdc`

A handle to the device context.

Return value

If the function succeeds, the return value specifies the mapping mode.

If the function fails, the return value is zero.

Remarks

The following are the various mapping modes.

MODE	DESCRIPTION
MM_ANISOTROPIC	Logical units are mapped to arbitrary units with arbitrarily scaled axes. Use the SetWindowExtEx and SetViewportExtEx functions to specify the units, orientation, and scaling required.
MM_HIENGLISH	Each logical unit is mapped to 0.001 inch. Positive x is to the right; positive y is up.
MM HIMETRIC	Each logical unit is mapped to 0.01 millimeter. Positive x is to the right; positive y is up.
MM_ISOTROPIC	Logical units are mapped to arbitrary units with equally scaled axes; that is, one unit along the x-axis is equal to one unit along the y-axis. Use the SetWindowExtEx and SetViewportExtEx functions to specify the units and the orientation of the axes. Graphics device interface makes adjustments as necessary to ensure the x and y units remain the same size. (When the windows extent is set, the viewport will be adjusted to keep the units isotropic).
MM LOENGLISH	Each logical unit is mapped to 0.01 inch. Positive x is to the right; positive y is up.

MM_LOMETRIC	Each logical unit is mapped to 0.1 millimeter. Positive x is to the right; positive y is up.
MM_TEXT	Each logical unit is mapped to one device pixel. Positive x is to the right; positive y is down.
MM_TWIPS	Each logical unit is mapped to one twentieth of a printer's point (1/1440 inch, also called a "twip"). Positive x is to the right; positive y is up.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[SetMapMode](#)

[SetViewportExtEx](#)

[SetWindowExtEx](#)

GetMetaFileA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

[GetMetaFile is no longer available for use as of Windows 2000. Instead, use [GetEnhMetaFile](#).]

The **GetMetaFile** function creates a handle that identifies the metafile stored in the specified file.

Syntax

```
HMETAFILE GetMetaFileA(
    [in] LPCSTR lpName
);
```

Parameters

[in] lpName

A pointer to a null-terminated string that specifies the name of a metafile.

Return value

If the function succeeds, the return value is a handle to the metafile.

If the function fails, the return value is **NULL**.

Remarks

This function is not implemented in the Win32 API. It is provided for compatibility with 16-bit versions of Windows. In Win32 applications, use the [GetEnhMetaFile](#) function.

NOTE

The wingdi.h header defines GetMetaFile as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GetEnhMetaFile](#)

GetMetaFileBitsEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetMetaFileBitsEx** function retrieves the contents of a Windows-format metafile and copies them into the specified buffer.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [GetEnhMetaFileBits](#).

Syntax

```
UINT GetMetaFileBitsEx(
    [in] HMF,
    [in] UINT cbBuffer,
    [out] LPVOID lpData
);
```

Parameters

[in] *hMF*

A handle to a Windows-format metafile.

[in] *cbBuffer*

The size, in bytes, of the buffer to receive the data.

[out] *lpData*

A pointer to a buffer that receives the metafile data. The buffer must be sufficiently large to contain the data. If *lpData* is **NULL**, the function returns the number of bytes required to hold the data.

Return value

If the function succeeds and the buffer pointer is **NULL**, the return value is the number of bytes required for the buffer; if the function succeeds and the buffer pointer is a valid pointer, the return value is the number of bytes copied.

If the function fails, the return value is zero.

Remarks

After the Windows-metafile bits are retrieved, they can be used to create a memory-based metafile by calling the [SetMetaFileBitsEx](#) function.

The **GetMetaFileBitsEx** function does not invalidate the metafile handle. An application must delete this handle by calling the [DeleteMetaFile](#) function.

To convert a Windows-format metafile into an enhanced-format metafile, use the [SetWinMetaFileBits](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteMetaFile](#)

[GetEnhMetaFileBits](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SetMetaFileBitsEx](#)

[SetWinMetaFileBits](#)

GetMetaFileW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

[GetMetaFile is no longer available for use as of Windows 2000. Instead, use [GetEnhMetaFile](#).]

The **GetMetaFile** function creates a handle that identifies the metafile stored in the specified file.

Syntax

```
HMETAFILE GetMetaFileW(
    [in] LPCWSTR lpName
);
```

Parameters

[in] lpName

A pointer to a null-terminated string that specifies the name of a metafile.

Return value

If the function succeeds, the return value is a handle to the metafile.

If the function fails, the return value is **NULL**.

Remarks

This function is not implemented in the Win32 API. It is provided for compatibility with 16-bit versions of Windows. In Win32 applications, use the [GetEnhMetaFile](#) function.

NOTE

The wingdi.h header defines GetMetaFile as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GetEnhMetaFile](#)

GetMetaRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetMetaRgn** function retrieves the current metaregion for the specified device context.

Syntax

```
int GetMetaRgn(
    [in] HDC hdc,
    [in] HRGN hrgn
);
```

Parameters

[in] *hdc*

A handle to the device context.

[in] *hrgn*

A handle to an existing region before the function is called. After the function returns, this parameter is a handle to a copy of the current metaregion.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

If the function succeeds, *hrgn* is a handle to a copy of the current metaregion. Subsequent changes to this copy will not affect the current metaregion.

The current clipping region of a device context is defined by the intersection of its clipping region and its metaregion.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[Clipping Functions](#)

[Clipping Overview](#)

[SetMetaRgn](#)

GetMiterLimit function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetMiterLimit** function retrieves the miter limit for the specified device context.

Syntax

```
BOOL GetMiterLimit(
    [in]    HDC      hdc,
    [out]   PFLOAT  plimit
);
```

Parameters

[in] hdc

Handle to the device context.

[out] plimit

Pointer to a floating-point value that receives the current miter limit.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The miter limit is used when drawing geometric lines that have miter joins.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtCreatePen](#)

[Path Functions](#)

[Paths Overview](#)

[SetMiterLimit](#)

GetNearestColor function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetNearestColor** function retrieves a color value identifying a color from the system palette that will be displayed when the specified color value is used.

Syntax

```
COLORREF GetNearestColor(
    [in] HDC      hdc,
    [in] COLORREF color
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `color`

A color value that identifies a requested color. To create a `COLORREF` color value, use the `RGB` macro.

Return value

If the function succeeds, the return value identifies a color from the system palette that corresponds to the given color value.

If the function fails, the return value is `CLR_INVALID`.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORREF](#)

[Color Functions](#)

[Colors Overview](#)

[GetDeviceCaps](#)

[GetNearestPaletteIndex](#)

[RGB](#)

GetNearestPaletteIndex function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetNearestPaletteIndex** function retrieves the index for the entry in the specified logical palette most closely matching a specified color value.

Syntax

```
UINT GetNearestPaletteIndex(
    [in] HPALETTE h,
    [in] COLORREF color
);
```

Parameters

[in] *h*

A handle to a logical palette.

[in] *color*

A color to be matched. To create a **COLORREF** color value, use the **RGB** macro.

Return value

If the function succeeds, the return value is the index of an entry in a logical palette.

If the function fails, the return value is **CLR_INVALID**.

Remarks

An application can determine whether a device supports palette operations by calling the **GetDeviceCaps** function and specifying the **RASTERCAPS** constant.

If the given logical palette contains entries with the **PC_EXPLICIT** flag set, the return value is undefined.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[COLORREF](#)

[Color Functions](#)

[Colors Overview](#)

[GetDeviceCaps](#)

[GetNearestColor](#)

[GetPaletteEntries](#)

[GetSystemPaletteEntries](#)

[RGB](#)

GetObject function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **GetObject** function retrieves information for the specified graphics object.

Syntax

```
int GetObject(
    [in]  HANDLE h,
    [in]  int     c,
    [out] LPVOID pv
);
```

Parameters

[in] **h**

A handle to the graphics object of interest. This can be a handle to one of the following: a logical bitmap, a brush, a font, a palette, a pen, or a device independent bitmap created by calling the [CreateDIBSection](#) function.

[in] **c**

The number of bytes of information to be written to the buffer.

[out] **pv**

A pointer to a buffer that receives the information about the specified graphics object.

The following table shows the type of information the buffer receives for each type of graphics object you can specify with *hgalobj*.

OBJECT TYPE	DATA WRITTEN TO BUFFER
HBITMAP	BITMAP
HBITMAP returned from a call to CreateDIBSection	DIBSECTION, if <i>cbBuffer</i> is set to <code>sizeof (DIBSECTION)</code> , or BITMAP, if <i>cbBuffer</i> is set to <code>sizeof (BITMAP)</code> .
HPALETTE	A WORD count of the number of entries in the logical palette
HPEN returned from a call to ExtCreatePen	EXTLOGOPEN
HPEN	LOGOPEN
HBRUSH	LOGBRUSH

HFONT	LOGFONT
-------	---------

If the *lpvObject* parameter is **NULL**, the function return value is the number of bytes required to store the information it writes to the buffer for the specified graphics object.

The address of *lpvObject* must be on a 4-byte boundary; otherwise, **GetObject** fails.

Return value

If the function succeeds, and *lpvObject* is a valid pointer, the return value is the number of bytes stored into the buffer.

If the function succeeds, and *lpvObject* is **NULL**, the return value is the number of bytes required to hold the information the function would store into the buffer.

If the function fails, the return value is zero.

Remarks

The buffer pointed to by the *lpvObject* parameter must be sufficiently large to receive the information about the graphics object. Depending on the graphics object, the function uses a **BITMAP**, **DIBSECTION**, **EXTLOGOPEN**, **LOGBRUSH**, **LOGFONT**, or **LOGOPEN** structure, or a count of table entries (for a logical palette).

If *hgdiobj* is a handle to a bitmap created by calling **CreateDIBSection**, and the specified buffer is large enough, the **GetObject** function returns a **DIBSECTION** structure. In addition, the **bmBits** member of the **BITMAP** structure contained within the **DIBSECTION** will contain a pointer to the bitmap's bit values.

If *hgdiobj* is a handle to a bitmap created by any other means, **GetObject** returns only the width, height, and color format information of the bitmap. You can obtain the bitmap's bit values by calling the **GetDIBits** or **GetBitmapBits** function.

If *hgdiobj* is a handle to a logical palette, **GetObject** retrieves a 2-byte integer that specifies the number of entries in the palette. The function does not retrieve the **LOGPALETTE** structure defining the palette. To retrieve information about palette entries, an application can call the **GetPaletteEntries** function.

If *hgdiobj* is a handle to a font, the **LOGFONT** that is returned is the **LOGFONT** used to create the font. If Windows had to make some interpolation of the font because the precise **LOGFONT** could not be represented, the interpolation will not be reflected in the **LOGFONT**. For example, if you ask for a vertical version of a font that doesn't support vertical painting, the **LOGFONT** indicates the font is vertical, but Windows will paint it horizontally.

Examples

For an example, see [Storing an Image](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAP](#)

[CreateDIBSection](#)

[DIBSECTION](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EXTLOGOPEN](#)

[GetBitmapBits](#)

[GetDIBits](#)

[GetPaletteEntries](#)

[GetRegionData](#)

[LOGBRUSH](#)

[LOGFONT](#)

[LOGPALETTE](#)

[LOGPEN](#)

GetObjectA function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **GetObject** function retrieves information for the specified graphics object.

Syntax

```
int GetObjectA(
    [in]  HANDLE h,
    [in]  int     c,
    [out] LPVOID pv
);
```

Parameters

[in] **h**

A handle to the graphics object of interest. This can be a handle to one of the following: a logical bitmap, a brush, a font, a palette, a pen, or a device independent bitmap created by calling the [CreateDIBSection](#) function.

[in] **c**

The number of bytes of information to be written to the buffer.

[out] **pv**

A pointer to a buffer that receives the information about the specified graphics object.

The following table shows the type of information the buffer receives for each type of graphics object you can specify with *hgalobj*.

OBJECT TYPE	DATA WRITTEN TO BUFFER
HBITMAP	BITMAP
HBITMAP returned from a call to CreateDIBSection	DIBSECTION, if <i>cbBuffer</i> is set to <code>sizeof (DIBSECTION)</code> , or BITMAP, if <i>cbBuffer</i> is set to <code>sizeof (BITMAP)</code> .
HPALETTE	A WORD count of the number of entries in the logical palette
HPEN returned from a call to ExtCreatePen	EXTLOGOPEN
HPEN	LOGOPEN
HBRUSH	LOGBRUSH

HFONT

LOGFONT

If the *lpvObject* parameter is **NULL**, the function return value is the number of bytes required to store the information it writes to the buffer for the specified graphics object.

The address of *lpvObject* must be on a 4-byte boundary; otherwise, **GetObject** fails.

Return value

If the function succeeds, and *lpvObject* is a valid pointer, the return value is the number of bytes stored into the buffer.

If the function succeeds, and *lpvObject* is **NULL**, the return value is the number of bytes required to hold the information the function would store into the buffer.

If the function fails, the return value is zero.

Remarks

The buffer pointed to by the *lpvObject* parameter must be sufficiently large to receive the information about the graphics object. Depending on the graphics object, the function uses a [BITMAP](#), [DIBSECTION](#), [EXTLOGOPEN](#), [LOGBRUSH](#), [LOGFONT](#), or [LOGOPEN](#) structure, or a count of table entries (for a logical palette).

If *hgdiobj* is a handle to a bitmap created by calling [CreateDIBSection](#), and the specified buffer is large enough, the **GetObject** function returns a [DIBSECTION](#) structure. In addition, the **bmBits** member of the [BITMAP](#) structure contained within the [DIBSECTION](#) will contain a pointer to the bitmap's bit values.

If *hgdiobj* is a handle to a bitmap created by any other means, **GetObject** returns only the width, height, and color format information of the bitmap. You can obtain the bitmap's bit values by calling the [GetDIBits](#) or [GetBitmapBits](#) function.

If *hgdiobj* is a handle to a logical palette, **GetObject** retrieves a 2-byte integer that specifies the number of entries in the palette. The function does not retrieve the [LOGPALETTE](#) structure defining the palette. To retrieve information about palette entries, an application can call the [GetPaletteEntries](#) function.

If *hgdiobj* is a handle to a font, the [LOGFONT](#) that is returned is the [LOGFONT](#) used to create the font. If Windows had to make some interpolation of the font because the precise [LOGFONT](#) could not be represented, the interpolation will not be reflected in the [LOGFONT](#). For example, if you ask for a vertical version of a font that doesn't support vertical painting, the [LOGFONT](#) indicates the font is vertical, but Windows will paint it horizontally.

Examples

For an example, see [Storing an Image](#).

NOTE

The wingdi.h header defines **GetObject** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAP](#)

[CreateDIBSection](#)

[DIBSECTION](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EXTLOGOPEN](#)

[GetBitmapBits](#)

[GetDIBits](#)

[GetPaletteEntries](#)

[GetRegionData](#)

[LOGBRUSH](#)

[LOGFONT](#)

[LOGPALETTE](#)

[LOGOPEN](#)

GetObjectType function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetObjectType** retrieves the type of the specified object.

Syntax

```
DWORD GetObjectType(  
    [in] HGDIOBJ h  
)
```

Parameters

[in] h

A handle to the graphics object.

Return value

If the function succeeds, the return value identifies the object. This value can be one of the following.

VALUE	MEANING
OBJ_BITMAP	Bitmap
OBJ_BRUSH	Brush
OBJ_COLORSPACE	Color space
OBJ_DC	Device context
OBJ_ENHMETADC	Enhanced metafile DC
OBJ_ENHMETAFILE	Enhanced metafile
OBJ_EXTPEN	Extended pen
OBJ_FONT	Font
OBJ_MEMDC	Memory DC
OBJ_METAFILE	Metafile
OBJ_METADC	Metafile DC
OBJ_PAL	Palette
OBJ_PEN	Pen

OBJ_REGION	Region
------------	--------

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetObject](#)

[SelectObject](#)

GetObjectW function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **GetObject** function retrieves information for the specified graphics object.

Syntax

```
int GetObjectW(
    [in]  HANDLE h,
    [in]  int     c,
    [out] LPVOID pv
);
```

Parameters

[in] **h**

A handle to the graphics object of interest. This can be a handle to one of the following: a logical bitmap, a brush, a font, a palette, a pen, or a device independent bitmap created by calling the [CreateDIBSection](#) function.

[in] **c**

The number of bytes of information to be written to the buffer.

[out] **pv**

A pointer to a buffer that receives the information about the specified graphics object.

The following table shows the type of information the buffer receives for each type of graphics object you can specify with *hgalobj*.

OBJECT TYPE	DATA WRITTEN TO BUFFER
HBITMAP	BITMAP
HBITMAP returned from a call to CreateDIBSection	DIBSECTION, if <i>cbBuffer</i> is set to <code>sizeof (DIBSECTION)</code> , or BITMAP, if <i>cbBuffer</i> is set to <code>sizeof (BITMAP)</code> .
HPALETTE	A WORD count of the number of entries in the logical palette
HPEN returned from a call to ExtCreatePen	EXTLOGOPEN
HPEN	LOGOPEN
HBRUSH	LOGBRUSH

HFONT

LOGFONT

If the *lpvObject* parameter is **NULL**, the function return value is the number of bytes required to store the information it writes to the buffer for the specified graphics object.

The address of *lpvObject* must be on a 4-byte boundary; otherwise, **GetObject** fails.

Return value

If the function succeeds, and *lpvObject* is a valid pointer, the return value is the number of bytes stored into the buffer.

If the function succeeds, and *lpvObject* is **NULL**, the return value is the number of bytes required to hold the information the function would store into the buffer.

If the function fails, the return value is zero.

Remarks

The buffer pointed to by the *lpvObject* parameter must be sufficiently large to receive the information about the graphics object. Depending on the graphics object, the function uses a [BITMAP](#), [DIBSECTION](#), [EXTLOGOPEN](#), [LOGBRUSH](#), [LOGFONT](#), or [LOGOPEN](#) structure, or a count of table entries (for a logical palette).

If *hgdiobj* is a handle to a bitmap created by calling [CreateDIBSection](#), and the specified buffer is large enough, the **GetObject** function returns a [DIBSECTION](#) structure. In addition, the **bmBits** member of the [BITMAP](#) structure contained within the [DIBSECTION](#) will contain a pointer to the bitmap's bit values.

If *hgdiobj* is a handle to a bitmap created by any other means, **GetObject** returns only the width, height, and color format information of the bitmap. You can obtain the bitmap's bit values by calling the [GetDIBits](#) or [GetBitmapBits](#) function.

If *hgdiobj* is a handle to a logical palette, **GetObject** retrieves a 2-byte integer that specifies the number of entries in the palette. The function does not retrieve the [LOGPALETTE](#) structure defining the palette. To retrieve information about palette entries, an application can call the [GetPaletteEntries](#) function.

If *hgdiobj* is a handle to a font, the [LOGFONT](#) that is returned is the [LOGFONT](#) used to create the font. If Windows had to make some interpolation of the font because the precise [LOGFONT](#) could not be represented, the interpolation will not be reflected in the [LOGFONT](#). For example, if you ask for a vertical version of a font that doesn't support vertical painting, the [LOGFONT](#) indicates the font is vertical, but Windows will paint it horizontally.

Examples

For an example, see [Storing an Image](#).

NOTE

The wingdi.h header defines **GetObject** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAP](#)

[CreateDIBSection](#)

[DIBSECTION](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EXTLOGOPEN](#)

[GetBitmapBits](#)

[GetDIBits](#)

[GetPaletteEntries](#)

[GetRegionData](#)

[LOGBRUSH](#)

[LOGFONT](#)

[LOGPALETTE](#)

[LOGOPEN](#)

GetOutlineTextMetricsA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetOutlineTextMetrics** function retrieves text metrics for TrueType fonts.

Syntax

```
UINT GetOutlineTextMetricsA(
    [in]         HDC           hdc,
    [in]         UINT          cjCopy,
    [out, optional] LPOUTLINETEXTMETRICA potm
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `cjCopy`

The size, in bytes, of the array that receives the text metrics.

[out, optional] `potm`

A pointer to an **OUTLINETEXTMETRIC** structure. If this parameter is **NULL**, the function returns the size of the buffer required for the retrieved metric data.

Return value

If the function succeeds, the return value is nonzero or the size of the required buffer.

If the function fails, the return value is zero.

Remarks

The **OUTLINETEXTMETRIC** structure contains most of the text metric information provided for TrueType fonts (including a **TEXTMETRIC** structure). The sizes returned in **OUTLINETEXTMETRIC** are in logical units; they depend on the current mapping mode.

NOTE

The wingdi.h header defines **GetOutlineTextMetrics** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextMetrics](#)

[OUTLINETEXTMETRIC](#)

[TEXTMETRIC](#)

GetOutlineTextMetricsW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetOutlineTextMetrics** function retrieves text metrics for TrueType fonts.

Syntax

```
UINT GetOutlineTextMetricsW(
    [in]           HDC           hdc,
    [in]           UINT          cjCopy,
    [out, optional] LPOUTLINETEXTMETRICW potm
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `cjCopy`

The size, in bytes, of the array that receives the text metrics.

[out, optional] `potm`

A pointer to an **OUTLINETEXTMETRIC** structure. If this parameter is **NULL**, the function returns the size of the buffer required for the retrieved metric data.

Return value

If the function succeeds, the return value is nonzero or the size of the required buffer.

If the function fails, the return value is zero.

Remarks

The **OUTLINETEXTMETRIC** structure contains most of the text metric information provided for TrueType fonts (including a **TEXTMETRIC** structure). The sizes returned in **OUTLINETEXTMETRIC** are in logical units; they depend on the current mapping mode.

NOTE

The wingdi.h header defines **GetOutlineTextMetrics** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextMetrics](#)

[OUTLINETEXTMETRIC](#)

[TEXTMETRIC](#)

GetPaletteEntries function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetPaletteEntries** function retrieves a specified range of palette entries from the given logical palette.

Syntax

```
UINT GetPaletteEntries(
    [in] HPALETTE      hpal,
    [in] UINT          iStart,
    [in] UINT          cEntries,
    [out] LPPALETTEENTRY pPalEntries
);
```

Parameters

[in] **hpal**

A handle to the logical palette.

[in] **iStart**

The first entry in the logical palette to be retrieved.

[in] **cEntries**

The number of entries in the logical palette to be retrieved.

[out] **pPalEntries**

A pointer to an array of **PALETTEENTRY** structures to receive the palette entries. The array must contain at least as many structures as specified by the *nEntries* parameter.

Return value

If the function succeeds and the handle to the logical palette is a valid pointer (not **NULL**), the return value is the number of entries retrieved from the logical palette. If the function succeeds and handle to the logical palette is **NULL**, the return value is the number of entries in the given palette.

If the function fails, the return value is zero.

Remarks

An application can determine whether a device supports palette operations by calling the [GetDeviceCaps](#) function and specifying the **RASTERCAPS** constant.

If the *nEntries* parameter specifies more entries than exist in the palette, the remaining members of the **PALETTEENTRY** structure are not altered.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[GetDeviceCaps](#)

[GetSystemPaletteEntries](#)

[PALETTEENTRY](#)

[SetPaletteEntries](#)

GetPath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetPath** function retrieves the coordinates defining the endpoints of lines and the control points of curves found in the path that is selected into the specified device context.

Syntax

```
int GetPath(
    [in]  HDC      hdc,
    [out] LPPOINT apt,
    [out] LPBYTE  aj,
    [in]   int     cpt
);
```

Parameters

[in] `hdc`

A handle to a device context that contains a closed path.

[out] `apt`

A pointer to an array of [POINT](#) structures that receives the line endpoints and curve control points, in logical coordinates.

[out] `aj`

A pointer to an array of bytes that receives the vertex types. This parameter can be one of the following values.

TYPE	DESCRIPTION
<code>PT_MOVETO</code>	Specifies that the corresponding point in the <i>lpPoints</i> parameter starts a disjoint figure.
<code>PT_LINETO</code>	Specifies that the previous point and the corresponding point in <i>lpPoints</i> are the endpoints of a line.
<code>PT_BEZIERTO</code>	Specifies that the corresponding point in <i>lpPoints</i> is a control point or ending point for a Bézier curve. <code>PT_BEZIERTO</code> values always occur in sets of three. The point in the path immediately preceding them defines the starting point for the Bézier curve. The first two <code>PT_BEZIERTO</code> points are the control points, and the third <code>PT_BEZIERTO</code> point is the ending (if hard-coded) point.

A `PT_LINETO` or `PT_BEZIERTO` value may be combined with the following value (by using the bitwise operator OR) to indicate that the corresponding point is the last point in a figure and the figure should be closed.

FLAG	DESCRIPTION
------	-------------

PT_CLOSEFIGURE	Specifies that the figure is automatically closed after the corresponding line or curve is drawn. The figure is closed by drawing a line from the line or curve endpoint to the point corresponding to the last PT_MOVETO.
-----------------------	--

[in] cpt

The total number of **POINT** structures that can be stored in the array pointed to by *lpPoints*. This value must be the same as the number of bytes that can be placed in the array pointed to by *lpTypes*.

Return value

If the *nSize* parameter is nonzero, the return value is the number of points enumerated. If *nSize* is 0, the return value is the total number of points in the path (and **GetPath** writes nothing to the buffers). If *nSize* is nonzero and is less than the number of points in the path, the return value is 1.

Remarks

The device context identified by the *hdc* parameter must contain a closed path.

The points of the path are returned in logical coordinates. Points are stored in the path in device coordinates, so **GetPath** changes the points from device coordinates to logical coordinates by using the inverse of the current transformation.

The **FlattenPath** function may be called before **GetPath** to convert all curves in the path into line segments.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[FlattenPath](#)

[POINT](#)

[Path Functions](#)

[Paths Overview](#)

[PolyDraw](#)

[WidenPath](#)

GetPixel function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetPixel** function retrieves the red, green, blue (RGB) color value of the pixel at the specified coordinates.

Syntax

```
COLORREF GetPixel(  
    [in] HDC hdc,  
    [in] int x,  
    [in] int y  
) ;
```

Parameters

[in] `hdc`

A handle to the [device context](#).

[in] `x`

The x-coordinate, in logical units, of the pixel to be examined.

[in] `y`

The y-coordinate, in logical units, of the pixel to be examined.

Return value

The return value is the [COLORREF](#) value that specifies the RGB of the pixel. If the pixel is outside of the current clipping region, the return value is CLR_INVALID (0xFFFFFFFF defined in Wingdi.h).

Remarks

The pixel must be within the boundaries of the current clipping region.

Not all devices support **GetPixel**. An application should call [GetDeviceCaps](#) to determine whether a specified device supports this function.

A bitmap must be selected within the device context, otherwise, CLR_INVALID is returned on all pixels.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[COLORREF](#)

[GetDeviceCaps](#)

[SetPixel](#)

GetPolyFillMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetPolyFillMode** function retrieves the current polygon fill mode.

Syntax

```
int GetPolyFillMode(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

Handle to the device context.

Return value

If the function succeeds, the return value specifies the polygon fill mode, which can be one of the following values.

VALUE	MEANING
ALTERNATE	Selects alternate mode (fills area between odd-numbered and even-numbered polygon sides on each scan line).
WINDING	Selects winding mode (fills any region with a nonzero winding value).

If an error occurs, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Region Functions](#)

[Regions Overview](#)

[SetPolyFillMode](#)

GetRandomRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetRandomRgn** function copies the system clipping region of a specified device context to a specific region.

Syntax

```
int GetRandomRgn(  
    [in] HDC hdc,  
    [in] HRGN hrgn,  
    [in] INT i  
>;
```

Parameters

[in] *hdc*

A handle to the device context.

[in] *hrgn*

A handle to a region. Before the function is called, this identifies an existing region. After the function returns, this identifies a copy of the current system region. The old region identified by *hrgn* is overwritten.

[in] *i*

This parameter must be SYSRGN.

Return value

If the function succeeds, the return value is 1. If the function fails, the return value is -1. If the region to be retrieved is **NULL**, the return value is 0. If the function fails or the region to be retrieved is **NULL**, *hrgn* is not initialized.

Remarks

When using the SYSRGN flag, note that the system clipping region might not be current because of window movements. Nonetheless, it is safe to retrieve and use the system clipping region within the [BeginPaint-EndPaint](#) block during **WM_PAINT** processing. In this case, the system region is the intersection of the update region and the current visible area of the window. Any window movement following the return of **GetRandomRgn** and before **EndPaint** will result in a new **WM_PAINT** message. Any other use of the SYSRGN flag may result in painting errors in your application.

The region returned is in screen coordinates.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPaint](#)

[Clipping Functions](#)

[Clipping Overview](#)

[EndPaint](#)

[ExtSelectClipRgn](#)

[GetClipBox](#)

[GetClipRgn](#)

[GetRegionData](#)

[OffsetRgn](#)

GetRasterizerCaps function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetRasterizerCaps** function returns flags indicating whether TrueType fonts are installed in the system.

Syntax

```
BOOL GetRasterizerCaps(
    [out] LPRASTERIZER_STATUS lpraststat,
    [in]  UINT             cjBytes
);
```

Parameters

[out] lpraststat

A pointer to a [RASTERIZER_STATUS](#) structure that receives information about the rasterizer.

[in] cjBytes

The number of bytes to be copied into the structure pointed to by the *lprs* parameter.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **GetRasterizerCaps** function enables applications and printer drivers to determine whether TrueType fonts are installed.

If the **TT_AVAILABLE** flag is set in the **wFlags** member of the [RASTERIZER_STATUS](#) structure, at least one TrueType font is installed. If the **TT_ENABLED** flag is set, TrueType is enabled for the system.

The actual number of bytes copied is either the member specified in the **cb** parameter or the length of the [RASTERIZER_STATUS](#) structure, whichever is less.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetOutlineTextMetrics](#)

[RASTERIZER_STATUS](#)

GetRegionData function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetRegionData** function fills the specified buffer with data describing a region. This data includes the dimensions of the rectangles that make up the region.

Syntax

```
DWORD GetRegionData(
    [in] Hrgn     hrgn,
    [in] DWORD    nCount,
    [out] LPRGNDATA lpRgnData
);
```

Parameters

[in] hrgn

A handle to the region.

[in] nCount

The size, in bytes, of the *lpRgnData* buffer.

[out] lpRgnData

A pointer to a [RGNDATA](#) structure that receives the information. The dimensions of the region are in logical units. If this parameter is **NULL**, the return value contains the number of bytes needed for the region data.

Return value

If the function succeeds and *dwCount* specifies an adequate number of bytes, the return value is always *dwCount*. If *dwCount* is too small or the function fails, the return value is 0. If *lpRgnData* is **NULL**, the return value is the required number of bytes.

If the function fails, the return value is zero.

Remarks

The **GetRegionData** function is used in conjunction with the [ExtCreateRegion](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePolyPolygonRgn](#)

[CreatePolygonRgn](#)

[CreateRectRgn](#)

[CreateRectRgnIndirect](#)

[CreateRoundRectRgn](#)

[ExtCreateRegion](#)

[RGNDATA](#)

[Region Functions](#)

[Regions Overview](#)

GetRgnBox function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetRgnBox** function retrieves the bounding rectangle of the specified region.

Syntax

```
int GetRgnBox(
    [in] HRGN hrgn,
    [out] LPRECT lprc
);
```

Parameters

[in] *hrgn*

A handle to the region.

[out] *lprc*

A pointer to a [RECT](#) structure that receives the bounding rectangle in logical units.

Return value

The return value specifies the region's complexity. It can be one of the following values:

VALUE	MEANING
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than a single rectangle.

If the *hrgn* parameter does not identify a valid region, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL

Gdi32.dll

See also

[RECT](#)

[Region Functions](#)

[Regions Overview](#)

GetROP2 function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetROP2** function retrieves the foreground mix mode of the specified device context. The mix mode specifies how the pen or interior color and the color already on the screen are combined to yield a new color.

Syntax

```
int GetROP2(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

Handle to the device context.

Return value

If the function succeeds, the return value specifies the foreground mix mode.

If the function fails, the return value is zero.

Remarks

Following are the foreground mix modes.

MIX MODE	DESCRIPTION
R2_BLACK	Pixel is always 0.
R2_COPYPEN	Pixel is the pen color.
R2_MASKNOTPEN	Pixel is a combination of the colors common to both the screen and the inverse of the pen.
R2_MASKPEN	Pixel is a combination of the colors common to both the pen and the screen.
R2_MASKPENNOT	Pixel is a combination of the colors common to both the pen and the inverse of the screen.
R2_MERGENOTPEN	Pixel is a combination of the screen color and the inverse of the pen color.
R2_MERGEPEN	Pixel is a combination of the pen color and the screen color.
R2_MERGEPPENNOT	Pixel is a combination of the pen color and the inverse of the screen color.

R2_NOP	Pixel remains unchanged.
R2_NOT	Pixel is the inverse of the screen color.
R2_NOTCOPYPEN	Pixel is the inverse of the pen color.
R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN color.
R2_NOTMERGESEN	Pixel is the inverse of the R2_MERGESEN color.
R2_NOTXORPEN	Pixel is the inverse of the R2_XORPEN color.
R2_WHITE	Pixel is always 1.
R2_XORPEN	Pixel is a combination of the colors in the pen and in the screen, but not in both.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[SetROP2](#)

GetRValue macro (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetRValue** macro retrieves an intensity value for the red component of a red, green, blue (RGB) value.

Syntax

```
void GetRValue(  
    rgb  
) ;
```

Parameters

rgb

Specifies an RGB color value.

Return value

None

Remarks

The intensity value is in the range 0 through 255.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[Color Macros](#)

[Colors Overview](#)

[GetBValue](#)

[GetGValue](#)

[PALETTEINDEX](#)

[PALETTERGB](#)

RGB

GetStockObject function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetStockObject** function retrieves a handle to one of the stock pens, brushes, fonts, or palettes.

Syntax

```
HGDIOBJ GetStockObject(  
    [in] int i  
)
```

Parameters

[in] **i**

The type of stock object. This parameter can be one of the following values.

VALUE	MEANING
BLACK_BRUSH	Black brush.
DKGRAY_BRUSH	Dark gray brush.
DC_BRUSH	Solid color brush. The default color is white. The color can be changed by using the SetDCBrushColor function. For more information, see the Remarks section.
GRAY_BRUSH	Gray brush.
HOLLOW_BRUSH	Hollow brush (equivalent to NULL_BRUSH).
LTGRAY_BRUSH	Light gray brush.
NULL_BRUSH	Null brush (equivalent to HOLLOW_BRUSH).
WHITE_BRUSH	White brush.
BLACK_PEN	Black pen.

DC_PEN	Solid pen color. The default color is white. The color can be changed by using the SetDCPenColor function. For more information, see the Remarks section.
NULL_PEN	Null pen. The null pen draws nothing.
WHITE_PEN	White pen.
ANSI_FIXED_FONT	Windows fixed-pitch (monospace) system font.
ANSI_VAR_FONT	Windows variable-pitch (proportional space) system font.
DEVICE_DEFAULT_FONT	Device-dependent font.
DEFAULT_GUI_FONT	<p>Default font for user interface objects such as menus and dialog boxes. It is not recommended that you use DEFAULT_GUI_FONT or SYSTEM_FONT to obtain the font used by dialogs and windows; for more information, see the remarks section.</p> <p>The default font is Tahoma.</p>
OEM_FIXED_FONT	Original equipment manufacturer (OEM) dependent fixed-pitch (monospace) font.
SYSTEM_FONT	<p>System font. By default, the system uses the system font to draw menus, dialog box controls, and text. It is not recommended that you use DEFAULT_GUI_FONT or SYSTEM_FONT to obtain the font used by dialogs and windows; for more information, see the remarks section.</p> <p>The default system font is Tahoma.</p>
SYSTEM_FIXED_FONT	Fixed-pitch (monospace) system font. This stock object is provided only for compatibility with 16-bit Windows versions earlier than 3.0.
DEFAULT_PALETTE	Default palette. This palette consists of the static colors in the system palette.

Return value

If the function succeeds, the return value is a handle to the requested logical object.

If the function fails, the return value is **NULL**.

Remarks

It is not recommended that you employ this method to obtain the current font used by dialogs and windows. Instead, use the [SystemParametersInfo](#) function with the SPI_GETNONCLIENTMETRICS parameter to retrieve the

current font. [SystemParametersInfo](#) will take into account the current theme and provides font information for captions, menus, and message dialogs.

Use the DKGRAY_BRUSH, GRAY_BRUSH, and LTGRAY_BRUSH stock objects only in windows with the CS_HREDRAW and CS_VREDRAW styles. Using a gray stock brush in any other style of window can lead to misalignment of brush patterns after a window is moved or sized. The origins of stock brushes cannot be adjusted.

The HOLLOW_BRUSH and NULL_BRUSH stock objects are equivalent.

It is not necessary (but it is not harmful) to delete stock objects by calling [DeleteObject](#).

Both DC_BRUSH and DC_PEN can be used interchangeably with other stock objects like BLACK_BRUSH and BLACK_PEN. For information on retrieving the current pen or brush color, see [GetDCBrushColor](#) and [GetDCPenColor](#). See [Setting the Pen or Brush Color](#) for an example of setting colors. The [GetStockObject](#) function with an argument of DC_BRUSH or DC_PEN can be used interchangeably with the [SetDCPenColor](#) and [SetDCBrushColor](#) functions.

Examples

For an example, see [Setting the Pen or Brush Color](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteObject](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[SelectObject](#)

GetStretchBltMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetStretchBltMode** function retrieves the current stretching mode. The stretching mode defines how color data is added to or removed from bitmaps that are stretched or compressed when the [StretchBlt](#) function is called.

Syntax

```
int GetStretchBltMode(  
    [in] HDC hdc  
>);
```

Parameters

[in] hdc

A handle to the device context.

Return value

If the function succeeds, the return value is the current stretching mode. This can be one of the following values.

VALUE	DESCRIPTION
BLACKONWHITE	Performs a Boolean AND operation using the color values for the eliminated and existing pixels. If the bitmap is a monochrome bitmap, this mode preserves black pixels at the expense of white pixels.
COLORONCOLOR	Deletes the pixels. This mode deletes all eliminated lines of pixels without trying to preserve their information.
HALFTONE	Maps pixels from the source rectangle into blocks of pixels in the destination rectangle. The average color over the destination block of pixels approximates the color of the source pixels.
STRETCH_ANDSCANS	Same as BLACKONWHITE.
STRETCH_DELETESCANS	Same as COLORONCOLOR.
STRETCH_HALFTONE	Same as HALFTONE.
STRETCH_ORSCANS	Same as WHITEONBLACK.
WHITEONBLACK	Performs a Boolean OR operation using the color values for the eliminated and existing pixels. If the bitmap is a monochrome bitmap, this mode preserves white pixels at the expense of black pixels.

If the function fails, the return value is zero.

Requirements

Requirements	
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[SetStretchBltMode](#)

GetSystemPaletteEntries function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetSystemPaletteEntries** function retrieves a range of palette entries from the system palette that is associated with the specified device context (DC).

Syntax

```
UINT GetSystemPaletteEntries(
    [in]    HDC      hdc,
    [in]    UINT     iStart,
    [in]    UINT     cEntries,
    [out]   LPPALETTEENTRY pPalEntries
);
```

Parameters

[in] hdc

A handle to the device context.

[in] iStart

The first entry to be retrieved from the system palette.

[in] cEntries

The number of entries to be retrieved from the system palette.

[out] pPalEntries

A pointer to an array of **PALETTEENTRY** structures to receive the palette entries. The array must contain at least as many structures as specified by the *cEntries* parameter. If this parameter is **NULL**, the function returns the total number of entries in the palette.

Return value

If the function succeeds, the return value is the number of entries retrieved from the palette.

If the function fails, the return value is zero.

Remarks

An application can determine whether a device supports palette operations by calling the [GetDeviceCaps](#) function and specifying the **RASTERCAPS** constant.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[GetDeviceCaps](#)

[GetPaletteEntries](#)

[PALETTEENTRY](#)

GetSystemPaletteUse function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetSystemPaletteUse** function retrieves the current state of the system (physical) palette for the specified device context (DC).

Syntax

```
UINT GetSystemPaletteUse(
    [in] HDC hdc
);
```

Parameters

[in] *hdc*

A handle to the device context.

Return value

If the function succeeds, the return value is the current state of the system palette. This parameter can be one of the following values.

VALUE	MEANING
SYSPAL_NOSTATIC	The system palette contains no static colors except black and white.
SYSPAL_STATIC	The system palette contains static colors that will not change when an application realizes its logical palette.
SYSPAL_ERROR	The given device context is invalid or does not support a color palette.

Remarks

By default, the system palette contains 20 static colors that are not changed when an application realizes its logical palette. An application can gain access to most of these colors by calling the [SetSystemPaletteUse](#) function.

The device context identified by the *hdc* parameter must represent a device that supports color palettes.

An application can determine whether a device supports color palettes by calling the [GetDeviceCaps](#) function and specifying the RASTERCAPS constant.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[GetDeviceCaps](#)

[SetSystemPaletteUse](#)

GetTextAlign function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextAlign** function retrieves the text-alignment setting for the specified device context.

Syntax

```
UINT GetTextAlign(
    [in] HDC hdc
);
```

Parameters

[in] hdc

A handle to the device context.

Return value

If the function succeeds, the return value is the status of the text-alignment flags. For more information about the return value, see the Remarks section. The return value is a combination of the following values.

VALUE	MEANING
TA_BASELINE	The reference point is on the base line of the text.
TA_BOTTOM	The reference point is on the bottom edge of the bounding rectangle.
TA_TOP	The reference point is on the top edge of the bounding rectangle.
TA_CENTER	The reference point is aligned horizontally with the center of the bounding rectangle.
TA_LEFT	The reference point is on the left edge of the bounding rectangle.
TA_RIGHT	The reference point is on the right edge of the bounding rectangle.
TA_RTLREADING	Middle East language edition of Windows: The text is laid out in right to left reading order, as opposed to the default left to right order. This only applies when the font selected into the device context is either Hebrew or Arabic.
TA_NOUPDATECP	The current position is not updated after each text output call.
TA_UPDATECP	The current position is updated after each text output call.

When the current font has a vertical default base line (as with Kanji), the following values are used instead of TA_BASELINE and TA_CENTER.

VALUE	MEANING
VTA_BASELINE	The reference point is on the base line of the text.
VTA_CENTER	The reference point is aligned vertically with the center of the bounding rectangle.

If the function fails, the return value is GDI_ERROR.

Remarks

The bounding rectangle is a rectangle bounding all of the character cells in a string of text. Its dimensions can be obtained by calling the [GetTextExtentPoint32](#) function.

The text-alignment flags determine how the [TextOut](#) and [ExtTextOut](#) functions align a string of text in relation to the string's reference point provided to [TextOut](#) or [ExtTextOut](#).

The text-alignment flags are not necessarily single bit flags and may be equal to zero. The flags must be examined in groups of related flags, as shown in the following list.

- TA_LEFT, TA_RIGHT, and TA_CENTER
- TA_BOTTOM, TA_TOP, and TA_BASELINE
- TA_NOUPDATECP and TA_UPDATECP

If the current font has a vertical default base line, the related flags are as shown in the following list.

- TA_LEFT, TA_RIGHT, and VTA_BASELINE
- TA_BOTTOM, TA_TOP, and VTA_CENTER
- TA_NOUPDATECP and TA_UPDATECP

To verify that a particular flag is set in the return value of this function:

1. Apply the bitwise OR operator to the flag and its related flags.
2. Apply the bitwise AND operator to the result and the return value.
3. Test for the equality of this result and the flag.

Examples

For an example, see [Setting the Text Alignment](#).

Requirements

Requirement	Description
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[SetTextAlign](#)

[TextOut](#)

GetTextCharacterExtra function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextCharacterExtra** function retrieves the current intercharacter spacing for the specified device context.

Syntax

```
int GetTextCharacterExtra(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

Handle to the device context.

Return value

If the function succeeds, the return value is the current intercharacter spacing, in logical coordinates.

If the function fails, the return value is 0x80000000.

Remarks

The intercharacter spacing defines the extra space, in logical units along the base line, that the [TextOut](#) or [ExtTextOut](#) functions add to each character as a line is written. The spacing is used to expand lines of text.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[SetTextCharacterExtra](#)

[TextOut](#)

GetTextColor function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextColor** function retrieves the current text color for the specified device context.

Syntax

```
COLORREF GetTextColor(  
    [in] HDC hdc  
>);
```

Parameters

[in] `hdc`

Handle to the device context.

Return value

If the function succeeds, the return value is the current text color as a [COLORREF](#) value.

If the function fails, the return value is `CLR_INVALID`. No extended error information is available.

Remarks

The text color defines the foreground color of characters drawn by using the [TextOut](#) or [ExtTextOut](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORREF](#)

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[SetTextColor](#)

[TextOut](#)

GetTextExtentExPointA function (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **GetTextExtentExPoint** function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters. (A text extent is the distance between the beginning of the space and a character that will fit in the space.) This information is useful for word-wrapping calculations.

Syntax

```
BOOL GetTextExtentExPointA(
    [in]  HDC      hdc,
    [in]  LPCSTR  lpszString,
    [in]  int     cchString,
    [in]  int     nMaxExtent,
    [out] LPINT   lpnFit,
    [out] LPINT   lpnDx,
    [out] LPSIZE  lpSize
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpszString`

A pointer to the null-terminated string for which extents are to be retrieved.

[in] `cchString`

The number of characters in the string pointed to by the `lpszStr` parameter. For an ANSI call it specifies the string length in bytes and for a Unicode it specifies the string length in WORDs. Note that for the ANSI function, characters in SBCS code pages take one byte each, while most characters in DBCS code pages take two bytes; for the Unicode function, most currently defined Unicode characters (those in the Basic Multilingual Plane (BMP)) are one WORD while Unicode surrogates are two WORDs.

[in] `nMaxExtent`

The maximum allowable width, in logical units, of the formatted string.

[out] `lpnFit`

A pointer to an integer that receives a count of the maximum number of characters that will fit in the space specified by the `nMaxExtent` parameter. When the `lpnFit` parameter is **NULL**, the `nMaxExtent` parameter is ignored.

[out] `lpnDx`

A pointer to an array of integers that receives partial string extents. Each element in the array gives the distance, in logical units, between the beginning of the string and one of the characters that fits in the space specified by the `nMaxExtent` parameter. This array must have at least as many elements as characters specified by the `cchString` parameter because the entire array is used internally. The function fills the array with valid extents for

as many characters as are specified by the *lpnFit* parameter. Any values in the rest of the array should be ignored. If *alpDx* is **NULL**, the function does not compute partial string widths.

For complex scripts, where a sequence of characters may be represented by any number of glyphs, the values in the *alpDx* array up to the number specified by the *lpnFit* parameter match one-to-one with code points. Again, you should ignore the rest of the values in the *alpDx* array.

[out] *lpSize*

A pointer to a [SIZE](#) structure that receives the dimensions of the string, in logical units. This parameter cannot be **NULL**.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

If both the *lpnFit* and *alpDx* parameters are **NULL**, calling the [GetTextExtentExPoint](#) function is equivalent to calling the [GetTextExtentPoint](#) function.

For the ANSI version of [GetTextExtentExPoint](#), the *lpDx* array has the same number of INT values as there are bytes in *lpString*. The INT values that correspond to the two bytes of a DBCS character are each the extent of the entire composite character.

Note, the *alpDx* values for [GetTextExtentExPoint](#) are not the same as the *lpDx* values for [ExtTextOut](#). To use the *alpDx* values in *lpDx*, you must first process them.

When this function returns the text extent, it assumes that the text is horizontal, that is, that the escapement is always 0. This is true for both the horizontal and vertical measurements of the text. Even if you use a font that specifies a nonzero escapement, this function doesn't use the angle while it computes the text extent. The app must convert it explicitly. However, when the graphics mode is set to [GM_ADVANCED](#) and the character orientation is 90 degrees from the print orientation, the values that this function return do not follow this rule. When the character orientation and the print orientation match for a given string, this function returns the dimensions of the string in the [SIZE](#) structure as { cx : 116, cy : 18 }. When the character orientation and the print orientation are 90 degrees apart for the same string, this function returns the dimensions of the string in the [SIZE](#) structure as { cx : 18, cy : 116 }.

This function returns the extent of each successive character in a string. When these are rounded to logical units, you get different results than what is returned from the [GetCharWidth](#), which returns the width of each individual character rounded to logical units.

NOTE

The wingdi.h header defines [GetTextExtentExPoint](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint](#)

[SIZE](#)

GetTextExtentExPointI function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextExtentExPointI** function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters. (A text extent is the distance between the beginning of the space and a character that will fit in the space.) This information is useful for word-wrapping calculations.

Syntax

```
BOOL GetTextExtentExPointI(
    [in]  HDC      hdc,
    [in]  LPWORD   lpwszString,
    [in]  int       cwchString,
    [in]  int       nMaxExtent,
    [out] LPINT    lpnFit,
    [out] LPINT    lpnDx,
    [out] LPSIZE   lpSize
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpwszString`

A pointer to an array of glyph indices for which extents are to be retrieved.

[in] `cwchString`

The number of glyphs in the array pointed to by the `pgi`/`n` parameter.

[in] `nMaxExtent`

The maximum allowable width, in logical units, of the formatted string.

[out] `lpnFit`

A pointer to an integer that receives a count of the maximum number of characters that will fit in the space specified by the `nMaxExtent` parameter. When the `lpnFit` parameter is **NULL**, the `nMaxExtent` parameter is ignored.

[out] `lpnDx`

A pointer to an array of integers that receives partial glyph extents. Each element in the array gives the distance, in logical units, between the beginning of the glyph indices array and one of the glyphs that fits in the space specified by the `nMaxExtent` parameter. Although this array should have at least as many elements as glyph indices specified by the `cgi` parameter, the function fills the array with extents only for as many glyph indices as are specified by the `lpnFit` parameter. If `lpnFit` is **NULL**, the function does not compute partial string widths.

[out] `lpSize`

A pointer to a **SIZE** structure that receives the dimensions of the glyph indices array, in logical units. This value

cannot be **NULL**.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

If both the *lpnFit* and *a/pDx* parameters are **NULL**, calling the **GetTextExtentExPointI** function is equivalent to calling the **GetTextExtentPointI** function.

When this function returns the text extent, it assumes that the text is horizontal, that is, that the escapement is always 0. This is true for both the horizontal and vertical measurements of the text. Even if you use a font that specifies a nonzero escapement, this function doesn't use the angle while it computes the text extent. The app must convert it explicitly. However, when the graphics mode is set to **GM_ADVANCED** and the character orientation is 90 degrees from the print orientation, the values that this function return do not follow this rule. When the character orientation and the print orientation match for a given string, this function returns the dimensions of the string in the **SIZE** structure as { cx : 116, cy : 18 }. When the character orientation and the print orientation are 90 degrees apart for the same string, this function returns the dimensions of the string in the **SIZE** structure as { cx : 18, cy : 116 }.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint](#)

[SIZE](#)

GetTextExtentExPointW function (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **GetTextExtentExPoint** function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters. (A text extent is the distance between the beginning of the space and a character that will fit in the space.) This information is useful for word-wrapping calculations.

Syntax

```
BOOL GetTextExtentExPointW(
    [in] HDC      hdc,
    [in] LPCWSTR lpszString,
    [in] int      cchString,
    [in] int      nMaxExtent,
    [out] LPINT   lpnFit,
    [out] LPINT   lpnDx,
    [out] LPSIZE  lpSize
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpszString`

A pointer to the null-terminated string for which extents are to be retrieved.

[in] `cchString`

The number of characters in the string pointed to by the `/pszStr` parameter. For an ANSI call it specifies the string length in bytes and for a Unicode it specifies the string length in WORDs. Note that for the ANSI function, characters in SBCS code pages take one byte each, while most characters in DBCS code pages take two bytes; for the Unicode function, most currently defined Unicode characters (those in the Basic Multilingual Plane (BMP)) are one WORD while Unicode surrogates are two WORDs.

[in] `nMaxExtent`

The maximum allowable width, in logical units, of the formatted string.

[out] `lpnFit`

A pointer to an integer that receives a count of the maximum number of characters that will fit in the space specified by the `nMaxExtent` parameter. When the `/pnFit` parameter is **NULL**, the `nMaxExtent` parameter is ignored.

[out] `lpnDx`

A pointer to an array of integers that receives partial string extents. Each element in the array gives the distance, in logical units, between the beginning of the string and one of the characters that fits in the space specified by the `nMaxExtent` parameter. This array must have at least as many elements as characters specified by the `cchString` parameter because the entire array is used internally. The function fills the array with valid extents for

as many characters as are specified by the *lpnFit* parameter. Any values in the rest of the array should be ignored. If *alpDx* is **NULL**, the function does not compute partial string widths.

For complex scripts, where a sequence of characters may be represented by any number of glyphs, the values in the *alpDx* array up to the number specified by the *lpnFit* parameter match one-to-one with code points. Again, you should ignore the rest of the values in the *alpDx* array.

[out] *lpSize*

A pointer to a [SIZE](#) structure that receives the dimensions of the string, in logical units. This parameter cannot be **NULL**.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

If both the *lpnFit* and *alpDx* parameters are **NULL**, calling the [GetTextExtentExPoint](#) function is equivalent to calling the [GetTextExtentPoint](#) function.

For the ANSI version of [GetTextExtentExPoint](#), the *lpDx* array has the same number of INT values as there are bytes in *lpString*. The INT values that correspond to the two bytes of a DBCS character are each the extent of the entire composite character.

Note, the *alpDx* values for [GetTextExtentExPoint](#) are not the same as the *lpDx* values for [ExtTextOut](#). To use the *alpDx* values in *lpDx*, you must first process them.

When this function returns the text extent, it assumes that the text is horizontal, that is, that the escapement is always 0. This is true for both the horizontal and vertical measurements of the text. Even if you use a font that specifies a nonzero escapement, this function doesn't use the angle while it computes the text extent. The app must convert it explicitly. However, when the graphics mode is set to [GM_ADVANCED](#) and the character orientation is 90 degrees from the print orientation, the values that this function return do not follow this rule. When the character orientation and the print orientation match for a given string, this function returns the dimensions of the string in the [SIZE](#) structure as { cx : 116, cy : 18 }. When the character orientation and the print orientation are 90 degrees apart for the same string, this function returns the dimensions of the string in the [SIZE](#) structure as { cx : 18, cy : 116 }.

This function returns the extent of each successive character in a string. When these are rounded to logical units, you get different results than what is returned from the [GetCharWidth](#), which returns the width of each individual character rounded to logical units.

NOTE

The wingdi.h header defines [GetTextExtentExPoint](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint](#)

[SIZE](#)

GetTextExtentPoint32A function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **GetTextExtentPoint32** function computes the width and height of the specified string of text.

Syntax

```
BOOL GetTextExtentPoint32A(
    [in]  HDC      hdc,
    [in]  LPCSTR  lpString,
    [in]  int       c,
    [out] LPSIZE  psizl
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpString`

A pointer to a buffer that specifies the text string. The string does not need to be null-terminated, because the `c` parameter specifies the length of the string.

[in] `c`

The [length of the string](#) pointed to by `lpString`.

[out] `psizl`

A pointer to a [SIZE](#) structure that receives the dimensions of the string, in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **GetTextExtentPoint32** function uses the currently selected font to compute the dimensions of the string. The width and height, in logical units, are computed without considering any clipping.

Because some devices kern characters, the sum of the extents of the characters in a string may not be equal to the extent of the string.

The calculated string width takes into account the intercharacter spacing set by the [SetTextCharacterExtra](#) function and the justification set by [SetTextJustification](#). This is true for both displaying on a screen and for printing. However, if `/pDx` is set in [ExtTextOut](#), **GetTextExtentPoint32** does not take into account either intercharacter spacing or justification. In addition, for EMF, the print result always takes both intercharacter spacing and justification into account.

When dealing with text displayed on a screen, the calculated string width takes into account the intercharacter

spacing set by the [SetTextCharacterExtra](#) function and the justification set by [SetTextJustification](#). However, if *lpDx* is set in [ExtTextOut](#), [GetTextExtentPoint32](#) does not take into account either intercharacter spacing or justification. However, when printing with EMF:

- The print result ignores intercharacter spacing, although [GetTextExtentPoint32](#) takes it into account.
- The print result takes justification into account, although [GetTextExtentPoint32](#) ignores it.

When this function returns the text extent, it assumes that the text is horizontal, that is, that the escapement is always 0. This is true for both the horizontal and vertical measurements of the text. Even if you use a font that specifies a nonzero escapement, this function doesn't use the angle while it computes the text extent. The app must convert it explicitly. However, when the graphics mode is set to [GM_ADVANCED](#) and the character orientation is 90 degrees from the print orientation, the values that this function return do not follow this rule. When the character orientation and the print orientation match for a given string, this function returns the dimensions of the string in the [SIZE](#) structure as { cx : 116, cy : 18 }. When the character orientation and the print orientation are 90 degrees apart for the same string, this function returns the dimensions of the string in the [SIZE](#) structure as { cx : 18, cy : 116 }.

[GetTextExtentPoint32](#) doesn't consider "\n" (new line) or "\r\n" (carriage return and new line) characters when it computes the height of a text string.

Examples

For an example, see [Drawing Text from Different Fonts on the Same Line](#).

NOTE

The wingdi.h header defines [GetTextExtentPoint32](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[SIZE](#)

[SetTextCharacterExtra](#)

[SetTextJustification](#)

GetTextExtentPoint32W function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **GetTextExtentPoint32** function computes the width and height of the specified string of text.

Syntax

```
BOOL GetTextExtentPoint32W(
    [in]  HDC      hdc,
    [in]  LPCWSTR lpString,
    [in]  int       c,
    [out] LPSIZE   psizl
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpString`

A pointer to a buffer that specifies the text string. The string does not need to be null-terminated, because the `c` parameter specifies the length of the string.

[in] `c`

The [length of the string](#) pointed to by `lpString`.

[out] `psizl`

A pointer to a [SIZE](#) structure that receives the dimensions of the string, in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **GetTextExtentPoint32** function uses the currently selected font to compute the dimensions of the string. The width and height, in logical units, are computed without considering any clipping.

Because some devices kern characters, the sum of the extents of the characters in a string may not be equal to the extent of the string.

The calculated string width takes into account the intercharacter spacing set by the [SetTextCharacterExtra](#) function and the justification set by [SetTextJustification](#). This is true for both displaying on a screen and for printing. However, if `/pDx` is set in [ExtTextOut](#), **GetTextExtentPoint32** does not take into account either intercharacter spacing or justification. In addition, for EMF, the print result always takes both intercharacter spacing and justification into account.

When dealing with text displayed on a screen, the calculated string width takes into account the intercharacter

spacing set by the [SetTextCharacterExtra](#) function and the justification set by [SetTextJustification](#). However, if *lpDx* is set in [ExtTextOut](#), [GetTextExtentPoint32](#) does not take into account either intercharacter spacing or justification. However, when printing with EMF:

- The print result ignores intercharacter spacing, although [GetTextExtentPoint32](#) takes it into account.
- The print result takes justification into account, although [GetTextExtentPoint32](#) ignores it.

When this function returns the text extent, it assumes that the text is horizontal, that is, that the escapement is always 0. This is true for both the horizontal and vertical measurements of the text. Even if you use a font that specifies a nonzero escapement, this function doesn't use the angle while it computes the text extent. The app must convert it explicitly. However, when the graphics mode is set to [GM_ADVANCED](#) and the character orientation is 90 degrees from the print orientation, the values that this function return do not follow this rule. When the character orientation and the print orientation match for a given string, this function returns the dimensions of the string in the [SIZE](#) structure as { cx : 116, cy : 18 }. When the character orientation and the print orientation are 90 degrees apart for the same string, this function returns the dimensions of the string in the [SIZE](#) structure as { cx : 18, cy : 116 }.

[GetTextExtentPoint32](#) doesn't consider "\n" (new line) or "\r\n" (carriage return and new line) characters when it computes the height of a text string.

Examples

For an example, see [Drawing Text from Different Fonts on the Same Line](#).

NOTE

The wingdi.h header defines [GetTextExtentPoint32](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[SIZE](#)

[SetTextCharacterExtra](#)

[SetTextJustification](#)

GetTextExtentPointA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextExtentPoint** function computes the width and height of the specified string of text.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should call the [GetTextExtentPoint32](#) function, which provides more accurate results.

Syntax

```
BOOL GetTextExtentPointA(
    [in]  HDC      hdc,
    [in]  LPCSTR  lpString,
    [in]  int      c,
    [out] LPSIZE  lpsz
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpString`

A pointer to the string that specifies the text. The string does not need to be zero-terminated, since *cbString* specifies the length of the string.

[in] `c`

The [length of the string](#) pointed to by *lpString*.

[out] `lpsz`

A pointer to a [SIZE](#) structure that receives the dimensions of the string, in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **GetTextExtentPoint** function uses the currently selected font to compute the dimensions of the string. The width and height, in logical units, are computed without considering any clipping. Also, this function assumes that the text is horizontal, that is, that the escapement is always 0. This is true for both the horizontal and vertical measurements of the text. Even if using a font specifying a nonzero escapement, this function will not use the angle while computing the text extent. The application must convert it explicitly.

Because some devices kern characters, the sum of the extents of the characters in a string may not be equal to

the extent of the string.

The calculated string width takes into account the intercharacter spacing set by the [SetTextCharacterExtra](#) function.

NOTE

The wingdi.h header defines GetTextExtentPoint as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[SIZE](#)

[SetTextCharacterExtra](#)

GetTextExtentPointI function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextExtentPointI** function computes the width and height of the specified array of glyph indices.

Syntax

```
BOOL GetTextExtentPointI(
    [in]  HDC     hdc,
    [in]  LPWORD pgiIn,
    [in]  int     cgi,
    [out] LPSIZE psize
);
```

Parameters

[in] `hdc`

Handle to the device context.

[in] `pgiIn`

Pointer to array of glyph indices.

[in] `cgi`

Specifies the number of glyph indices.

[out] `psize`

Pointer to a `SIZE` structure that receives the dimensions of the string, in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **GetTextExtentPointI** function uses the currently selected font to compute the dimensions of the array of glyph indices. The width and height, in logical units, are computed without considering any clipping.

When this function returns the text extent, it assumes that the text is horizontal, that is, that the escapement is always 0. This is true for both the horizontal and vertical measurements of the text. Even if you use a font that specifies a nonzero escapement, this function doesn't use the angle while it computes the text extent. The app must convert it explicitly. However, when the graphics mode is set to `GM_ADVANCED` and the character orientation is 90 degrees from the print orientation, the values that this function return do not follow this rule. When the character orientation and the print orientation match for a given string, this function returns the dimensions of the string in the `SIZE` structure as { cx : 116, cy : 18 }. When the character orientation and the print orientation are 90 degrees apart for the same string, this function returns the dimensions of the string in the `SIZE` structure as { cx : 18, cy : 116 }.

Because some devices kern characters, the sum of the extents of the individual glyph indices may not be equal to the extent of the entire array of glyph indices.

The calculated string width takes into account the intercharacter spacing set by the [SetTextCharacterExtra](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint](#)

[GetTextExtentPoint32](#)

[SIZE](#)

[SetTextCharacterExtra](#)

GetTextExtentPointW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextExtentPoint** function computes the width and height of the specified string of text.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should call the [GetTextExtentPoint32](#) function, which provides more accurate results.

Syntax

```
BOOL GetTextExtentPointW(
    [in]  HDC      hdc,
    [in]  LPCWSTR lpString,
    [in]  int      c,
    [out] LPSIZE  lpsz
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpString`

A pointer to the string that specifies the text. The string does not need to be zero-terminated, since *cbString* specifies the length of the string.

[in] `c`

The [length of the string](#) pointed to by *lpString*.

[out] `lpsz`

A pointer to a [SIZE](#) structure that receives the dimensions of the string, in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **GetTextExtentPoint** function uses the currently selected font to compute the dimensions of the string. The width and height, in logical units, are computed without considering any clipping. Also, this function assumes that the text is horizontal, that is, that the escapement is always 0. This is true for both the horizontal and vertical measurements of the text. Even if using a font specifying a nonzero escapement, this function will not use the angle while computing the text extent. The application must convert it explicitly.

Because some devices kern characters, the sum of the extents of the characters in a string may not be equal to

the extent of the string.

The calculated string width takes into account the intercharacter spacing set by the [SetTextCharacterExtra](#) function.

NOTE

The wingdi.h header defines GetTextExtentPoint as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[SIZE](#)

[SetTextCharacterExtra](#)

GetTextFaceA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextFace** function retrieves the typeface name of the font that is selected into the specified device context.

Syntax

```
int GetTextFaceA(
    [in]  HDC   hdc,
    [in]  int   c,
    [out] LPSTR lpName
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `c`

The length of the buffer pointed to by `lpFaceName`. For the ANSI function it is a BYTE count and for the Unicode function it is a WORD count. Note that for the ANSI function, characters in SBCS code pages take one byte each, while most characters in DBCS code pages take two bytes; for the Unicode function, most currently defined Unicode characters (those in the Basic Multilingual Plane (BMP)) are one WORD while Unicode surrogates are two WORDs.

[out] `lpName`

A pointer to the buffer that receives the typeface name. If this parameter is **NULL**, the function returns the number of characters in the name, including the terminating null character.

Return value

If the function succeeds, the return value is the number of characters copied to the buffer.

If the function fails, the return value is zero.

Remarks

The typeface name is copied as a null-terminated character string.

If the name is longer than the number of characters specified by the `nCount` parameter, the name is truncated.

NOTE

The wingdi.h header defines `GetTextFace` as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the `UNICODE` preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[GetTextColor](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

GetTextFaceW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextFace** function retrieves the typeface name of the font that is selected into the specified device context.

Syntax

```
int GetTextFaceW(
    [in]  HDC      hdc,
    [in]  int      c,
    [out] LPWSTR  lpName
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `c`

The length of the buffer pointed to by `lpFaceName`. For the ANSI function it is a BYTE count and for the Unicode function it is a WORD count. Note that for the ANSI function, characters in SBCS code pages take one byte each, while most characters in DBCS code pages take two bytes; for the Unicode function, most currently defined Unicode characters (those in the Basic Multilingual Plane (BMP)) are one WORD while Unicode surrogates are two WORDs.

[out] `lpName`

A pointer to the buffer that receives the typeface name. If this parameter is **NULL**, the function returns the number of characters in the name, including the terminating null character.

Return value

If the function succeeds, the return value is the number of characters copied to the buffer.

If the function fails, the return value is zero.

Remarks

The typeface name is copied as a null-terminated character string.

If the name is longer than the number of characters specified by the `nCount` parameter, the name is truncated.

NOTE

The wingdi.h header defines `GetTextFace` as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the `UNICODE` preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[GetTextColor](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

GetTextMetrics function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextMetrics** function fills the specified buffer with the metrics for the currently selected font.

Syntax

```
BOOL GetTextMetrics(
    [in]    HDC      hdc,
    [out]   LPTEXTMETRIC lptm
);
```

Parameters

[in] hdc

A handle to the device context.

[out] lptm

A pointer to the **TEXTMETRIC** structure that receives the text metrics.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

To determine whether a font is a TrueType font, first select it into a DC, then call **GetTextMetrics**, and then check for **TMPF_TRUETYPE** in **TEXTMETRIC.tmPitchAndFamily**. Note that **GetDC** returns an uninitialized DC, which has "System" (a bitmap font) as the default font; thus the need to select a font into the DC.

Examples

For an example, see "Displaying Keyboard Input" in [Using Keyboard Input](#) or [Drawing Text from Different Fonts on the Same Line](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[GetTextExtentPoint32](#)

[GetTextFace](#)

[SetTextJustification](#)

[TEXTMETRIC](#)

GetTextMetricsA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextMetrics** function fills the specified buffer with the metrics for the currently selected font.

Syntax

```
BOOL GetTextMetricsA(
    [in]    HDC           hdc,
    [out]   LPTEXTMETRICA lptm
);
```

Parameters

[in] hdc

A handle to the device context.

[out] lptm

A pointer to the [TEXTMETRIC](#) structure that receives the text metrics.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

To determine whether a font is a TrueType font, first select it into a DC, then call **GetTextMetrics**, and then check for TMPF_TRUETYPE in TEXTMETRIC.tmPitchAndFamily. Note that [GetDC](#) returns an uninitialized DC, which has "System" (a bitmap font) as the default font; thus the need to select a font into the DC.

Examples

For an example, see "Displaying Keyboard Input" in [Using Keyboard Input](#) or [Drawing Text from Different Fonts on the Same Line](#).

NOTE

The wingdi.h header defines **GetTextMetrics** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[GetTextExtentPoint32](#)

[GetTextFace](#)

[SetTextJustification](#)

[TEXTMETRIC](#)

GetTextMetricsW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTextMetrics** function fills the specified buffer with the metrics for the currently selected font.

Syntax

```
BOOL GetTextMetricsW(
    [in]    HDC           hdc,
    [out]   LPTEXTMETRICW lptm
);
```

Parameters

[in] hdc

A handle to the device context.

[out] lptm

A pointer to the **TEXTMETRIC** structure that receives the text metrics.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

To determine whether a font is a TrueType font, first select it into a DC, then call **GetTextMetrics**, and then check for **TMPF_TRUETYPE** in **TEXTMETRIC.tmPitchAndFamily**. Note that **GetDC** returns an uninitialized DC, which has "System" (a bitmap font) as the default font; thus the need to select a font into the DC.

Examples

For an example, see "Displaying Keyboard Input" in [Using Keyboard Input](#) or [Drawing Text from Different Fonts on the Same Line](#).

NOTE

The wingdi.h header defines **GetTextMetrics** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[GetTextExtentPoint32](#)

[GetTextFace](#)

[SetTextJustification](#)

[TEXTMETRIC](#)

GetViewportExtEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetViewportExtEx** function retrieves the x-extent and y-extent of the current viewport for the specified device context.

Syntax

```
BOOL GetViewportExtEx(
    [in]  HDC      hdc,
    [out] LPSIZE  lpsize
);
```

Parameters

[in] hdc

A handle to the device context.

[out] lpsize

A pointer to a [SIZE](#) structure that receives the x- and y-extents, in device units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetWindowExtEx](#)

[SetViewportExtEx](#)

[SetWindowExtEx](#)

GetViewportOrgEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetViewportOrgEx** function retrieves the x-coordinates and y-coordinates of the viewport origin for the specified device context.

Syntax

```
BOOL GetViewportOrgEx(
    [in]  HDC      hdc,
    [out] LPPOINT lppoint
);
```

Parameters

[in] hdc

A handle to the device context.

[out] lppoint

A pointer to a **POINT** structure that receives the coordinates of the origin, in device units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetWindowOrgEx](#)

POINT

[SetViewportOrgEx](#)

[SetWindowOrgEx](#)

GetWindowExtEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

This function retrieves the x-extent and y-extent of the window for the specified device context.

Syntax

```
BOOL GetWindowExtEx(
    [in]    HDC     hdc,
    [out]   LPSIZE lpsize
);
```

Parameters

[in] hdc

A handle to the device context.

[out] lpsize

A pointer to a **SIZE** structure that receives the x- and y-extents in page-space units, that is, logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetViewportExtEx](#)

[SetViewportExtEx](#)

[SetWindowExtEx](#)

GetWindowOrgEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetWindowOrgEx** function retrieves the x-coordinates and y-coordinates of the window origin for the specified device context.

Syntax

```
BOOL GetWindowOrgEx(
    [in]  HDC     hdc,
    [out] LPPOINT lppoint
);
```

Parameters

[in] `hdc`

A handle to the device context.

[out] `lppoint`

A pointer to a [POINT](#) structure that receives the coordinates, in logical units, of the window origin.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetViewportOrgEx](#)

[SetViewportOrgEx](#)

[SetWindowOrgEx](#)

GetWinMetaFileBits function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetWinMetaFileBits** function converts the enhanced-format records from a metafile into Windows-format records and stores the converted records in the specified buffer.

Syntax

```
UINT GetWinMetaFileBits(
    [in] HENHMETAFILE hemf,
    [in] UINT         cbData16,
    [out] LPBYTE      pData16,
    [in] INT          iMapMode,
    [in] HDC          hdcRef
);
```

Parameters

[in] `hemf`

A handle to the enhanced metafile.

[in] `cbData16`

The size, in bytes, of the buffer into which the converted records are to be copied.

[out] `pData16`

A pointer to the buffer that receives the converted records. If *lpbBuffer* is **NULL**, **GetWinMetaFileBits** returns the number of bytes required to store the converted metafile records.

[in] `iMapMode`

The mapping mode to use in the converted metafile.

[in] `hdcRef`

A handle to the reference device context.

Return value

If the function succeeds and the buffer pointer is **NULL**, the return value is the number of bytes required to store the converted records; if the function succeeds and the buffer pointer is a valid pointer, the return value is the size of the metafile data in bytes.

If the function fails, the return value is zero.

Remarks

This function converts an enhanced metafile into a Windows-format metafile so that its picture can be displayed in an application that recognizes the older format.

The system uses the reference device context to determine the resolution of the converted metafile.

The **GetWinMetaFileBits** function does not invalidate the enhanced metafile handle. An application should call the [DeleteEnhMetaFile](#) function to release the handle when it is no longer needed.

To create a scalable Windows-format metafile, specify MM_ANISOTROPIC as the *fnMapMode* parameter.

The upper-left corner of the metafile picture is always mapped to the origin of the reference device.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SetMapMode](#)

[SetWinMetaFileBits](#)

GetWorldTransform function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetWorldTransform** function retrieves the current world-space to page-space transformation.

Syntax

```
BOOL GetWorldTransform(  
    [in]  HDC      hdc,  
    [out] LPXFORM lpxf  
)
```

Parameters

[in] hdc

A handle to the device context.

[out] lpxf

A pointer to an **XFORM** structure that receives the current world-space to page-space transformation.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The precision of the transformation may be altered if an application calls the **ModifyWorldTransform** function prior to calling **GetWorldTransform**. (This is because the internal format for storing transformation values uses a higher precision than a **FLOAT** value.)

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[ModifyWorldTransform](#)

[SetWorldTransform](#)

GLYPHMETRICS structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GLYPHMETRICS** structure contains information about the placement and orientation of a glyph in a character cell.

Syntax

```
typedef struct _GLYPHMETRICS {
    UINT gmBlackBoxX;
    UINT gmBlackBoxY;
    POINT gmptGlyphOrigin;
    short gmCellIncX;
    short gmCellIncY;
} GLYPHMETRICS, *LPGLYPHMETRICS;
```

Members

gmBlackBoxX

The width of the smallest rectangle that completely encloses the glyph (its black box).

gmBlackBoxY

The height of the smallest rectangle that completely encloses the glyph (its black box).

gmptGlyphOrigin

The x- and y-coordinates of the upper left corner of the smallest rectangle that completely encloses the glyph.

gmCellIncX

The horizontal distance from the origin of the current character cell to the origin of the next character cell.

gmCellIncY

The vertical distance from the origin of the current character cell to the origin of the next character cell.

Remarks

Values in the **GLYPHMETRICS** structure are specified in device units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetGlyphOutline](#)

GLYPHSET structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The GLYPHSET structure contains information about a range of Unicode code points.

Syntax

```
typedef struct tagGLYPHSET {  
    DWORD    cbThis;  
    DWORD    f1Accel;  
    DWORD    cGlyphsSupported;  
    DWORD    cRanges;  
    WCRANGE ranges[1];  
} GLYPHSET, *PGLYPHSET, *LPGLYPHSET;
```

Members

cbThis

The size, in bytes, of this structure.

f1Accel

Flags describing the maximum size of the glyph indices. This member can be the following value.

VALUE	MEANING
GS_8BIT_INDICES	Treat glyph indices as 8-bit wide values. Otherwise, they are 16-bit wide values.

cGlyphsSupported

The total number of Unicode code points supported in the font.

cRanges

The total number of Unicode ranges in **ranges**.

ranges

Array of Unicode ranges that are supported in the font.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetFontUnicodeRanges](#)

[WCRANGE](#)

GOBJENUMPROC callback function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The `EnumObjectsProc` function is an application-defined callback function used with the [EnumObjects](#) function. It is used to process the object data. The `GOBJENUMPROC` type defines a pointer to this callback function. `EnumObjectsProc` is a placeholder for the application-defined function name.

Syntax

```
GOBJENUMPROC Gobjenumproc;

int Gobjenumproc(
    LPVOID unnamedParam1,
    LPARAM unnamedParam2
)
{...}
```

Parameters

unnamedParam1

unnamedParam2

Return value

To continue enumeration, the callback function must return a nonzero value. This value is user-defined.

To stop enumeration, the callback function must return zero.

Remarks

An application must register this function by passing its address to the [EnumObjects](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumObjects](#)

[GlobalAlloc](#)

[GlobalLock](#)

[LOGBRUSH](#)

[LOGOPEN](#)

GRADIENT_RECT structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GRADIENT_RECT** structure specifies the index of two vertices in the *pVertex* array in the **GradientFill** function. These two vertices form the upper-left and lower-right boundaries of a rectangle.

Syntax

```
typedef struct _GRADIENT_RECT {  
    ULONG UpperLeft;  
    ULONG LowerRight;  
} GRADIENT_RECT, *PGRADIENT_RECT, *LPGRADIENT_RECT;
```

Members

UpperLeft

The upper-left corner of a rectangle.

LowerRight

The lower-right corner of a rectangle.

Remarks

The **GRADIENT_RECT** structure specifies the values of the *pVertex* array that are used when the *dwMode* parameter of the **GradientFill** function is **GRADIENT_FILL_RECT_H** or **GRADIENT_FILL_RECT_V**. For related **GradientFill** structures, see **GRADIENT_TRIANGLE** and **TRIVERTEX**.

The following images shows examples of a rectangle with a gradient fill - one in horizontal mode, the other in vertical mode.



Examples

For an example, see [Drawing a Shaded Rectangle](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Bitmap Structures](#)

[Bitmaps Overview](#)

[GRADIENT_TRIANGLE](#)

[GradientFill](#)

[TRIVERTEX](#)

GRADIENT_TRIANGLE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GRADIENT_TRIANGLE** structure specifies the index of three vertices in the *pVertex* array in the **GradientFill** function. These three vertices form one triangle.

Syntax

```
typedef struct _GRADIENT_TRIANGLE {  
    ULONG Vertex1;  
    ULONG Vertex2;  
    ULONG Vertex3;  
} GRADIENT_TRIANGLE, *PGRADIENT_TRIANGLE, *LPGRADIENT_TRIANGLE;
```

Members

Vertex1

The first point of the triangle where sides intersect.

Vertex2

The second point of the triangle where sides intersect.

Vertex3

The third point of the triangle where sides intersect.

Remarks

The **GRADIENT_TRIANGLE** structure specifies the values in the *pVertex* array that are used when the *dwMode* parameter of the **GradientFill** function is **GRADIENT_FILL_TRIANGLE**. For related **GradientFill** structures, see **GRADIENT_RECT** and **TRIVERTEX**.

The following image shows an example of a triangle with a gradient fill.



Examples

For an example, see [Drawing a Shaded Triangle](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Bitmap Structures](#)

[Bitmaps Overview](#)

[GRADIENT_RECT](#)

[GradientFill](#)

[TRIVERTEX](#)

GradientFill function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GradientFill** function fills rectangle and triangle structures.

Syntax

```
BOOL GradientFill(
    [in] HDC         hdc,
    [in] PTRIVERTEX pVertex,
    [in] ULONG       nVertex,
    [in] PVOID       pMesh,
    [in] ULONG       nMesh,
    [in] ULONG       ulMode
);
```

Parameters

[in] *hdc*

A handle to the destination device context.

[in] *pVertex*

A pointer to an array of [TRIVERTEX](#) structures that each define a vertex.

[in] *nVertex*

The number of vertices in *pVertex*.

[in] *pMesh*

An array of [GRADIENT_TRIANGLE](#) structures in triangle mode, or an array of [GRADIENT_RECT](#) structures in rectangle mode.

[in] *nMesh*

The number of elements (triangles or rectangles) in *pMesh*.

[in] *ulMode*

The gradient fill mode. This parameter can be one of the following values.

VALUE	MEANING
<code>GRADIENT_FILL_RECT_H</code>	In this mode, two endpoints describe a rectangle. The rectangle is defined to have a constant color (specified by the TRIVERTEX structure) for the left and right edges. GDI interpolates the color from the left to right edge and fills the interior.

GRADIENT_FILL_RECT_V	In this mode, two endpoints describe a rectangle. The rectangle is defined to have a constant color (specified by the TRIVERTEX structure) for the top and bottom edges. GDI interpolates the color from the top to bottom edge and fills the interior.
GRADIENT_FILL_TRIANGLE	In this mode, an array of TRIVERTEX structures is passed to GDI along with a list of array indexes that describe separate triangles. GDI performs linear interpolation between triangle vertices and fills the interior. Drawing is done directly in 24- and 32-bpp modes. Dithering is performed in 16-, 8-, 4-, and 1-bpp mode.

Return value

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**.

Remarks

To add smooth shading to a triangle, call the [GradientFill](#) function with the three triangle endpoints. GDI will linearly interpolate and fill the triangle. Here is the drawing output of a shaded triangle.



To add smooth shading to a rectangle, call [GradientFill](#) with the upper-left and lower-right coordinates of the rectangle. There are two shading modes used when drawing a rectangle. In horizontal mode, the rectangle is shaded from left-to-right. In vertical mode, the rectangle is shaded from top-to-bottom. Here is the drawing output of two shaded rectangles - one in horizontal mode, the other in



vertical mode:

The

[GradientFill](#) function uses a mesh method to specify the endpoints of the object to draw. All vertices are passed to [GradientFill](#) in the *pVertex* array. The *pMesh* parameter specifies how these vertices are connected to form an object. When filling a rectangle, *pMesh* points to an array of [GRADIENT_RECT](#) structures. Each [GRADIENT_RECT](#) structure specifies the index of two vertices in the *pVertex* array. These two vertices form the upper-left and lower-right boundary of one rectangle.

In the case of filling a triangle, *pMesh* points to an array of [GRADIENT_TRIANGLE](#) structures. Each [GRADIENT_TRIANGLE](#) structure specifies the index of three vertices in the *pVertex* array. These three vertices form one triangle.

To simplify hardware acceleration, this routine is not required to be pixel-perfect in the triangle interior.

Note that [GradientFill](#) does not use the Alpha member of the [TRIVERTEX](#) structure. To use [GradientFill](#) with transparency, call [GradientFill](#) and then call [AlphaBlend](#) with the desired values for the alpha channel of each vertex.

For more information, see [Smooth Shading](#), [Drawing a Shaded Triangle](#), and [Drawing a Shaded Rectangle](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Msimg32.lib
DLL	Msimg32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[EMRGRADIENTFILL](#)

[GRADIENT_RECT](#)

[GRADIENT_TRIANGLE](#)

[TRIVERTEX](#)

HANDLETABLE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **HANDLETABLE** structure is an array of handles, each of which identifies a graphics device interface (GDI) object.

Syntax

```
typedef struct tagHANDLETABLE {  
    HGDIOBJ objectHandle[1];  
} HANDLETABLE, *PHANDLETABLE, *LPHANDLETABLE;
```

Members

objectHandle

An array of handles.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EnhMetaFileProc](#)

[EnumMetaFileProc](#)

[Metafile Structures](#)

[Metafiles Overview](#)

IntersectClipRect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **IntersectClipRect** function creates a new clipping region from the intersection of the current clipping region and the specified rectangle.

Syntax

```
int IntersectClipRect(
    [in] HDC hdc,
    [in] int left,
    [in] int top,
    [in] int right,
    [in] int bottom
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `left`

The x-coordinate, in logical units, of the upper-left corner of the rectangle.

[in] `top`

The y-coordinate, in logical units, of the upper-left corner of the rectangle.

[in] `right`

The x-coordinate, in logical units, of the lower-right corner of the rectangle.

[in] `bottom`

The y-coordinate, in logical units, of the lower-right corner of the rectangle.

Return value

The return value specifies the new clipping region's type and can be one of the following values.

RETURN CODE	DESCRIPTION
<code>NULLREGION</code>	Region is empty.
<code>SIMPLEREGION</code>	Region is a single rectangle.
<code>COMPLEXREGION</code>	Region is more than one rectangle.

ERROR	An error occurred. (The current clipping region is unaffected.)
-------	---

Remarks

The lower and right-most edges of the given rectangle are excluded from the clipping region.

If a clipping region does not already exist then the system may apply a default clipping region to the specified HDC. A clipping region is then created from the intersection of that default clipping region and the rectangle specified in the function parameters.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Clipping Functions](#)

[Clipping Overview](#)

[ExcludeClipRect](#)

InvertRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **InvertRgn** function inverts the colors in the specified region.

Syntax

```
BOOL InvertRgn(
    [in] HDC hdc,
    [in] HRGN hrgn
);
```

Parameters

[in] **hdc**

Handle to the device context.

[in] **hrgn**

Handle to the region for which colors are inverted. The region's coordinates are presumed to be logical coordinates.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

On monochrome screens, the **InvertRgn** function makes white pixels black and black pixels white. On color screens, this inversion is dependent on the type of technology used to generate the colors for the screen.

Examples

For an example, see [Using Brushes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[FillRgn](#)

[PaintRgn](#)

[Region Functions](#)

[Regions Overview](#)

KERNINGPAIR structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The KERNINGPAIR structure defines a kerning pair.

Syntax

```
typedef struct tagKERNINGPAIR {  
    WORD wFirst;  
    WORD wSecond;  
    int iKernAmount;  
} KERNINGPAIR, *LPKERNINGPAIR;
```

Members

wFirst

The character code for the first character in the kerning pair.

wSecond

The character code for the second character in the kerning pair.

iKernAmount

The amount this pair will be kerned if they appear side by side in the same font and size. This value is typically negative, because pair kerning usually results in two characters being set more tightly than normal. The value is specified in logical units; that is, it depends on the current mapping mode.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetKerningPairs](#)

LineDDA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **LineDDA** function determines which pixels should be highlighted for a line defined by the specified starting and ending points.

Syntax

```
BOOL LineDDA(
    [in] int      xStart,
    [in] int      yStart,
    [in] int      xEnd,
    [in] int      yEnd,
    [in] LINEDDAPROC lpProc,
    [in] LPARAM   data
);
```

Parameters

[in] **xStart**

Specifies the x-coordinate, in logical units, of the line's starting point.

[in] **yStart**

Specifies the y-coordinate, in logical units, of the line's starting point.

[in] **xEnd**

Specifies the x-coordinate, in logical units, of the line's ending point.

[in] **yEnd**

Specifies the y-coordinate, in logical units, of the line's ending point.

[in] **lpProc**

Pointer to an application-defined callback function. For more information, see the [LineDDAProc](#) callback function.

[in] **data**

Pointer to the application-defined data.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **LineDDA** function passes the coordinates for each point along the line, except for the line's ending point, to the application-defined callback function. In addition to passing the coordinates of a point, this function passes any existing application-defined data.

The coordinates passed to the callback function match pixels on a video display only if the default transformations and mapping modes are used.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Line and Curve Functions](#)

[LineDDAProc](#)

[Lines and Curves Overview](#)

LINEDDAPROC callback function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **LineDDAProc** function is an application-defined callback function used with the [LineDDA](#) function. It is used to process coordinates. The **LINEDDAPROC** type defines a pointer to this callback function. **LineDDAProc** is a placeholder for the application-defined function name.

Syntax

```
LINEDDAPROC Lineddaproc;  
  
void Lineddaproc(  
    int unnamedParam1,  
    int unnamedParam2,  
    LPARAM unnamedParam3  
)  
{...}
```

Parameters

unnamedParam1

unnamedParam2

unnamedParam3

Return value

None

Remarks

An application registers a **LineDDAProc** function by passing its address to the [LineDDA](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[Line and Curve Functions](#)

LineDDA

Lines and Curves Overview

LineTo function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **LineTo** function draws a line from the current position up to, but not including, the specified point.

Syntax

```
BOOL LineTo(  
    [in] HDC hdc,  
    [in] int x,  
    [in] int y  
);
```

Parameters

[in] **hdc**

Handle to a device context.

[in] **x**

Specifies the x-coordinate, in logical units, of the line's ending point.

[in] **y**

Specifies the y-coordinate, in logical units, of the line's ending point.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The line is drawn by using the current pen and, if the pen is a geometric pen, the current brush.

If **LineTo** succeeds, the current position is set to the specified ending point.

Examples

For an example, see [Drawing Markers](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

[MoveToEx](#)

[Polyline](#)

[PolylineTo](#)

LOGBRUSH structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **LOGBRUSH** structure defines the style, color, and pattern of a physical brush. It is used by the [CreateBrushIndirect](#) and [ExtCreatePen](#) functions.

Syntax

```
typedef struct tagLOGBRUSH {  
    UINT     lbStyle;  
    COLORREF lbColor;  
    ULONG_PTR lbHatch;  
} LOGBRUSH, *PLOGBRUSH, *NPLOGBRUSH, *LPLOGBRUSH;
```

Members

lbStyle

The brush style. The **lbStyle** member must be one of the following styles.

VALUE	MEANING
BS_DIBPATTERN	A pattern brush defined by a device-independent bitmap (DIB) specification. If lbStyle is BS_DIBPATTERN, the lbHatch member contains a handle to a packed DIB. For more information, see discussion in lbHatch .
BS_DIBPATTERN8X8	See BS_DIBPATTERN.
BS_DIBPATTERNNPT	A pattern brush defined by a device-independent bitmap (DIB) specification. If lbStyle is BS_DIBPATTERNNPT, the lbHatch member contains a pointer to a packed DIB. For more information, see discussion in lbHatch .
BS_HATCHED	Hatched brush.
BS_HOLLOW	Hollow brush.
BS_NULL	Same as BS_HOLLOW.
BS_PATTERN	Pattern brush defined by a memory bitmap.
BS_PATTERN8X8	See BS_PATTERN.
BS_SOLID	Solid brush.

lbColor

The color in which the brush is to be drawn. If **lbStyle** is the BS_HOLLOW or BS_PATTERN style, **lbColor** is ignored.

If **lbStyle** is BS_DIBPATTERN or BS_DIBPATTERNNPT, the low-order word of **lbColor** specifies whether the

bmiColors members of the [BITMAPINFO](#) structure contain explicit red, green, blue (RGB) values or indexes into the currently realized logical palette. The **IbColor** member must be one of the following values.

VALUE	MEANING
DIB_PAL_COLORS	The color table consists of an array of 16-bit indexes into the currently realized logical palette.
DIB_RGB_COLORS	The color table contains literal RGB values.

If **IbStyle** is BS_HATCHED or BS_SOLID, **IbColor** is a [COLORREF](#) color value. To create a [COLORREF](#) color value, use the [RGB](#) macro.

1bHatch

A hatch style. The meaning depends on the brush style defined by **IbStyle**.

If **IbStyle** is BS_DIBPATTERN, the **IbHatch** member contains a handle to a packed DIB. To obtain this handle, an application calls the [GlobalAlloc](#) function with GMEM_MOVEABLE (or [LocalAlloc](#) with LMEM_MOVEABLE) to allocate a block of memory and then fills the memory with the packed DIB. A packed DIB consists of a [BITMAPINFO](#) structure immediately followed by the array of bytes that define the pixels of the bitmap.

If **IbStyle** is BS_DIBPATTERNPT, the **IbHatch** member contains a pointer to a packed DIB. The pointer derives from the memory block created by [LocalAlloc](#) with LMEM_FIXED set or by [GlobalAlloc](#) with GMEM_FIXED set, or it is the pointer returned by a call like [LocalLock](#) (handle_to_the_dib). A packed DIB consists of a [BITMAPINFO](#) structure immediately followed by the array of bytes that define the pixels of the bitmap.

If **IbStyle** is BS_HATCHED, the **IbHatch** member specifies the orientation of the lines used to create the hatch. It can be one of the following values.

VALUE	MEANING
HS_BDIAGONAL	A 45-degree upward, left-to-right hatch
HS_CROSS	Horizontal and vertical cross-hatch
HS_DIAGCROSS	45-degree crosshatch
HS_FDIAGONAL	A 45-degree downward, left-to-right hatch
HS_HORIZONTAL	Horizontal hatch
HS_VERTICAL	Vertical hatch

If **IbStyle** is BS_PATTERN, **IbHatch** is a handle to the bitmap that defines the pattern. The bitmap cannot be a DIB section bitmap, which is created by the [CreateDIBSection](#) function.

If **IbStyle** is BS_SOLID or BS_HOLLOW, **IbHatch** is ignored.

Remarks

Although **IbColor** controls the foreground color of a hatch brush, the [SetBkMode](#) and [SetBkColor](#) functions control the background color.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[Brush Structures](#)

[Brushes Overview](#)

[COLORREF](#)

[CreateBrushIndirect](#)

[CreateDIBSection](#)

[ExtCreatePen](#)

[LOGBRUSH32](#)

[RGB](#)

[SetBkColor](#)

[SetBkMode](#)

LOGBRUSH32 structure (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **LOGBRUSH32** structure defines the style, color, and pattern of a physical brush. It is similar to [LOGBRUSH](#), but it is used to maintain compatibility between 32-bit platforms and 64-bit platforms when we record the metafile record on one platform and then play it on another. Thus, it is only used in [EMRCREATEBRUSHINDIRECT](#). If the code will only be on one platform, **LOGBRUSH** is sufficient.

Syntax

```
typedef struct tagLOGBRUSH32 {  
    UINT     lbStyle;  
    COLORREF lbColor;  
    ULONG    lbHatch;  
} LOGBRUSH32, *PLOGBRUSH32, *NPLOGBRUSH32, *LPLOGBRUSH32;
```

Members

lbStyle

The brush style. The **IbStyle** member must be one of the following styles.

VALUE	MEANING
BS_DIBPATTERN	A pattern brush defined by a device-independent bitmap (DIB) specification. If IbStyle is BS_DIBPATTERN, the IbHatch member contains a handle to a packed DIB. For more information, see discussion in IbHatch .
BS_DIBPATTERN8X8	Same as BS_DIBPATTERN.
BS_DIBPATTERNPT	A pattern brush defined by a device-independent bitmap (DIB) specification. If IbStyle is BS_DIBPATTERNPT, the IbHatch member contains a pointer to a packed DIB. For more information, see discussion in IbHatch .
BS_HATCHED	Hatched brush.
BS_HOLLOW	Hollow brush.
BS_NULL	Same as BS_HOLLOW.
BS_PATTERN	Pattern brush defined by a memory bitmap.
BS_PATTERN8X8	Same as BS_PATTERN.
BS_SOLID	Solid brush.

lbColor

The color in which the brush is to be drawn. If **IbStyle** is the BS_HOLLOW or BS_PATTERN style, **IbColor** is

ignored.

If **IbStyle** is BS_DIBPATTERN or BS_DIBPATTERNPT, the low-order word of **IbColor** specifies whether the **bmiColors** members of the [BITMAPINFO](#) structure contain explicit red, green, blue (RGB) values or indexes into the currently realized logical palette. The **IbColor** member must be one of the following values.

VALUE	MEANING
DIB_PAL_COLORS	The color table consists of an array of 16-bit indexes into the currently realized logical palette.
DIB_RGB_COLORS	The color table contains literal RGB values.

If **IbStyle** is BS_HATCHED or BS_SOLID, **IbColor** is a [COLORREF](#) color value. To create a [COLORREF](#) color value, use the [RGB](#) macro.

IbHatch

A hatch style. The meaning depends on the brush style defined by **IbStyle**.

If **IbStyle** is BS_DIBPATTERN, the **IbHatch** member contains a handle to a packed DIB. To obtain this handle, an application calls the [GlobalAlloc](#) function with GMEM_MOVEABLE (or [LocalAlloc](#) with LMEM_MOVEABLE) to allocate a block of memory and then fills the memory with the packed DIB. A packed DIB consists of a [BITMAPINFO](#) structure immediately followed by the array of bytes that define the pixels of the bitmap.

If **IbStyle** is BS_DIBPATTERNPT, the **IbHatch** member contains a pointer to a packed DIB. The pointer derives from the memory block created by [LocalAlloc](#) with LMEM_FIXED set or by [GlobalAlloc](#) with GMEM_FIXED set, or it is the pointer returned by a call like [LocalLock](#) (handle_to_the_dib). A packed DIB consists of a [BITMAPINFO](#) structure immediately followed by the array of bytes that define the pixels of the bitmap.

If **IbStyle** is BS_HATCHED, the **IbHatch** member specifies the orientation of the lines used to create the hatch. It can be one of the following values.

VALUE	MEANING
HS_BDIAGONAL	A 45-degree upward, left-to-right hatch
HS_CROSS	Horizontal and vertical cross-hatch
HS_DIAGCROSS	45-degree crosshatch
HS_FDIAGONAL	A 45-degree downward, left-to-right hatch
HS_HORIZONTAL	Horizontal hatch
HS_VERTICAL	Vertical hatch

If **IbStyle** is BS_PATTERN, **IbHatch** is a handle to the bitmap that defines the pattern. The bitmap cannot be a DIB section bitmap, which is created by the [CreateDIBSection](#) function.

If **IbStyle** is BS_SOLID or BS_HOLLOW, **IbHatch** is ignored.

Remarks

Although **IbColor** controls the foreground color of a hatch brush, the [SetBkMode](#) and [SetBkColor](#) functions control the background color.

Brushes can be created from bitmaps or DIBs larger than 8 by 8 pixels.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[Brush Structures](#)

[Brushes Overview](#)

[COLORREF](#)

[CreateDIBSection](#)

[EMRCREATEBRUSHINDIRECT](#)

[LOGBRUSH](#)

[RGB](#)

[SetBkColor](#)

[SetBkMode](#)

LOGFONTA structure (wingdi.h)

4/21/2022 • 8 minutes to read • [Edit Online](#)

The LOGFONT structure defines the attributes of a font.

Syntax

```
typedef struct tagLOGFONTA {
    LONG lfHeight;
    LONG lfWidth;
    LONG lfEscapement;
    LONG lfOrientation;
    LONG lfWeight;
    BYTE lfItalic;
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE lfClipPrecision;
    BYTE lfQuality;
    BYTE lfPitchAndFamily;
    CHAR lfFaceName[LF_FACESIZE];
} LOGFONTA, *PLOGFONTA, *NPLOGFONTA, *LPLOGFONTA;
```

Members

lfHeight

The height, in logical units, of the font's character cell or character. The character height value (also known as the em height) is the character cell height value minus the internal-leading value. The font mapper interprets the value specified in **lfHeight** in the following manner.

VALUE	MEANING
> 0	The font mapper transforms this value into device units and matches it against the cell height of the available fonts.
0	The font mapper uses a default height value when it searches for a match.
< 0	The font mapper transforms this value into device units and matches its absolute value against the character height of the available fonts.

For all height comparisons, the font mapper looks for the largest font that does not exceed the requested size.

This mapping occurs when the font is used for the first time.

For the MM_TEXT mapping mode, you can use the following formula to specify a height for a font with a specified point size:

```
lfHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);
```

lfWidth

The average width, in logical units, of characters in the font. If **IfWidth** is zero, the aspect ratio of the device is matched against the digitization aspect ratio of the available fonts to find the closest match, determined by the absolute value of the difference.

lfEscapement

The angle, in tenths of degrees, between the escapement vector and the x-axis of the device. The escapement vector is parallel to the base line of a row of text.

When the graphics mode is set to GM_ADVANCED, you can specify the escapement angle of the string independently of the orientation angle of the string's characters.

When the graphics mode is set to GM_COMPATIBLE, **IfEscapement** specifies both the escapement and orientation. You should set **IfEscapement** and **IfOrientation** to the same value.

lfOrientation

The angle, in tenths of degrees, between each character's base line and the x-axis of the device.

lfWeight

The weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is zero, a default weight is used.

The following values are defined for convenience.

VALUE	WEIGHT
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700

FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

`lfItalic`

An italic font if set to **TRUE**.

`lfUnderline`

An underlined font if set to **TRUE**.

`lfStrikeOut`

A strikeout font if set to **TRUE**.

`lfCharSet`

The character set. The following values are predefined.

ANSI_CHARSET
 BALTIC_CHARSET
 CHINESEBIG5_CHARSET
 DEFAULT_CHARSET
 EASTEUROPE_CHARSET
 GB2312_CHARSET
 GREEK_CHARSET
 HANGUL_CHARSET
 MAC_CHARSET
 OEM_CHARSET
 RUSSIAN_CHARSET
 SHIFTJIS_CHARSET
 SYMBOL_CHARSET
 TURKISH_CHARSET
 VIETNAMESE_CHARSET

Korean language edition of Windows:

JOHAB_CHARSET

Middle East language edition of Windows:

ARABIC_CHARSET
 HEBREW_CHARSET

Thai language edition of Windows:

THAI_CHARSET

The OEM_CHARSET value specifies a character set that is operating-system dependent.

DEFAULT_CHARSET is set to a value based on the current system locale. For example, when the system locale is English (United States), it is set as ANSI_CHARSET.

Fonts with other character sets may exist in the operating system. If an application uses a font with an unknown character set, it should not attempt to translate or interpret strings that are rendered with that font.

This parameter is important in the font mapping process. To ensure consistent results, specify a specific character set. If you specify a typeface name in the **IfFaceName** member, make sure that the **IfCharSet** value matches the character set of the typeface specified in **IfFaceName**.

1fOutPrecision

The output precision. The output precision defines how closely the output must match the requested font's height, width, character orientation, escapement, pitch, and font type. It can be one of the following values.

VALUE	MEANING
OUT_CHARACTER_PRECIS	Not used.
OUT_DEFAULT_PRECIS	Specifies the default font mapper behavior.
OUT_DEVICE_PRECIS	Instructs the font mapper to choose a Device font when the system contains multiple fonts with the same name.
OUT_OUTLINE_PRECIS	This value instructs the font mapper to choose from TrueType and other outline-based fonts.
OUT_PS_ONLY_PRECIS	Instructs the font mapper to choose from only PostScript fonts. If there are no PostScript fonts installed in the system, the font mapper returns to default behavior.
OUT_RASTER_PRECIS	Instructs the font mapper to choose a raster font when the system contains multiple fonts with the same name.
OUT_STRING_PRECIS	This value is not used by the font mapper, but it is returned when raster fonts are enumerated.
OUT_STROKE_PRECIS	This value is not used by the font mapper, but it is returned when TrueType, other outline-based fonts, and vector fonts are enumerated.
OUT_TT_ONLY_PRECIS	Instructs the font mapper to choose from only TrueType fonts. If there are no TrueType fonts installed in the system, the font mapper returns to default behavior.
OUT_TT_PRECIS	Instructs the font mapper to choose a TrueType font when the system contains multiple fonts with the same name.

Applications can use the OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS, OUT_TT_PRECIS, and OUT_PS_ONLY_PRECIS values to control how the font mapper chooses a font when the operating system contains more than one font with a specified name. For example, if an operating system contains a font named Symbol in raster and TrueType form, specifying OUT_TT_PRECIS forces the font mapper to choose the TrueType version. Specifying OUT_TT_ONLY_PRECIS forces the font mapper to choose a TrueType font, even if it must substitute a TrueType font of another name.

1fClipPrecision

The clipping precision. The clipping precision defines how to clip characters that are partially outside the clipping region. It can be one or more of the following values.

For more information about the orientation of coordinate systems, see the description of the *nOrientation* parameter.

VALUE	MEANING
CLIP_CHARACTER_PRECIS	Not used.
CLIP_DEFAULT_PRECIS	Specifies default clipping behavior.
CLIP_DFA_DISABLE	Windows XP SP1: Turns off font association for the font. Note that this flag is not guaranteed to have any effect on any platform after Windows Server 2003.
CLIP_EMBEDDED	You must specify this flag to use an embedded read-only font.
CLIP_LH_ANGLES	When this value is used, the rotation for all fonts depends on whether the orientation of the coordinate system is left-handed or right-handed. If not used, device fonts always rotate counterclockwise, but the rotation of other fonts is dependent on the orientation of the coordinate system.
CLIP_MASK	Not used.
CLIP_DFA_OVERRIDE	Turns off font association for the font. This is identical to CLIP_DFA_DISABLE, but it can have problems in some situations; the recommended flag to use is CLIP_DFA_DISABLE.
CLIP_STROKE_PRECIS	Not used by the font mapper, but is returned when raster, vector, or TrueType fonts are enumerated. For compatibility, this value is always returned when enumerating fonts.
CLIP_TT_ALWAYS	Not used.

lfQuality

The output quality. The output quality defines how carefully the graphics device interface (GDI) must attempt to match the logical-font attributes to those of an actual physical font. It can be one of the following values.

VALUE	MEANING
ANTIALIASED_QUALITY	Font is always antialiased if the font supports it and the size of the font is not too small or too large.
CLEARTYPE_QUALITY	If set, text is rendered (when possible) using ClearType antialiasing method. See Remarks for more information.
DEFAULT_QUALITY	Appearance of the font does not matter.
DRAFT_QUALITY	Appearance of the font is less important than when PROOF_QUALITY is used. For GDI raster fonts, scaling is enabled, which means that more font sizes are available, but the quality may be lower. Bold, italic, underline, and strikeout fonts are synthesized if necessary.
NONANTIALIASED_QUALITY	Font is never antialiased.

PROOF_QUALITY	Character quality of the font is more important than exact matching of the logical-font attributes. For GDI raster fonts, scaling is disabled and the font closest in size is chosen. Although the chosen font size may not be mapped exactly when PROOF_QUALITY is used, the quality of the font is high and there is no distortion of appearance. Bold, italic, underline, and strikeout fonts are synthesized if necessary.
---------------	--

If neither ANTIALIASED_QUALITY nor NONANTIALIASED_QUALITY is selected, the font is antialiased only if the user chooses smooth screen fonts in Control Panel.

lfPitchAndFamily

The pitch and family of the font. The two low-order bits specify the pitch of the font and can be one of the following values.

- DEFAULT_PITCH
- FIXED_PITCH
- VARIABLE_PITCH

Bits 4 through 7 of the member specify the font family and can be one of the following values.

- FF_DECORATIVE
- FF_DONTCARE
- FF_MODERN
- FF_ROMAN
- FF_SCRIPT
- FF_SWISS

The proper value can be obtained by using the Boolean OR operator to join one pitch constant with one family constant.

Font families describe the look of a font in a general way. They are intended for specifying fonts when the exact typeface desired is not available. The values for font families are as follows.

VALUE	MEANING
FF_DECORATIVE	Novelty fonts. Old English is an example.
FF_DONTCARE	Use default font.
FF_MODERN	Fonts with constant stroke width (monospace), with or without serifs. Monospace fonts are usually modern. Pica, Elite, and CourierNew are examples.
FF_ROMAN	Fonts with variable stroke width (proportional) and with serifs. MS Serif is an example.
FF_SCRIPT	Fonts designed to look like handwriting. Script and Cursive are examples.
FF_SWISS	Fonts with variable stroke width (proportional) and without serifs. MS Sans Serif is an example.

lfFaceName

A null-terminated string that specifies the typeface name of the font. The length of this string must not exceed 32 **TCHAR** values, including the terminating **NULL**. The [EnumFontFamiliesEx](#) function can be used to enumerate the typeface names of all currently available fonts. If **IfFaceName** is an empty string, GDI uses the first font that matches the other specified attributes.

Remarks

The following situations do not support ClearType antialiasing:

- Text is rendered on a printer.
- Display set for 256 colors or less.
- Text is rendered to a terminal server client.
- The font is not a TrueType font or an OpenType font with TrueType outlines. For example, the following do not support ClearType antialiasing: Type 1 fonts, Postscript OpenType fonts without TrueType outlines, bitmap fonts, vector fonts, and device fonts.
- The font has tuned embedded bitmaps, for any font sizes that contain the embedded bitmaps. For example, this occurs commonly in East Asian fonts.

NOTE

The wingdi.h header defines **LOGFONT** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps UWP apps]
Minimum supported server	Windows 2000 Server [desktop apps UWP apps]
Header	wingdi.h (include Windows.h)

See also

[CreateFont](#)

[CreateFontIndirect](#)

[EnumFontFamiliesEx](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

LOGFONTW structure (wingdi.h)

4/21/2022 • 8 minutes to read • [Edit Online](#)

The LOGFONT structure defines the attributes of a font.

Syntax

```
typedef struct tagLOGFONTW {
    LONG lfHeight;
    LONG lfWidth;
    LONG lfEscapement;
    LONG lfOrientation;
    LONG lfWeight;
    BYTE lfItalic;
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE lfClipPrecision;
    BYTE lfQuality;
    BYTE lfPitchAndFamily;
    WCHAR lfFaceName[LF_FACESIZE];
} LOGFONTW, *PLOGFONTW, *NPLOGFONTW, *LPLOGFONTW;
```

Members

lfHeight

The height, in logical units, of the font's character cell or character. The character height value (also known as the em height) is the character cell height value minus the internal-leading value. The font mapper interprets the value specified in **lfHeight** in the following manner.

VALUE	MEANING
> 0	The font mapper transforms this value into device units and matches it against the cell height of the available fonts.
0	The font mapper uses a default height value when it searches for a match.
< 0	The font mapper transforms this value into device units and matches its absolute value against the character height of the available fonts.

For all height comparisons, the font mapper looks for the largest font that does not exceed the requested size.

This mapping occurs when the font is used for the first time.

For the MM_TEXT mapping mode, you can use the following formula to specify a height for a font with a specified point size:

```
lfHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);
```

lfWidth

The average width, in logical units, of characters in the font. If **IfWidth** is zero, the aspect ratio of the device is matched against the digitization aspect ratio of the available fonts to find the closest match, determined by the absolute value of the difference.

lfEscapement

The angle, in tenths of degrees, between the escapement vector and the x-axis of the device. The escapement vector is parallel to the base line of a row of text.

When the graphics mode is set to GM_ADVANCED, you can specify the escapement angle of the string independently of the orientation angle of the string's characters.

When the graphics mode is set to GM_COMPATIBLE, **IfEscapement** specifies both the escapement and orientation. You should set **IfEscapement** and **IfOrientation** to the same value.

lfOrientation

The angle, in tenths of degrees, between each character's base line and the x-axis of the device.

lfWeight

The weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is zero, a default weight is used.

The following values are defined for convenience.

VALUE	WEIGHT
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700

FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

`lfItalic`

An italic font if set to **TRUE**.

`lfUnderline`

An underlined font if set to **TRUE**.

`lfStrikeOut`

A strikeout font if set to **TRUE**.

`lfCharSet`

The character set. The following values are predefined.

ANSI_CHARSET
 BALTIC_CHARSET
 CHINESEBIG5_CHARSET
 DEFAULT_CHARSET
 EASTEUROPE_CHARSET
 GB2312_CHARSET
 GREEK_CHARSET
 HANGUL_CHARSET
 MAC_CHARSET
 OEM_CHARSET
 RUSSIAN_CHARSET
 SHIFTJIS_CHARSET
 SYMBOL_CHARSET
 TURKISH_CHARSET
 VIETNAMESE_CHARSET

Korean language edition of Windows:

JOHAB_CHARSET

Middle East language edition of Windows:

ARABIC_CHARSET
 HEBREW_CHARSET

Thai language edition of Windows:

THAI_CHARSET

The OEM_CHARSET value specifies a character set that is operating-system dependent.

DEFAULT_CHARSET is set to a value based on the current system locale. For example, when the system locale is English (United States), it is set as ANSI_CHARSET.

Fonts with other character sets may exist in the operating system. If an application uses a font with an unknown character set, it should not attempt to translate or interpret strings that are rendered with that font.

This parameter is important in the font mapping process. To ensure consistent results, specify a specific character set. If you specify a typeface name in the **IfFaceName** member, make sure that the **IfCharSet** value matches the character set of the typeface specified in **IfFaceName**.

1fOutPrecision

The output precision. The output precision defines how closely the output must match the requested font's height, width, character orientation, escapement, pitch, and font type. It can be one of the following values.

VALUE	MEANING
OUT_CHARACTER_PRECIS	Not used.
OUT_DEFAULT_PRECIS	Specifies the default font mapper behavior.
OUT_DEVICE_PRECIS	Instructs the font mapper to choose a Device font when the system contains multiple fonts with the same name.
OUT_OUTLINE_PRECIS	This value instructs the font mapper to choose from TrueType and other outline-based fonts.
OUT_PS_ONLY_PRECIS	Instructs the font mapper to choose from only PostScript fonts. If there are no PostScript fonts installed in the system, the font mapper returns to default behavior.
OUT_RASTER_PRECIS	Instructs the font mapper to choose a raster font when the system contains multiple fonts with the same name.
OUT_STRING_PRECIS	This value is not used by the font mapper, but it is returned when raster fonts are enumerated.
OUT_STROKE_PRECIS	This value is not used by the font mapper, but it is returned when TrueType, other outline-based fonts, and vector fonts are enumerated.
OUT_TT_ONLY_PRECIS	Instructs the font mapper to choose from only TrueType fonts. If there are no TrueType fonts installed in the system, the font mapper returns to default behavior.
OUT_TT_PRECIS	Instructs the font mapper to choose a TrueType font when the system contains multiple fonts with the same name.

Applications can use the OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS, OUT_TT_PRECIS, and OUT_PS_ONLY_PRECIS values to control how the font mapper chooses a font when the operating system contains more than one font with a specified name. For example, if an operating system contains a font named Symbol in raster and TrueType form, specifying OUT_TT_PRECIS forces the font mapper to choose the TrueType version. Specifying OUT_TT_ONLY_PRECIS forces the font mapper to choose a TrueType font, even if it must substitute a TrueType font of another name.

1fClipPrecision

The clipping precision. The clipping precision defines how to clip characters that are partially outside the clipping region. It can be one or more of the following values.

For more information about the orientation of coordinate systems, see the description of the *nOrientation* parameter.

VALUE	MEANING
CLIP_CHARACTER_PRECIS	Not used.
CLIP_DEFAULT_PRECIS	Specifies default clipping behavior.
CLIP_DFA_DISABLE	Windows XP SP1: Turns off font association for the font. Note that this flag is not guaranteed to have any effect on any platform after Windows Server 2003.
CLIP_EMBEDDED	You must specify this flag to use an embedded read-only font.
CLIP_LH_ANGLES	When this value is used, the rotation for all fonts depends on whether the orientation of the coordinate system is left-handed or right-handed. If not used, device fonts always rotate counterclockwise, but the rotation of other fonts is dependent on the orientation of the coordinate system.
CLIP_MASK	Not used.
CLIP_DFA_OVERRIDE	Turns off font association for the font. This is identical to CLIP_DFA_DISABLE, but it can have problems in some situations; the recommended flag to use is CLIP_DFA_DISABLE.
CLIP_STROKE_PRECIS	Not used by the font mapper, but is returned when raster, vector, or TrueType fonts are enumerated. For compatibility, this value is always returned when enumerating fonts.
CLIP_TT_ALWAYS	Not used.

lfQuality

The output quality. The output quality defines how carefully the graphics device interface (GDI) must attempt to match the logical-font attributes to those of an actual physical font. It can be one of the following values.

VALUE	MEANING
ANTIALIASED_QUALITY	Font is always antialiased if the font supports it and the size of the font is not too small or too large.
CLEARTYPE_QUALITY	If set, text is rendered (when possible) using ClearType antialiasing method. See Remarks for more information.
DEFAULT_QUALITY	Appearance of the font does not matter.
DRAFT_QUALITY	Appearance of the font is less important than when PROOF_QUALITY is used. For GDI raster fonts, scaling is enabled, which means that more font sizes are available, but the quality may be lower. Bold, italic, underline, and strikeout fonts are synthesized if necessary.
NONANTIALIASED_QUALITY	Font is never antialiased.

PROOF_QUALITY	Character quality of the font is more important than exact matching of the logical-font attributes. For GDI raster fonts, scaling is disabled and the font closest in size is chosen. Although the chosen font size may not be mapped exactly when PROOF_QUALITY is used, the quality of the font is high and there is no distortion of appearance. Bold, italic, underline, and strikeout fonts are synthesized if necessary.
---------------	--

If neither ANTIALIASED_QUALITY nor NONANTIALIASED_QUALITY is selected, the font is antialiased only if the user chooses smooth screen fonts in Control Panel.

lfPitchAndFamily

The pitch and family of the font. The two low-order bits specify the pitch of the font and can be one of the following values.

- DEFAULT_PITCH
- FIXED_PITCH
- VARIABLE_PITCH

Bits 4 through 7 of the member specify the font family and can be one of the following values.

- FF_DECORATIVE
- FF_DONTCARE
- FF_MODERN
- FF_ROMAN
- FF_SCRIPT
- FF_SWISS

The proper value can be obtained by using the Boolean OR operator to join one pitch constant with one family constant.

Font families describe the look of a font in a general way. They are intended for specifying fonts when the exact typeface desired is not available. The values for font families are as follows.

VALUE	MEANING
FF_DECORATIVE	Novelty fonts. Old English is an example.
FF_DONTCARE	Use default font.
FF_MODERN	Fonts with constant stroke width (monospace), with or without serifs. Monospace fonts are usually modern. Pica, Elite, and CourierNew are examples.
FF_ROMAN	Fonts with variable stroke width (proportional) and with serifs. MS Serif is an example.
FF_SCRIPT	Fonts designed to look like handwriting. Script and Cursive are examples.
FF_SWISS	Fonts with variable stroke width (proportional) and without serifs. MS Sans Serif is an example.

lfFaceName

A null-terminated string that specifies the typeface name of the font. The length of this string must not exceed 32 **TCHAR** values, including the terminating **NULL**. The [EnumFontFamiliesEx](#) function can be used to enumerate the typeface names of all currently available fonts. If **IfFaceName** is an empty string, GDI uses the first font that matches the other specified attributes.

Remarks

The following situations do not support ClearType antialiasing:

- Text is rendered on a printer.
- Display set for 256 colors or less.
- Text is rendered to a terminal server client.
- The font is not a TrueType font or an OpenType font with TrueType outlines. For example, the following do not support ClearType antialiasing: Type 1 fonts, Postscript OpenType fonts without TrueType outlines, bitmap fonts, vector fonts, and device fonts.
- The font has tuned embedded bitmaps, for any font sizes that contain the embedded bitmaps. For example, this occurs commonly in East Asian fonts.

NOTE

The wingdi.h header defines **LOGFONT** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps UWP apps]
Minimum supported server	Windows 2000 Server [desktop apps UWP apps]
Header	wingdi.h (include Windows.h)

See also

[CreateFont](#)

[CreateFontIndirect](#)

[EnumFontFamiliesEx](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

LOGPALETTE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The LOGPALETTE structure defines a logical palette.

Syntax

```
typedef struct tagLOGPALETTE {  
    WORD      palVersion;  
    WORD      palNumEntries;  
    PALETTEENTRY palPalEntry[1];  
} LOGPALETTE, *PLOGPALETTE, *NPLOGPALETTE, *LPLOGPALETTE;
```

Members

`palVersion`

The version number of the system.

`palNumEntries`

The number of entries in the logical palette.

`palPalEntry`

Specifies an array of [PALETTEENTRY](#) structures that define the color and usage of each entry in the logical palette.

Remarks

The colors in the palette-entry table should appear in order of importance because entries earlier in the logical palette are most likely to be placed in the system palette.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Color Structures](#)

[Colors Overview](#)

[CreatePalette](#)

[PALETTEENTRY](#)

LOGPEN structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **LOGPEN** structure defines the style, width, and color of a pen. The [CreatePenIndirect](#) function uses the **LOGPEN** structure.

Syntax

```
typedef struct tagLOGPEN {  
    UINT     lopnStyle;  
    POINT    lopnWidth;  
    COLORREF lopnColor;  
} LOGPEN, *PLOGPEN, *NPLOGPEN, *LPLOGPEN;
```

Members

lopnStyle

The pen style, which can be one of the following values.

VALUE	MEANING
PS_SOLID	The pen is solid.
PS_DASH	The pen is dashed.
PS_DOT	The pen is dotted.
PS_DASHDOT	The pen has alternating dashes and dots.
PS_DASHDOTDOT	The pen has dashes and double dots.
PS_NULL	The pen is invisible.
PS_INSIDEFRAME	The pen is solid. When this pen is used in any GDI drawing function that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to geometric pens.

lopnWidth

The [POINT](#) structure that contains the pen width, in logical units. If the **pointer** member is **NULL**, the pen is one pixel wide on raster devices. The **y** member in the [POINT](#) structure for **lopnWidth** is not used.

lopnColor

The pen color. To generate a [COLORREF](#) structure, use the [RGB](#) macro.

Remarks

If the width of the pen is greater than 1 and the pen style is **PS_INSIDEFRAME**, the line is drawn inside the frame

of all GDI objects except polygons and polylines. If the pen color does not match an available RGB value, the pen is drawn with a logical (dithered) color. If the pen width is less than or equal to 1, the PS_INSIDEFRAME style is identical to the PS_SOLID style.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[COLORREF](#)

[CreatePenIndirect](#)

[POINT](#)

[Pen Structures](#)

[Pens Overview](#)

[RGB](#)

LPtoDP function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **LPtoDP** function converts logical coordinates into device coordinates. The conversion depends on the mapping mode of the device context, the settings of the origins and extents for the window and viewport, and the world transformation.

Syntax

```
BOOL LPtoDP(
    [in]      HDC      hdc,
    [in, out] LPPOINT lppt,
    [in]      int      c
);
```

Parameters

[in] hdc

A handle to the device context.

[in, out] lppt

A pointer to an array of **POINT** structures. The x-coordinates and y-coordinates contained in each of the **POINT** structures will be transformed.

[in] c

The number of points in the array.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **LPtoDP** function fails if the logical coordinates exceed 32 bits, or if the converted device coordinates exceed 27 bits. In the case of such an overflow, the results for all the points are undefined.

LPtoDP calculates complex floating-point arithmetic, and it has a caching system for efficiency. Therefore, the conversion result of an initial call to **LPtoDP** might not exactly match the conversion result of a later call to **LPtoDP**. We recommend not to write code that relies on the exact match of the conversion results from multiple calls to **LPtoDP** even if the parameters that are passed to each call are identical.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[DPtoLP](#)

[POINT](#)

MAKEPOINTS macro (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MAKEPOINTS** macro converts a value that contains the x- and y-coordinates of a point into a [POINTS](#) structure.

Syntax

```
void MAKEPOINTS(  
    1  
);
```

Parameters

1

The coordinates of a point. The x-coordinate is in the low-order word, and the y-coordinate is in the high-order word.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[GetMessagePos](#)

[Rectangle Macros](#)

[Rectangles Overview](#)

MAKEROP4 macro (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MAKEROP4** macro creates a quaternary raster operation code for use with the [MaskBlt](#) function. The macro takes two ternary raster operation codes as input, one for the foreground and one for the background, and packs their Boolean operation indexes into the high-order word of a 32-bit value. The low-order word of this value will be ignored.

Syntax

```
void MAKEROP4(  
    fore,  
    back  
>);
```

Parameters

`fore`

The foreground ternary raster operation code.

`back`

The background ternary raster operation code.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[Bitmap Macros](#)

[Bitmaps Overview](#)

[MaskBlt](#)

MaskBlt function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **MaskBlt** function combines the color data for the source and destination bitmaps using the specified mask and raster operation.

Syntax

```
BOOL MaskBlt(
    [in] HDC      hdcDest,
    [in] int       xDest,
    [in] int       yDest,
    [in] int       width,
    [in] int       height,
    [in] HDC      hdcSrc,
    [in] int       xSrc,
    [in] int       ySrc,
    [in] HBITMAP  hbmMask,
    [in] int       xMask,
    [in] int       yMask,
    [in] DWORD     rop
);
```

Parameters

[in] `hdcDest`

A handle to the destination device context.

[in] `xDest`

The x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] `yDest`

The y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] `width`

The width, in logical units, of the destination rectangle and source bitmap.

[in] `height`

The height, in logical units, of the destination rectangle and source bitmap.

[in] `hdcSrc`

A handle to the device context from which the bitmap is to be copied. It must be zero if the `dwRop` parameter specifies a raster operation that does not include a source.

[in] `xSrc`

The x-coordinate, in logical units, of the upper-left corner of the source bitmap.

[in] `ySrc`

The y-coordinate, in logical units, of the upper-left corner of the source bitmap.

[in] *hbmMask*

A handle to the monochrome mask bitmap combined with the color bitmap in the source device context.

[in] *xMask*

The horizontal pixel offset for the mask bitmap specified by the *hbmMask* parameter.

[in] *yMask*

The vertical pixel offset for the mask bitmap specified by the *hbmMask* parameter.

[in] *rop*

The foreground and background ternary raster operation codes (ROPs) that the function uses to control the combination of source and destination data. The background raster operation code is stored in the high-order byte of the high-order word of this value; the foreground raster operation code is stored in the low-order byte of the high-order word of this value; the low-order word of this value is ignored, and should be zero. The macro [MAKEROP4](#) creates such combinations of foreground and background raster operation codes.

For a discussion of foreground and background in the context of this function, see the following Remarks section.

For a list of common raster operation codes (ROPs), see the [BitBlt](#) function. Note that the CAPTUREBLT ROP generally cannot be used for printing device contexts.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The [MaskBlt](#) function uses device-dependent bitmaps.

A value of 1 in the mask specified by *hbmMask* indicates that the foreground raster operation code specified by *dwRop* should be applied at that location. A value of 0 in the mask indicates that the background raster operation code specified by *dwRop* should be applied at that location.

If the raster operations require a source, the mask rectangle must cover the source rectangle. If it does not, the function will fail. If the raster operations do not require a source, the mask rectangle must cover the destination rectangle. If it does not, the function will fail.

If a rotation or shear transformation is in effect for the source device context when this function is called, an error occurs. However, other types of transformation are allowed.

If the color formats of the source, pattern, and destination bitmaps differ, this function converts the pattern or source format, or both, to match the destination format.

If the mask bitmap is not a monochrome bitmap, an error occurs.

When an enhanced metafile is being recorded, an error occurs (and the function returns FALSE) if the source device context identifies an enhanced-metafile device context.

Not all devices support the [MaskBlt](#) function. An application should call the [GetDeviceCaps](#) function with the *nIndex* parameter as RC_BITBLT to determine whether a device supports this function.

If no mask bitmap is supplied, this function behaves exactly like [BitBlt](#), using the foreground raster operation code.

ICM: No color management is performed when blits occur.

When used in a multiple monitor system, both *hdcSrc* and *hdcDest* must refer to the same device or the function will fail. To transfer data between DCs for different devices, convert the memory bitmap (compatible bitmap, or DDB) to a DIB by calling [GetDIBits](#). To display the DIB to the second device, call [SetDIBits](#) or [StretchDIBits](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BitBlt](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[GetDIBits](#)

[GetDeviceCaps](#)

[PlgBlt](#)

[SetDIBits](#)

[StretchBlt](#)

[StretchDIBits](#)

MAT2 structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The MAT2 structure contains the values for a transformation matrix used by the [GetGlyphOutline](#) function.

Syntax

```
typedef struct _MAT2 {  
    FIXED eM11;  
    FIXED eM12;  
    FIXED eM21;  
    FIXED eM22;  
} MAT2, *LPMAT2;
```

Members

eM11

A fixed-point value for the M11 component of a 3 by 3 transformation matrix.

eM12

A fixed-point value for the M12 component of a 3 by 3 transformation matrix.

eM21

A fixed-point value for the M21 component of a 3 by 3 transformation matrix.

eM22

A fixed-point value for the M22 component of a 3 by 3 transformation matrix.

Remarks

The identity matrix produces a transformation in which the transformed graphical object is identical to the source object. In the identity matrix, the value of **eM11** is 1, the value of **eM12** is zero, the value of **eM21** is zero, and the value of **eM22** is 1.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetGlyphOutline](#)

METAHEADER structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The METAHEADER structure contains information about a Windows-format metafile.

Syntax

```
typedef struct tagMETAHEADER {  
    WORD    mtType;  
    WORD    mtHeaderSize;  
    WORD    mtVersion;  
    DWORD   mtSize;  
    WORD    mtNoObjects;  
    DWORD   mtMaxRecord;  
    WORD    mtNoParameters;  
} METAHEADER, *PMETAHEADER, *LPMETAHEADER;
```

Members

mtType

Specifies whether the metafile is in memory or recorded in a disk file. This member can be one of the following values.

VALUE	MEANING
1	Metafile is in memory.
2	Metafile is in a disk file.

mtHeaderSize

The size, in words, of the metafile header.

mtVersion

The system version number. The version number for metafiles that support device-independent bitmaps (DIBs) is 0x0300. Otherwise, the version number is 0x0100.

mtSize

The size, in words, of the file.

mtNoObjects

The maximum number of objects that exist in the metafile at the same time.

mtMaxRecord

The size, in words, of the largest record in the metafile.

mtNoParameters

Reserved.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[METARECORD](#)

[Metafile Structures](#)

[Metafiles Overview](#)

METARECORD structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The METARECORD structure contains a Windows-format metafile record.

Syntax

```
typedef struct tagMETARECORD {  
    DWORD rdSize;  
    WORD  rdFunction;  
    WORD  rdParm[1];  
} METARECORD, *PMETARECORD, *LPMETARECORD;
```

Members

`rdSize`

The size, in words, of the record.

`rdFunction`

The function number.

`rdParm`

An array of words containing the function parameters, in reverse of the order they are passed to the function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[METAHEADER](#)

[Metafile Structures](#)

[Metafiles Overview](#)

MFENUMPROC callback function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumMetaFileProc** function is an application-defined callback function that processes Windows-format metafile records. This function is called by the [EnumMetaFile](#) function. The **MFENUMPROC** type defines a pointer to this callback function. **EnumMetaFileProc** is a placeholder for the application-defined function name.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [EnhMetaFileProc](#).

Syntax

```
MFENUMPROC Mfenumproc;

int Mfenumproc(
    HDC hdc,
    HANDLETABLE *lph,
    METARECORD *lpMR,
    [in] int nObj,
    LPARAM param
)
{...}
```

Parameters

hdc

lph

lpMR

[in] nObj

Specifies the number of objects with associated handles in the handle table.

param

Return value

This function must return a nonzero value to continue enumeration; to stop enumeration, it must return zero.

Remarks

An application must register the callback function by passing its address to the [EnumMetaFile](#) function.

EnumMetaFileProc is a placeholder for the application-supplied function name.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[EnhMetaFileProc](#)

[EnumEnhMetaFile](#)

[EnumMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

ModifyWorldTransform function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ModifyWorldTransform** function changes the world transformation for a device context using the specified mode.

Syntax

```
BOOL ModifyWorldTransform(
    [in] HDC      hdc,
    [in] const XFORM *lpxf,
    [in] DWORD     mode
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpxf`

A pointer to an [XFORM](#) structure used to modify the world transformation for the given device context.

[in] `mode`

Specifies how the transformation data modifies the current world transformation. This parameter must be one of the following values.

VALUE	MEANING
<code>MWT_IDENTITY</code>	Resets the current world transformation by using the identity matrix. If this mode is specified, the XFORM structure pointed to by <code>/pXform</code> is ignored.
<code>MWT_LEFTMULTIPLY</code>	Multiplies the current transformation by the data in the XFORM structure. (The data in the XFORM structure becomes the left multiplicand, and the data for the current transformation becomes the right multiplicand.)
<code>MWT_RIGHTMULTIPLY</code>	Multiplies the current transformation by the data in the XFORM structure. (The data in the XFORM structure becomes the right multiplicand, and the data for the current transformation becomes the left multiplicand.)

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **ModifyWorldTransform** function will fail unless graphics mode for the specified device context has been set to GM_ADVANCED by previously calling the [SetGraphicsMode](#) function. Likewise, it will not be possible to reset the graphics mode for the device context to the default GM_COMPATIBLE mode, unless world transform has first been reset to the default identity transform by calling [SetWorldTransform](#) or [ModifyWorldTransform](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetWorldTransform](#)

[SetGraphicsMode](#)

[SetWorldTransform](#)

[XFORM](#)

MoveToEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MoveToEx** function updates the current position to the specified point and optionally returns the previous position.

Syntax

```
BOOL MoveToEx(
    [in]  HDC      hdc,
    [in]  int      x,
    [in]  int      y,
    [out] LPPOINT lppt
);
```

Parameters

[in] hdc

Handle to a device context.

[in] x

Specifies the x-coordinate, in logical units, of the new position, in logical units.

[in] y

Specifies the y-coordinate, in logical units, of the new position, in logical units.

[out] lppt

Pointer to a **POINT** structure that receives the previous current position. If this parameter is a **NULL** pointer, the previous position is not returned.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **MoveToEx** function affects all drawing functions.

Examples

For an example, see [Drawing Markers](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AngleArc](#)

[Line and Curve Functions](#)

[LineTo](#)

[Lines and Curves Overview](#)

[POINT](#)

[PolyBezierTo](#)

[PolylineTo](#)

NEWTEXTMETRICA structure (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The NEWTEXTMETRIC structure contains data that describes a physical font.

Syntax

```
typedef struct tagNEWTEXTMETRICA {  
    LONG tmHeight;  
    LONG tmAscent;  
    LONG tmDescent;  
    LONG tmInternalLeading;  
    LONG tmExternalLeading;  
    LONG tmAveCharWidth;  
    LONG tmMaxCharWidth;  
    LONG tmWeight;  
    LONG tmOverhang;  
    LONG tmDigitizedAspectX;  
    LONG tmDigitizedAspectY;  
    BYTE tmFirstChar;  
    BYTE tmLastChar;  
    BYTE tmDefaultChar;  
    BYTE tmBreakChar;  
    BYTE tmItalic;  
    BYTE tmUnderlined;  
    BYTE tmStruckOut;  
    BYTE tmPitchAndFamily;  
    BYTE tmCharSet;  
    DWORD ntmFlags;  
    UINT ntmSizeEM;  
    UINT ntmCellHeight;  
    UINT ntmAvgWidth;  
} NEWTEXTMETRICA, *PNEWTEXTMETRICA, *NPNEWTEXTMETRICA, *LPNEWTEXTMETRICA;
```

Members

tmHeight

The height (ascent + descent) of characters.

tmAscent

The ascent (units above the base line) of characters.

tmDescent

The descent (units below the base line) of characters.

tmInternalLeading

The amount of leading (space) inside the bounds set by the **tmHeight** member. Accent marks and other diacritical characters may occur in this area. The designer may set this member to zero.

tmExternalLeading

The amount of extra leading (space) that the application adds between rows. Since this area is outside the font, it contains no marks and is not altered by text output calls in either OPAQUE or TRANSPARENT mode. The

designer may set this member to zero.

`tmAveCharWidth`

The average width of characters in the font (generally defined as the width of the letter x). This value does not include overhang required for bold or italic characters.

`tmMaxCharWidth`

The width of the widest character in the font.

`tmWeight`

The weight of the font.

`tmOverhang`

The extra width per string that may be added to some synthesized fonts. When synthesizing some attributes, such as bold or italic, graphics device interface (GDI) or a device may have to add width to a string on both a per-character and per-string basis. For example, GDI makes a string bold by expanding the spacing of each character and overstriking by an offset value; it italicizes a font by shearing the string. In either case, there is an overhang past the basic string. For bold strings, the overhang is the distance by which the overstrike is offset. For italic strings, the overhang is the amount the top of the font is sheared past the bottom of the font.

The `tmOverhang` member enables the application to determine how much of the character width returned by a [GetTextExtentPoint32](#) function call on a single character is the actual character width and how much is the per-string extra width. The actual width is the extent minus the overhang.

`tmDigitizedAspectX`

The horizontal aspect of the device for which the font was designed.

`tmDigitizedAspectY`

The vertical aspect of the device for which the font was designed. The ratio of the `tmDigitizedAspectX` and `tmDigitizedAspectY` members is the aspect ratio of the device for which the font was designed.

`tmFirstChar`

The value of the first character defined in the font.

`tmLastChar`

The value of the last character defined in the font.

`tmDefaultChar`

The value of the character to be substituted for characters that are not in the font.

`tmBreakChar`

The value of the character to be used to define word breaks for text justification.

`tmItalic`

An italic font if it is nonzero.

`tmUnderlined`

An underlined font if it is nonzero.

`tmStruckOut`

A strikeout font if it is nonzero.

tmPitchAndFamily

The pitch and family of the selected font. The low-order bit (bit 0) specifies the pitch of the font. If it is 1, the font is variable pitch (or proportional). If it is 0, the font is fixed pitch (or monospace). Bits 1 and 2 specify the font type. If both bits are 0, the font is a raster font; if bit 1 is 1 and bit 2 is 0, the font is a vector font; if bit 1 is 0 and bit 2 is set, or if both bits are 1, the font is some other type. Bit 3 is 1 if the font is a device font; otherwise, it is 0.

The four high-order bits designate the font family. The **tmPitchAndFamily** member can be combined with the hexadecimal value 0xF0 by using the bitwise AND operator and can then be compared with the font family names for an identical match. For more information about the font families, see [LOGFONT](#).

tmCharSet

The character set of the font.

ntmFlags

Specifies whether the font is italic, underscored, outlined, bold, and so forth. May be any reasonable combination of the following values.

BIT	NAME	MEANING
0	NTM_ITALIC	italic
5	NTM_BOLD	bold
8	NTM_REGULAR	regular
16	NTM_NONNEGATIVE_AC	no glyph in a font at any size has a negative A or C space.
17	NTM_PS_OPENTYPE	PostScript OpenType font
18	NTM_TT_OPENTYPE	TrueType OpenType font
19	NTM_MULTIPLEMMASTER	multiple master font
20	NTM_TYPE1	Type 1 font
21	NTM_DSIG	font with a digital signature. This allows traceability and ensures that the font has been tested and is not corrupted

ntmSizeEM

The size of the em square for the font. This value is in notional units (that is, the units for which the font was designed).

ntmCellHeight

The height, in notional units, of the font. This value should be compared with the value of the **ntmSizeEM** member.

ntmAvgWidth

The average width of characters in the font, in notional units. This value should be compared with the value of

the **ntmSizeEM** member.

Remarks

The last four members of the **NEWTEXTMETRIC** structure are not included in the **TEXTMETRIC** structure; in all other respects, the structures are identical.

The sizes in the **NEWTEXTMETRIC** structure are typically specified in logical units; that is, they depend on the current mapping mode of the display context.

NOTE

The wingdi.h header defines **NEWTEXTMETRIC** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EnumFontFamilies](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

[LOGFONT](#)

NEWTEXTMETRICEXA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The NEWTEXTMETRICEX structure contains information about a physical font.

Syntax

```
typedef struct tagNEWTEXTMETRICEXA {  
    NEWTEXTMETRICA ntmTm;  
    FONTSIGNATURE ntmFontSig;  
} NEWTEXTMETRICEXA;
```

Members

ntmTm

A [NEWTEXTMETRIC](#) structure.

ntmFontSig

A [FONTSIGNATURE](#) structure indicating the coverage of the font.

Remarks

NOTE

The wingdi.h header defines NEWTEXTMETRICEX as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[FONTSIGNATURE](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[NEWTEXTMETRIC](#)

NEWTEXTMETRICEXW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The NEWTEXTMETRICEX structure contains information about a physical font.

Syntax

```
typedef struct tagNEWTEXTMETRICEXW {
    NEWTEXTMETRICW ntmTm;
    FONTSIGNATURE  ntmFontSig;
} NEWTEXTMETRICEXW;
```

Members

ntmTm

A [NEWTEXTMETRIC](#) structure.

ntmFontSig

A [FONTSIGNATURE](#) structure indicating the coverage of the font.

Remarks

NOTE

The wingdi.h header defines NEWTEXTMETRICEX as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[FONTSIGNATURE](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[NEWTEXTMETRIC](#)

NEWTEXTMETRICW structure (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The NEWTEXTMETRIC structure contains data that describes a physical font.

Syntax

```
typedef struct tagNEWTEXTMETRICW {
    LONG tmHeight;
    LONG tmAscent;
    LONG tmDescent;
    LONG tmInternalLeading;
    LONG tmExternalLeading;
    LONG tmAveCharWidth;
    LONG tmMaxCharWidth;
    LONG tmWeight;
    LONG tmOverhang;
    LONG tmDigitizedAspectX;
    LONG tmDigitizedAspectY;
    WCHAR tmFirstChar;
    WCHAR tmLastChar;
    WCHAR tmDefaultChar;
    WCHAR tmBreakChar;
    BYTE tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    DWORD ntmFlags;
    UINT ntmSizeEM;
    UINT ntmCellHeight;
    UINT ntmAvgWidth;
} NEWTEXTMETRICW, *PNEWTEXTMETRICW, *NPNEWTEXTMETRICW, *LPNEWTEXTMETRICW;
```

Members

tmHeight

The height (ascent + descent) of characters.

tmAscent

The ascent (units above the base line) of characters.

tmDescent

The descent (units below the base line) of characters.

tmInternalLeading

The amount of leading (space) inside the bounds set by the **tmHeight** member. Accent marks and other diacritical characters may occur in this area. The designer may set this member to zero.

tmExternalLeading

The amount of extra leading (space) that the application adds between rows. Since this area is outside the font, it contains no marks and is not altered by text output calls in either OPAQUE or TRANSPARENT mode. The

designer may set this member to zero.

`tmAveCharWidth`

The average width of characters in the font (generally defined as the width of the letter x). This value does not include overhang required for bold or italic characters.

`tmMaxCharWidth`

The width of the widest character in the font.

`tmWeight`

The weight of the font.

`tmOverhang`

The extra width per string that may be added to some synthesized fonts. When synthesizing some attributes, such as bold or italic, graphics device interface (GDI) or a device may have to add width to a string on both a per-character and per-string basis. For example, GDI makes a string bold by expanding the spacing of each character and overstriking by an offset value; it italicizes a font by shearing the string. In either case, there is an overhang past the basic string. For bold strings, the overhang is the distance by which the overstrike is offset. For italic strings, the overhang is the amount the top of the font is sheared past the bottom of the font.

The `tmOverhang` member enables the application to determine how much of the character width returned by a [GetTextExtentPoint32](#) function call on a single character is the actual character width and how much is the per-string extra width. The actual width is the extent minus the overhang.

`tmDigitizedAspectX`

The horizontal aspect of the device for which the font was designed.

`tmDigitizedAspectY`

The vertical aspect of the device for which the font was designed. The ratio of the `tmDigitizedAspectX` and `tmDigitizedAspectY` members is the aspect ratio of the device for which the font was designed.

`tmFirstChar`

The value of the first character defined in the font.

`tmLastChar`

The value of the last character defined in the font.

`tmDefaultChar`

The value of the character to be substituted for characters that are not in the font.

`tmBreakChar`

The value of the character to be used to define word breaks for text justification.

`tmItalic`

An italic font if it is nonzero.

`tmUnderlined`

An underlined font if it is nonzero.

`tmStruckOut`

A strikeout font if it is nonzero.

tmPitchAndFamily

The pitch and family of the selected font. The low-order bit (bit 0) specifies the pitch of the font. If it is 1, the font is variable pitch (or proportional). If it is 0, the font is fixed pitch (or monospace). Bits 1 and 2 specify the font type. If both bits are 0, the font is a raster font; if bit 1 is 1 and bit 2 is 0, the font is a vector font; if bit 1 is 0 and bit 2 is set, or if both bits are 1, the font is some other type. Bit 3 is 1 if the font is a device font; otherwise, it is 0.

The four high-order bits designate the font family. The **tmPitchAndFamily** member can be combined with the hexadecimal value 0xF0 by using the bitwise AND operator and can then be compared with the font family names for an identical match. For more information about the font families, see [LOGFONT](#).

tmCharSet

The character set of the font.

ntmFlags

Specifies whether the font is italic, underscored, outlined, bold, and so forth. May be any reasonable combination of the following values.

BIT	NAME	MEANING
0	NTM_ITALIC	italic
5	NTM_BOLD	bold
8	NTM_REGULAR	regular
16	NTM_NONNEGATIVE_AC	no glyph in a font at any size has a negative A or C space.
17	NTM_PS_OPENTYPE	PostScript OpenType font
18	NTM_TT_OPENTYPE	TrueType OpenType font
19	NTM_MULTIPLEMMASTER	multiple master font
20	NTM_TYPE1	Type 1 font
21	NTM_DSIG	font with a digital signature. This allows traceability and ensures that the font has been tested and is not corrupted

ntmSizeEM

The size of the em square for the font. This value is in notional units (that is, the units for which the font was designed).

ntmCellHeight

The height, in notional units, of the font. This value should be compared with the value of the **ntmSizeEM** member.

ntmAvgWidth

The average width of characters in the font, in notional units. This value should be compared with the value of

the **ntmSizeEM** member.

Remarks

The last four members of the **NEWTEXTMETRIC** structure are not included in the **TEXTMETRIC** structure; in all other respects, the structures are identical.

The sizes in the **NEWTEXTMETRIC** structure are typically specified in logical units; that is, they depend on the current mapping mode of the display context.

NOTE

The wingdi.h header defines **NEWTEXTMETRIC** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EnumFontFamilies](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

[LOGFONT](#)

OffsetClipRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **OffsetClipRgn** function moves the clipping region of a device context by the specified offsets.

Syntax

```
int OffsetClipRgn(
    [in] HDC hdc,
    [in] int x,
    [in] int y
);
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **x**

The number of logical units to move left or right.

[in] **y**

The number of logical units to move up or down.

Return value

The return value specifies the new region's complexity and can be one of the following values.

RETURN CODE	DESCRIPTION
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred. (The current clipping region is unaffected.)

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Clipping Functions](#)

[Clipping Overview](#)

[SelectClipRgn](#)

OffsetRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **OffsetRgn** function moves a region by the specified offsets.

Syntax

```
int OffsetRgn(
    [in] HRGN hrgn,
    [in] int x,
    [in] int y
);
```

Parameters

[in] **hrgn**

Handle to the region to be moved.

[in] **x**

Specifies the number of logical units to move left or right.

[in] **y**

Specifies the number of logical units to move up or down.

Return value

The return value specifies the new region's complexity. It can be one of the following values.

VALUE	MEANING
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred; region is unaffected.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Region Functions](#)

[Regions Overview](#)

OffsetViewportOrgEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **OffsetViewportOrgEx** function modifies the viewport origin for a device context using the specified horizontal and vertical offsets.

Syntax

```
BOOL OffsetViewportOrgEx(
    [in] HDC     hdc,
    [in] int     x,
    [in] int     y,
    [out] LPPOINT lppt
);
```

Parameters

[in] hdc

A handle to the device context.

[in] x

The horizontal offset, in device units.

[in] y

The vertical offset, in device units.

[out] lppt

A pointer to a **POINT** structure. The previous viewport origin, in device units, is placed in this structure. If *lpPoint* is **NULL**, the previous viewport origin is not returned.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The new origin is the sum of the current origin and the horizontal and vertical offsets.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetViewportOrgEx](#)

[OffsetWindowOrgEx](#)

[SetViewportOrgEx](#)

OffsetWindowOrgEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **OffsetWindowOrgEx** function modifies the window origin for a device context using the specified horizontal and vertical offsets.

Syntax

```
BOOL OffsetWindowOrgEx(
    [in]  HDC      hdc,
    [in]  int      x,
    [in]  int      y,
    [out] LPPOINT lppt
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The horizontal offset, in logical units.

[in] `y`

The vertical offset, in logical units.

[out] `lppt`

A pointer to a [POINT](#) structure. The logical coordinates of the previous window origin are placed in this structure. If *lpPoint* is **NULL**, the previous origin is not returned.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetViewportOrgEx](#)

[OffsetViewportOrgEx](#)

[POINT](#)

OUTLINETEXTMETRICA structure (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The OUTLINETEXTMETRIC structure contains metrics describing a TrueType font.

Syntax

```
typedef struct _OUTLINETEXTMETRICA {
    UINT          otmSize;
    TEXTMETRICA  otmTextMetrics;
    BYTE          otmFiller;
    PANOSE        otmPanoseNumber;
    UINT          otmfsSelection;
    UINT          otmfsType;
    int           otmsCharSlopeRise;
    int           otmsCharSlopeRun;
    int           otmItalicAngle;
    UINT          otmEMSquare;
    int           otmAscent;
    int           otmDescent;
    UINT          otmLineGap;
    UINT          otmsCapEmHeight;
    UINT          otmsXHeight;
    RECT          otmrcFontBox;
    int           otmMacAscent;
    int           otmMacDescent;
    UINT          otmMacLineGap;
    UINT          otmusMinimumPPEM;
    POINT         otmptSubscriptSize;
    POINT         otmptSubscriptOffset;
    POINT         otmptSuperscriptSize;
    POINT         otmptSuperscriptOffset;
    UINT          otmsStrikeoutSize;
    int           otmsStrikeoutPosition;
    int           otmsUnderscoreSize;
    int           otmsUnderscorePosition;
    PSTR          otmpFamilyName;
    PSTR          otmpFaceName;
    PSTR          otmpStyleName;
    PSTR          otmpFullName;
} OUTLINETEXTMETRICA, *POUTLINETEXTMETRICA, *NPOUTLINETEXTMETRICA, *LPOUTLINETEXTMETRICA;
```

Members

`otmSize`

The size, in bytes, of the OUTLINETEXTMETRIC structure.

`otmTextMetrics`

A [TEXTMETRIC](#) structure containing further information about the font.

`otmFiller`

A value that causes the structure to be byte-aligned.

`otmPanoseNumber`

The PANOSE number for this font.

otmfsSelection

The nature of the font pattern. This member can be a combination of the following bits.

BIT	MEANING
0	Italic
1	Underscore
2	Negative
3	Outline
4	Strikeout
5	Bold

otmfsType

Indicates whether the font is licensed. Licensed fonts must not be modified or exchanged. If bit 1 is set, the font may not be embedded in a document. If bit 1 is clear, the font can be embedded. If bit 2 is set, the embedding is read-only.

otmsCharSlopeRise

The slope of the cursor. This value is 1 if the slope is vertical. Applications can use this value and the value of the **otmsCharSlopeRun** member to create an italic cursor that has the same slope as the main italic angle (specified by the **otmItalicAngle** member).

otmsCharSlopeRun

The slope of the cursor. This value is zero if the slope is vertical. Applications can use this value and the value of the **otmsCharSlopeRise** member to create an italic cursor that has the same slope as the main italic angle (specified by the **otmItalicAngle** member).

otmItalicAngle

The main italic angle of the font, in tenths of a degree counterclockwise from vertical. Regular (roman) fonts have a value of zero. Italic fonts typically have a negative italic angle (that is, they lean to the right).

otmEMSquare

The number of logical units defining the x- or y-dimension of the em square for this font. (The number of units in the x- and y-directions are always the same for an em square.)

otmAscent

The maximum distance characters in this font extend above the base line. This is the typographic ascent for the font.

otmDescent

The maximum distance characters in this font extend below the base line. This is the typographic descent for the font.

otmLineGap

The typographic line spacing.

`otmsCapEmHeight`

Not supported.

`otmsXHeight`

Not supported.

`otmrcFontBox`

The bounding box for the font.

`otmMacAscent`

The maximum distance characters in this font extend above the base line for the Macintosh computer.

`otmMacDescent`

The maximum distance characters in this font extend below the base line for the Macintosh computer.

`otmMacLineGap`

The line-spacing information for the Macintosh computer.

`otmusMinimumPPEM`

The smallest recommended size for this font, in pixels per em-square.

`otmpSubscriptSize`

The recommended horizontal and vertical size for subscripts in this font.

`otmpSubscriptOffset`

The recommended horizontal and vertical offset for subscripts in this font. The subscript offset is measured from the character origin to the origin of the subscript character.

`otmpSuperscriptSize`

The recommended horizontal and vertical size for superscripts in this font.

`otmpSuperscriptOffset`

The recommended horizontal and vertical offset for superscripts in this font. The superscript offset is measured from the character base line to the base line of the superscript character.

`otmsStrikeoutSize`

The width of the strikeout stroke for this font. Typically, this is the width of the em dash for the font.

`otmsStrikeoutPosition`

The position of the strikeout stroke relative to the base line for this font. Positive values are above the base line and negative values are below.

`otmsUnderscoreSize`

The thickness of the underscore character for this font.

`otmsUnderscorePosition`

The position of the underscore character for this font.

`otmpFamilyName`

The offset from the beginning of the structure to a string specifying the family name for the font.

`otmpFaceName`

The offset from the beginning of the structure to a string specifying the typeface name for the font. (This typeface name corresponds to the name specified in the [LOGFONT](#) structure.)

`otmpStyleName`

The offset from the beginning of the structure to a string specifying the style name for the font.

`otmpFullName`

The offset from the beginning of the structure to a string specifying the full name for the font. This name is unique for the font and often contains a version number or other identifying information.

Remarks

The sizes returned in [OUTLINETEXTMETRIC](#) are specified in logical units; that is, they depend on the current mapping mode of the specified display context.

Note, [OUTLINETEXTMETRIC](#) is defined using the current pack setting. To avoid problems, make sure that the application is built using the platform default packing. For example, 32-bit Windows uses a default of 8-byte packing. For more information, see the MSDN topic "C-Compiler Packing Issues".

NOTE

The wingdi.h header defines [OUTLINETEXTMETRIC](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the `UNICODE` preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetOutlineTextMetrics](#)

[LOGFONT](#)

[TEXTMETRIC](#)

OUTLINETEXTMETRICW structure (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The OUTLINETEXTMETRIC structure contains metrics describing a TrueType font.

Syntax

```
typedef struct _OUTLINETEXTMETRICW {
    UINT          otmSize;
    TEXTMETRICW  otmTextMetrics;
    BYTE          otmFiller;
    PANOSE        otmPanoseNumber;
    UINT          otmfsSelection;
    UINT          otmfsType;
    int           otmsCharSlopeRise;
    int           otmsCharSlopeRun;
    int           otmItalicAngle;
    UINT          otmEMSquare;
    int           otmAscent;
    int           otmDescent;
    UINT          otmLineGap;
    UINT          otmsCapEmHeight;
    UINT          otmsXHeight;
    RECT          otmrcFontBox;
    int           otmMacAscent;
    int           otmMacDescent;
    UINT          otmMacLineGap;
    UINT          otmusMinimumPPEM;
    POINT         otmptSubscriptSize;
    POINT         otmptSubscriptOffset;
    POINT         otmptSuperscriptSize;
    POINT         otmptSuperscriptOffset;
    UINT          otmsStrikeoutSize;
    int           otmsStrikeoutPosition;
    int           otmsUnderscoreSize;
    int           otmsUnderscorePosition;
    PSTR          otmpFamilyName;
    PSTR          otmpFaceName;
    PSTR          otmpStyleName;
    PSTR          otmpFullName;
} OUTLINETEXTMETRICW, *POUTLINETEXTMETRICW, *NPOUTLINETEXTMETRICW, *LPOUTLINETEXTMETRICW;
```

Members

`otmSize`

The size, in bytes, of the OUTLINETEXTMETRIC structure.

`otmTextMetrics`

A [TEXTMETRIC](#) structure containing further information about the font.

`otmFiller`

A value that causes the structure to be byte-aligned.

`otmPanoseNumber`

The PANOSE number for this font.

otmfsSelection

The nature of the font pattern. This member can be a combination of the following bits.

BIT	MEANING
0	Italic
1	Underscore
2	Negative
3	Outline
4	Strikeout
5	Bold

otmfsType

Indicates whether the font is licensed. Licensed fonts must not be modified or exchanged. If bit 1 is set, the font may not be embedded in a document. If bit 1 is clear, the font can be embedded. If bit 2 is set, the embedding is read-only.

otmsCharSlopeRise

The slope of the cursor. This value is 1 if the slope is vertical. Applications can use this value and the value of the **otmsCharSlopeRun** member to create an italic cursor that has the same slope as the main italic angle (specified by the **otmItalicAngle** member).

otmsCharSlopeRun

The slope of the cursor. This value is zero if the slope is vertical. Applications can use this value and the value of the **otmsCharSlopeRise** member to create an italic cursor that has the same slope as the main italic angle (specified by the **otmItalicAngle** member).

otmItalicAngle

The main italic angle of the font, in tenths of a degree counterclockwise from vertical. Regular (roman) fonts have a value of zero. Italic fonts typically have a negative italic angle (that is, they lean to the right).

otmEMSquare

The number of logical units defining the x- or y-dimension of the em square for this font. (The number of units in the x- and y-directions are always the same for an em square.)

otmAscent

The maximum distance characters in this font extend above the base line. This is the typographic ascent for the font.

otmDescent

The maximum distance characters in this font extend below the base line. This is the typographic descent for the font.

otmLineGap

The typographic line spacing.

`otmsCapEmHeight`

Not supported.

`otmsXHeight`

Not supported.

`otmrcFontBox`

The bounding box for the font.

`otmMacAscent`

The maximum distance characters in this font extend above the base line for the Macintosh computer.

`otmMacDescent`

The maximum distance characters in this font extend below the base line for the Macintosh computer.

`otmMacLineGap`

The line-spacing information for the Macintosh computer.

`otmusMinimumPPEM`

The smallest recommended size for this font, in pixels per em-square.

`otmpSubscriptSize`

The recommended horizontal and vertical size for subscripts in this font.

`otmpSubscriptOffset`

The recommended horizontal and vertical offset for subscripts in this font. The subscript offset is measured from the character origin to the origin of the subscript character.

`otmpSuperscriptSize`

The recommended horizontal and vertical size for superscripts in this font.

`otmpSuperscriptOffset`

The recommended horizontal and vertical offset for superscripts in this font. The superscript offset is measured from the character base line to the base line of the superscript character.

`otmsStrikeoutSize`

The width of the strikeout stroke for this font. Typically, this is the width of the em dash for the font.

`otmsStrikeoutPosition`

The position of the strikeout stroke relative to the base line for this font. Positive values are above the base line and negative values are below.

`otmsUnderscoreSize`

The thickness of the underscore character for this font.

`otmsUnderscorePosition`

The position of the underscore character for this font.

`otmpFamilyName`

The offset from the beginning of the structure to a string specifying the family name for the font.

`otmpFaceName`

The offset from the beginning of the structure to a string specifying the typeface name for the font. (This typeface name corresponds to the name specified in the [LOGFONT](#) structure.)

`otmpStyleName`

The offset from the beginning of the structure to a string specifying the style name for the font.

`otmpFullName`

The offset from the beginning of the structure to a string specifying the full name for the font. This name is unique for the font and often contains a version number or other identifying information.

Remarks

The sizes returned in [OUTLINETEXTMETRIC](#) are specified in logical units; that is, they depend on the current mapping mode of the specified display context.

Note, [OUTLINETEXTMETRIC](#) is defined using the current pack setting. To avoid problems, make sure that the application is built using the platform default packing. For example, 32-bit Windows uses a default of 8-byte packing. For more information, see the MSDN topic "C-Compiler Packing Issues".

NOTE

The wingdi.h header defines [OUTLINETEXTMETRIC](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the `UNICODE` preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetOutlineTextMetrics](#)

[LOGFONT](#)

[TEXTMETRIC](#)

PaintRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PaintRgn** function paints the specified region by using the brush currently selected into the device context.

Syntax

```
BOOL PaintRgn(
    [in] HDC hdc,
    [in] HRGN hrgn
);
```

Parameters

[in] **hdc**

Handle to the device context.

[in] **hrgn**

Handle to the region to be filled. The region's coordinates are presumed to be logical coordinates.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[FillRgn](#)

[Region Functions](#)

[Regions Overview](#)

PALETTEINDEX macro (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PALETTEINDEX** macro accepts an index to a logical-color palette entry and returns a palette-entry specifier consisting of a [COLORREF](#) value that specifies the color associated with the given index. An application using a logical palette can pass this specifier, instead of an explicit red, green, blue (RGB) value, to GDI functions that expect a color. This allows the function to use the color in the specified palette entry.

Syntax

```
void PALETTEINDEX(  
    i  
);
```

Parameters

i

An index to the palette entry containing the color to be used for a graphics operation.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[COLORREF](#)

[Color Macros](#)

[Colors Overview](#)

[PALETTERGB](#)

[RGB](#)

PALETTERRGB macro (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PALETTERRGB** macro accepts three values that represent the relative intensities of red, green, and blue and returns a palette-relative red, green, blue (RGB) specifier consisting of 2 in the high-order byte and an RGB value in the three low-order bytes. An application using a color palette can pass this specifier, instead of an explicit RGB value, to functions that expect a color.

Syntax

```
void PALETTERRGB(  
    r,  
    g,  
    b  
)
```

Parameters

r

The intensity of the red color field.

g

The intensity of the green color field.

b

The intensity of the blue color field.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[COLORREF](#)

[Color Macros](#)

Colors Overview

PALETTEINDEX

RGB

PANOSE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PANOSE** structure describes the PANOSE font-classification values for a TrueType font. These characteristics are then used to associate the font with other fonts of similar appearance but different names.

Syntax

```
typedef struct tagPANOSE {
    BYTE bFamilyType;
    BYTE bSerifStyle;
    BYTE bWeight;
    BYTE bProportion;
    BYTE bContrast;
    BYTE bStrokeVariation;
    BYTE bArmStyle;
    BYTE bLetterform;
    BYTE bMidline;
    BYTE bXHeight;
} PANOSE, *LPPANOSE;
```

Members

bFamilyType

For Latin fonts, one of one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_FAMILY_TEXT_DISPLAY	Text and display
PAN_FAMILY_SCRIPT	Script
PAN_FAMILY_DECORATIVE	Decorative
PAN_FAMILY_PICTORIAL	Pictorial

bSerifStyle

The serif style. For Latin fonts, one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit

PAN_SERIF_COVE	Cove
PAN_SERIF_OBTUSE_COVE	Obtuse cove
PAN_SERIF_SQUARE_COVE	Square cove
PAN_SERIF_OBTUSE_SQUARE_COVE	Obtuse square cove
PAN_SERIF_SQUARE	Square
PAN_SERIF_THIN	Thin
PAN_SERIF_BONE	Bone
PAN_SERIF_EXAGGERATED	Exaggerated
PAN_SERIF_TRIANGLE	Triangle
PAN_SERIF_NORMAL_SANS	Normal sans serif
PAN_SERIF_OBTUSE_SANS	Obtuse sans serif
PAN_SERIF_PERP_SANS	Perp sans serif
PAN_SERIF_FLARED	Flared
PAN_SERIF_ROUNDED	Rounded

bWeight

For Latin fonts, one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_WEIGHT_VERY_LIGHT	Very light
PAN_WEIGHT_LIGHT	Light
PAN_WEIGHT_THIN	Thin
PAN_WEIGHT_BOOK	Book
PAN_WEIGHT_MEDIUM	Medium
PAN_WEIGHT_DEMI	Demibold
PAN_WEIGHT_BOLD	Bold
PAN_WEIGHT_HEAVY	Heavy

PAN_WEIGHT_BLACK	Black
PAN_WEIGHT_NORD	Nord

bProportion

For Latin fonts, one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_PROP_OLD_STYLE	Old style
PAN_PROP_MODERN	Modern
PAN_PROP_EVEN_WIDTH	Even width
PAN_PROP_EXPANDED	Expanded
PAN_PROP_CONDENSED	Condensed
PAN_PROP_VERY_EXPANDED	Very expanded
PAN_PROP_VERY_CONDENSED	Very condensed
PAN_PROP_MONOSPACED	Monospaced

bContrast

For Latin fonts, one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_CONTRAST_NONE	None
PAN_CONTRAST_VERY_LOW	Very low
PAN_CONTRAST_LOW	Low
PAN_CONTRAST_MEDIUM_LOW	Medium low
PAN_CONTRAST_MEDIUM	Medium
PAN_CONTRAST_MEDIUM_HIGH	Medium high
PAN_CONTRAST_HIGH	High

PAN_CONTRAST VERY HIGH	Very high
------------------------	-----------

bStrokeVariation

For Latin fonts, one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_STROKE_GRADUAL_DIAG	Gradual/diagonal
PAN_STROKE_GRADUAL_TRAN	Gradual/transitional
PAN_STROKE_GRADUAL_VERT	Gradual/vertical
PAN_STROKE_GRADUAL_HORZ	Gradual/horizontal
PAN_STROKE_RAPID_VERT	Rapid/vertical
PAN_STROKE_RAPID_HORZ	Rapid/horizontal
PAN_STROKE_INSTANT_VERT	Instant/vertical

bArmStyle

For Latin fonts, one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_STRAIGHT_ARMS_HORZ	Straight arms/horizontal
PAN_STRAIGHT_ARMS_WEDGE	Straight arms/wedge
PAN_STRAIGHT_ARMS_VERT	Straight arms/vertical
PAN_STRAIGHT_ARMS_SINGLE_SERIF	Straight arms/single-serif
PAN_STRAIGHT_ARMS_DOUBLE_SERIF	Straight arms/double-serif
PAN_BENT_ARMS_HORZ	Nonstraight arms/horizontal
PAN_BENT_ARMS_WEDGE	Nonstraight arms/wedge
PAN_BENT_ARMS_VERT	Nonstraight arms/vertical
PAN_BENT_ARMS_SINGLE_SERIF	Nonstraight arms/single-serif

PAN_BENT_ARMS_DOUBLE_SERIF	Nonstraight arms/double-serif
----------------------------	-------------------------------

bLetterform

For Latin fonts, one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_LETT_NORMAL_CONTACT	Normal/contact
PAN_LETT_NORMAL_WEIGHTED	Normal/weighted
PAN_LETT_NORMAL_BOXED	Normal/boxed
PAN_LETT_NORMAL_FLATTENED	Normal/flattened
PAN_LETT_NORMAL_ROUNDED	Normal/rounded
PAN_LETT_NORMAL_OFF_CENTER	Normal/off center
PAN_LETT_NORMAL_SQUARE	Normal/square
PAN_LETT_OBLIQUE_CONTACT	Oblique/contact
PAN_LETT_OBLIQUE_WEIGHTED	Oblique/weighted
PAN_LETT_OBLIQUE_BOXED	Oblique/boxed
PAN_LETT_OBLIQUE_FLATTENED	Oblique/flattened
PAN_LETT_OBLIQUE_ROUNDED	Oblique/rounded
PAN_LETT_OBLIQUE_OFF_CENTER	Oblique/off center
PAN_LETT_OBLIQUE_SQUARE	Oblique/square

bMidline

For Latin fonts, one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_MIDLINE_STANDARD_TRIMMED	Standard/trimmed
PAN_MIDLINE_STANDARD_POINTED	Standard/pointed

PAN_MIDLIN_STANDARD_SERIFED	Standard/serifed
PAN_MIDLIN_HIGH_TRIMMED	High/trimmed
PAN_MIDLIN_HIGH_POINTED	High/pointed
PAN_MIDLIN_HIGH_SERIFED	High/serifed
PAN_MIDLIN_CONSTANT_TRIMMED	Constant/trimmed
PAN_MIDLIN_CONSTANT_POINTED	Constant/pointed
PAN_MIDLIN_CONSTANT_SERIFED	Constant/serifed
PAN_MIDLIN_LOW_TRIMMED	Low/trimmed
PAN_MIDLIN_LOW_POINTED	Low/pointed
PAN_MIDLIN_LOW_SERIFED	Low/serifed

bXHeight

For Latin fonts, one of the following values.

VALUE	MEANING
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_XHEIGHT_CONSTANT_SMALL	Constant/small
PAN_XHEIGHT_CONSTANT_STD	Constant/standard
PAN_XHEIGHT_CONSTANT_LARGE	Constant/large
PAN_XHEIGHT_DUCKING_SMALL	Ducking/small
PAN_XHEIGHT_DUCKING_STD	Ducking/standard
PAN_XHEIGHT_DUCKING_LARGE	Ducking/large

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EXTLOGFONT](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[OUTLINETEXTMETRIC](#)

PatBlt function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PatBlt** function paints the specified rectangle using the brush that is currently selected into the specified device context. The brush color and the surface color or colors are combined by using the specified raster operation.

Syntax

```
BOOL PatBlt(
    [in] HDC    hdc,
    [in] int     x,
    [in] int     y,
    [in] int     w,
    [in] int     h,
    [in] DWORD   rop
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The x-coordinate, in logical units, of the upper-left corner of the rectangle to be filled.

[in] `y`

The y-coordinate, in logical units, of the upper-left corner of the rectangle to be filled.

[in] `w`

The width, in logical units, of the rectangle.

[in] `h`

The height, in logical units, of the rectangle.

[in] `rop`

The raster operation code. This code can be one of the following values.

VALUE	MEANING
PATCOPY	Copies the specified pattern into the destination bitmap.
PATINVERT	Combines the colors of the specified pattern with the colors of the destination rectangle by using the Boolean XOR operator.

DSTINVERT	Inverts the destination rectangle.
BLACKNESS	Fills the destination rectangle using the color associated with index 0 in the physical palette. (This color is black for the default physical palette.)
WHITENESS	Fills the destination rectangle using the color associated with index 1 in the physical palette. (This color is white for the default physical palette.)

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The values of the *dwRop* parameter for this function are a limited subset of the full 256 ternary raster-operation codes; in particular, an operation code that refers to a source rectangle cannot be used.

Not all devices support the **PatBlt** function. For more information, see the description of the RC_BITBLT capability in the [GetDeviceCaps](#) function.

Examples

For an example, see "Example of Menu-Item Bitmaps" in [Using Menus](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Brush Functions](#)

[Brushes Overview](#)

[GetDeviceCaps](#)

PathToRegion function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PathToRegion** function creates a region from the path that is selected into the specified device context. The resulting region uses device coordinates.

Syntax

```
HRGN PathToRegion(  
    [in] HDC hdc  
>);
```

Parameters

[in] *hdc*

Handle to a device context that contains a closed path.

Return value

If the function succeeds, the return value identifies a valid region.

If the function fails, the return value is zero.

Remarks

When you no longer need the **HRGN** object call the [DeleteObject](#) function to delete it.

The device context identified by the *hdc* parameter must contain a closed path.

After **PathToRegion** converts a path into a region, the system discards the closed path from the specified device context.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

Pie function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **Pie** function draws a pie-shaped wedge bounded by the intersection of an ellipse and two radials. The pie is outlined by using the current pen and filled by using the current brush.

Syntax

```
BOOL Pie(
    [in] HDC hdc,
    [in] int left,
    [in] int top,
    [in] int right,
    [in] int bottom,
    [in] int xr1,
    [in] int yr1,
    [in] int xr2,
    [in] int yr2
);
```

Parameters

[in] hdc

A handle to the device context.

[in] left

The x-coordinate, in logical coordinates, of the upper-left corner of the bounding rectangle.

[in] top

The y-coordinate, in logical coordinates, of the upper-left corner of the bounding rectangle.

[in] right

The x-coordinate, in logical coordinates, of the lower-right corner of the bounding rectangle.

[in] bottom

The y-coordinate, in logical coordinates, of the lower-right corner of the bounding rectangle.

[in] xr1

The x-coordinate, in logical coordinates, of the endpoint of the first radial.

[in] yr1

The y-coordinate, in logical coordinates, of the endpoint of the first radial.

[in] xr2

The x-coordinate, in logical coordinates, of the endpoint of the second radial.

[in] yr2

The y-coordinate, in logical coordinates, of the endpoint of the second radial.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The curve of the pie is defined by an ellipse that fits the specified bounding rectangle. The curve begins at the point where the ellipse intersects the first radial and extends counterclockwise to the point where the ellipse intersects the second radial.

The current position is neither used nor updated by the **Pie** function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AngleArc](#)

[Arc](#)

[ArcTo](#)

[Chord](#)

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

PlayEnhMetaFile function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PlayEnhMetaFile** function displays the picture stored in the specified enhanced-format metafile.

Syntax

```
BOOL PlayEnhMetaFile(
    [in] HDC         hdc,
    [in] HENHMETAFILE hmf,
    [in] const RECT  *lprect
);
```

Parameters

[in] `hdc`

A handle to the device context for the output device on which the picture will appear.

[in] `hmf`

A handle to the enhanced metafile.

[in] `lprect`

A pointer to a [RECT](#) structure that contains the coordinates of the bounding rectangle used to display the picture. The coordinates are specified in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

When an application calls the **PlayEnhMetaFile** function, the system uses the picture frame in the enhanced-metafile header to map the picture onto the rectangle pointed to by the *lprect* parameter. (This picture may be sheared or rotated by setting the world transform in the output device before calling **PlayEnhMetaFile**.) Points along the edges of the rectangle are included in the picture.

An enhanced-metafile picture can be clipped by defining the clipping region in the output device before playing the enhanced metafile.

If an enhanced metafile contains an optional palette, an application can achieve consistent colors by setting up a color palette on the output device before calling **PlayEnhMetaFile**. To retrieve the optional palette, use the [GetEnhMetaFilePaletteEntries](#) function.

An enhanced metafile can be embedded in a newly created enhanced metafile by calling **PlayEnhMetaFile** and playing the source enhanced metafile into the device context for the new enhanced metafile.

The states of the output device context are preserved by this function. Any object created but not deleted in the enhanced metafile is deleted by this function.

To stop this function, an application can call the [CancelDC](#) function from another thread to terminate the operation. In this case, the function returns **FALSE**.

Examples

For an example, see [Opening an Enhanced Metafile and Displaying Its Contents](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CancelDC](#)

[GetEnhMetaFileHeader](#)

[GetEnhMetaFilePaletteEntries](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[RECT](#)

PlayEnhMetaFileRecord function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PlayEnhMetaFileRecord** function plays an enhanced-metafile record by executing the graphics device interface (GDI) functions identified by the record.

Syntax

```
BOOL PlayEnhMetaFileRecord(
    [in] HDC             hdc,
    [in] LPHANDLETABLE  pht,
    [in] const ENHMETARECORD *pmr,
    [in] UINT            cht
);
```

Parameters

[in] *hdc*

A handle to the device context passed to the [EnumEnhMetaFile](#) function.

[in] *pht*

A pointer to a table of handles to GDI objects used when playing the metafile. The first entry in this table contains the enhanced-metafile handle.

[in] *pmr*

A pointer to the enhanced-metafile record to be played.

[in] *cht*

The number of handles in the handle table.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

This is an enhanced-metafile function.

An application typically uses **PlayEnhMetaFileRecord** in conjunction with the [EnumEnhMetaFile](#) function to process and play an enhanced-format metafile one record at a time.

The *hdc*, *lpHandleTable*, and *nHandles* parameters must be exactly those passed to the [EnhMetaFileProc](#) callback procedure by the [EnumEnhMetaFile](#) function.

If **PlayEnhMetaFileRecord** does not recognize a record, it ignores the record and returns TRUE.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnumEnhMetaFile](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayEnhMetaFile](#)

PlayMetaFile function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PlayMetaFile** function displays the picture stored in the given Windows-format metafile on the specified device.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [PlayEnhMetaFile](#).

Syntax

```
BOOL PlayMetaFile(
    [in] HDC      hdc,
    [in] HMETAFILE hmf
);
```

Parameters

[in] `hdc`

Handle to a device context.

[in] `hmf`

Handle to a Windows-format metafile.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

To convert a Windows-format metafile into an enhanced format metafile, use the [SetWinMetaFileBits](#) function.

A Windows-format metafile can be played multiple times.

A Windows-format metafile can be embedded in a second Windows-format metafile by calling the **PlayMetaFile** function and playing the source metafile into the device context for the target metafile.

Any object created but not deleted in the Windows-format metafile is deleted by this function.

To stop this function, an application can call the [CancelDC](#) function from another thread to terminate the operation. In this case, the function returns **FALSE**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CancelDC](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SetWinMetaFileBits](#)

PlayMetaFileRecord function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PlayMetaFileRecord** function plays a Windows-format metafile record by executing the graphics device interface (GDI) function contained within that record.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [PlayEnhMetaFileRecord](#).

Syntax

```
BOOL PlayMetaFileRecord(
    [in] HDC         hdc,
    [in] LPHANDLETABLE lpHandleTable,
    [in] LPMETARECORD lpMR,
    [in] UINT        noObjs
);
```

Parameters

[in] `hdc`

A handle to a device context.

[in] `lpHandleTable`

A pointer to a [HANDLETABLE](#) structure representing the table of handles to GDI objects used when playing the metafile.

[in] `lpMR`

A pointer to the Windows-format metafile record.

[in] `noObjs`

The number of handles in the handle table.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

To convert a Windows-format metafile into an enhanced-format metafile, use the [SetWinMetaFileBits](#) function.

An application typically uses **PlayMetaFileRecord** in conjunction with the [EnumMetaFile](#) function to process and play a Windows-format metafile one record at a time.

The *lpHandleTable* and *nHandles* parameters must be identical to those passed to the [EnumMetaFileProc](#) callback procedure by [EnumMetaFile](#).

If the [PlayMetaFileRecord](#) function does not recognize a record, it ignores the record and returns **TRUE**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[EnumMetaFile](#)

[HANDLETABLE](#)

[METARECORD](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayMetaFile](#)

[SetWinMetaFileBits](#)

PlgBlt function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **PlgBlt** function performs a bit-block transfer of the bits of color data from the specified rectangle in the source device context to the specified parallelogram in the destination device context. If the given bitmask handle identifies a valid monochrome bitmap, the function uses this bitmap to mask the bits of color data from the source rectangle.

Syntax

```
BOOL PlgBlt(
    [in] HDC      hdcDest,
    [in] const POINT *lpPoint,
    [in] HDC      hdcSrc,
    [in] int       xSrc,
    [in] int       ySrc,
    [in] int       width,
    [in] int       height,
    [in] HBITMAP   hbmMask,
    [in] int       xMask,
    [in] int       yMask
);
```

Parameters

[in] hdcDest

A handle to the destination device context.

[in] lpPoint

A pointer to an array of three points in logical space that identify three corners of the destination parallelogram. The upper-left corner of the source rectangle is mapped to the first point in this array, the upper-right corner to the second point in this array, and the lower-left corner to the third point. The lower-right corner of the source rectangle is mapped to the implicit fourth point in the parallelogram.

[in] hdcSrc

A handle to the source device context.

[in] xSrc

The x-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] ySrc

The y-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] width

The width, in logical units, of the source rectangle.

[in] height

The height, in logical units, of the source rectangle.

[in] *hbmMask*

A handle to an optional monochrome bitmap that is used to mask the colors of the source rectangle.

[in] *xMask*

The x-coordinate, in logical units, of the upper-left corner of the monochrome bitmap.

[in] *yMask*

The y-coordinate, in logical units, of the upper-left corner of the monochrome bitmap.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **PiBgBlt** function works with device-dependent bitmaps.

The fourth vertex of the parallelogram (D) is defined by treating the first three points (A, B, and C) as vectors and computing $D = B + CA$.

If the bitmask exists, a value of one in the mask indicates that the source pixel color should be copied to the destination. A value of zero in the mask indicates that the destination pixel color is not to be changed. If the mask rectangle is smaller than the source and destination rectangles, the function replicates the mask pattern.

Scaling, translation, and reflection transformations are allowed in the source device context; however, rotation and shear transformations are not. If the mask bitmap is not a monochrome bitmap, an error occurs. The stretching mode for the destination device context is used to determine how to stretch or compress the pixels, if that is necessary.

When an enhanced metafile is being recorded, an error occurs if the source device context identifies an enhanced-metafile device context.

The destination coordinates are transformed according to the destination device context; the source coordinates are transformed according to the source device context. If the source transformation has a rotation or shear, an error is returned.

If the destination and source rectangles do not have the same color format, **PiBgBlt** converts the source rectangle to match the destination rectangle.

Not all devices support the **PiBgBlt** function. For more information, see the description of the RC_BITBLT raster capability in the [GetDeviceCaps](#) function.

If the source and destination device contexts represent incompatible devices, **PiBgBlt** returns an error.

When used in a multiple monitor system, both *hdcSrc* and *hdcDest* must refer to the same device or the function will fail. To transfer data between DCs for different devices, convert the memory bitmap to a DIB by calling [GetDIBits](#). To display the DIB to the second device, call [SetDIBits](#) or [StretchDIBits](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BitBlt](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[GetDIBits](#)

[GetDeviceCaps](#)

[MaskBlt](#)

[SetDIBits](#)

[SetStretchBltMode](#)

[StretchBlt](#)

[StretchDIBits](#)

POINTFX structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **POINTFX** structure contains the coordinates of points that describe the outline of a character in a TrueType font.

Syntax

```
typedef struct tagPOINTFX {  
    FIXED x;  
    FIXED y;  
} POINTFX, *LPOINTFX;
```

Members

x

The x-component of a point on the outline of a TrueType character.

y

The y-component of a point on the outline of a TrueType character.

Remarks

The **POINTFX** structure is a member of the [TTPOLYCURVE](#) and [TTPOLYGONHEADER](#) structures. Values in the **POINTFX** structure are specified in device units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[FIXED](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[TTPOLYCURVE](#)

[TTPOLYGONHEADER](#)

PolyBezier function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PolyBezier** function draws one or more Bézier curves.

Syntax

```
BOOL PolyBezier(
    [in] HDC         hdc,
    [in] const POINT *apt,
    [in] DWORD       cpt
);
```

Parameters

[in] *hdc*

A handle to a device context.

[in] *apt*

A pointer to an array of **POINT** structures that contain the endpoints and control points of the curve(s), in logical units.

[in] *cpt*

The number of points in the *lppt* array. This value must be one more than three times the number of curves to be drawn, because each Bézier curve requires two control points and an endpoint, and the initial curve requires an additional starting point.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **PolyBezier** function draws cubic Bézier curves by using the endpoints and control points specified by the *lppt* parameter. The first curve is drawn from the first point to the fourth point by using the second and third points as control points. Each subsequent curve in the sequence needs exactly three more points: the ending point of the previous curve is used as the starting point, the next two points in the sequence are control points, and the third is the ending point.

The current position is neither used nor updated by the **PolyBezier** function. The figure is not filled.

This function draws lines by using the current pen.

Examples

For an example, see [Redrawing in the Update Region](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

[MoveToEx](#)

[POINT](#)

[PolyBezierTo](#)

PolyBezierTo function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PolyBezierTo** function draws one or more Bézier curves.

Syntax

```
BOOL PolyBezierTo(
    [in] HDC      hdc,
    [in] const POINT *apt,
    [in] DWORD     cpt
);
```

Parameters

[in] **hdc**

A handle to a device context.

[in] **apt**

A pointer to an array of **POINT** structures that contains the endpoints and control points, in logical units.

[in] **cpt**

The number of points in the *lppt* array. This value must be three times the number of curves to be drawn because each Bézier curve requires two control points and an ending point.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

This function draws cubic Bézier curves by using the control points specified by the *lppt* parameter. The first curve is drawn from the current position to the third point by using the first two points as control points. For each subsequent curve, the function needs exactly three more points, and uses the ending point of the previous curve as the starting point for the next.

PolyBezierTo moves the current position to the ending point of the last Bézier curve. The figure is not filled.

This function draws lines by using the current pen.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

[MoveToEx](#)

[POINT](#)

[PolyBezier](#)

PolyDraw function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PolyDraw** function draws a set of line segments and Bézier curves.

Syntax

```
BOOL PolyDraw(
    [in] HDC         hdc,
    [in] const POINT *apt,
    [in] const BYTE  *aj,
    [in] int          cpt
);
```

Parameters

[in] **hdc**

A handle to a device context.

[in] **apt**

A pointer to an array of **POINT** structures that contains the endpoints for each line segment and the endpoints and control points for each Bézier curve, in logical units.

[in] **aj**

A pointer to an array that specifies how each point in the *lpt* array is used. This parameter can be one of the following values.

TYPE	MEANING
PT_MOVETO	Specifies that this point starts a disjoint figure. This point becomes the new current position.
PT_LINETO	Specifies that a line is to be drawn from the current position to this point, which then becomes the new current position.
PT_BEZIERTO	Specifies that this point is a control point or ending point for a Bézier curve. PT_BEZIERTO types always occur in sets of three. The current position defines the starting point for the Bézier curve. The first two PT_BEZIERTO points are the control points, and the third PT_BEZIERTO point is the ending point. The ending point becomes the new current position. If there are not three consecutive PT_BEZIERTO points, an error results.

A PT_LINETO or PT_BEZIERTO type can be combined with the following value by using the bitwise operator OR to indicate that the corresponding point is the last point in a figure and the figure is closed.

VALUE	MEANING
PT_CLOSEFIGURE	<p>Specifies that the figure is automatically closed after the PT_LINETO or PT_BEZIERTO type for this point is done. A line is drawn from this point to the most recent PT_MOVETO or MoveToEx point.</p> <p>This value is combined with the PT_LINETO type for a line, or with the PT_BEZIERTO type of the ending point for a Bézier curve, by using the bitwise operator OR.</p> <p>The current position is set to the ending point of the closing line.</p>

[in] cpt

The total number of points in the *lpt* array, the same as the number of bytes in the *lpbTypes* array.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The [PolyDraw](#) function can be used in place of consecutive calls to [MoveToEx](#), [LineTo](#), and [PolyBezierTo](#) functions to draw disjoint figures. The lines and curves are drawn using the current pen and figures are not filled. If there is an active path started by calling [BeginPath](#), [PolyDraw](#) adds to the path.

The points contained in the *lpt* array and in the *lpbTypes* array indicate whether each point is part of a [MoveTo](#), [LineTo](#), or [PolyBezierTo](#) operation. It is also possible to close figures.

This function updates the current position.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[EndPath](#)

[Line and Curve Functions](#)

[LineTo](#)

[Lines and Curves Overview](#)

[MoveToEx](#)

[POINT](#)

[PolyBezierTo](#)

[PolyLine](#)

Polygon function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **Polygon** function draws a polygon consisting of two or more vertices connected by straight lines. The polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode.

Syntax

```
BOOL Polygon(
    [in] HDC         hdc,
    [in] const POINT *apt,
    [in] int          cpt
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `apt`

A pointer to an array of `POINT` structures that specify the vertices of the polygon, in logical coordinates.

[in] `cpt`

The number of vertices in the array. This value must be greater than or equal to 2.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The polygon is closed automatically by drawing a line from the last vertex to the first.

The current position is neither used nor updated by the **Polygon** function.

Any extra points are ignored. To draw a line with more points, divide your data into groups, each of which have less than the maximum number of points, and call the function for each group of points. Remember to connect the line segments.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

[GetPolyFillMode](#)

[POINT](#)

[PolyPolygon](#)

[Polyline](#)

[PolylineTo](#)

[SetPolyFillMode](#)

Polyline function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **Polyline** function draws a series of line segments by connecting the points in the specified array.

Syntax

```
BOOL Polyline(
    [in] HDC         hdc,
    [in] const POINT *apt,
    [in] int          cpt
);
```

Parameters

[in] `hdc`

A handle to a device context.

[in] `apt`

A pointer to an array of [POINT](#) structures, in logical units.

[in] `cpt`

The number of points in the array. This number must be greater than or equal to two.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The lines are drawn from the first point through subsequent points by using the current pen. Unlike the [LineTo](#) or [PolylineTo](#) functions, the **Polyline** function neither uses nor updates the current position.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[Line and Curve Functions](#)

[LineTo](#)

[Lines and Curves Overview](#)

[MoveToEx](#)

[POINT](#)

[PolyPolyline](#)

[PolylineTo](#)

PolylineTo function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PolylineTo** function draws one or more straight lines.

Syntax

```
BOOL PolylineTo(
    [in] HDC         hdc,
    [in] const POINT *apt,
    [in] DWORD       cpt
);
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **apt**

A pointer to an array of **POINT** structures that contains the vertices of the line, in logical units.

[in] **cpt**

The number of points in the array.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Unlike the **Polyline** function, the **PolylineTo** function uses and updates the current position.

A line is drawn from the current position to the first point specified by the *lppt* parameter by using the current pen. For each additional line, the function draws from the ending point of the previous line to the next point specified by *lppt*.

PolylineTo moves the current position to the ending point of the last line.

If the line segments drawn by this function form a closed figure, the figure is not filled.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Line and Curve Functions](#)

[LineTo](#)

[Lines and Curves Overview](#)

[MoveToEx](#)

[POINT](#)

[PolyPolyline](#)

[Polyline](#)

PolyPolygon function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PolyPolygon** function draws a series of closed polygons. Each polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode. The polygons drawn by this function can overlap.

Syntax

```
BOOL PolyPolygon(
    [in] HDC         hdc,
    [in] const POINT *apt,
    [in] const INT   *asz,
    [in] int         csz
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `apt`

A pointer to an array of **POINT** structures that define the vertices of the polygons, in logical coordinates. The polygons are specified consecutively. Each polygon is closed automatically by drawing a line from the last vertex to the first. Each vertex should be specified once.

[in] `asz`

A pointer to an array of integers, each of which specifies the number of points in the corresponding polygon. Each integer must be greater than or equal to 2.

[in] `csz`

The total number of polygons.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The current position is neither used nor updated by this function.

Any extra points are ignored. To draw the polygons with more points, divide your data into groups, each of which have less than the maximum number of points, and call the function for each group of points. Note, it is best to have a polygon in only one of the groups.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

[GetPolyFillMode](#)

[POINT](#)

[Polygon](#)

[Polyline](#)

[PolylineTo](#)

[SetPolyFillMode](#)

PolyPolyline function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PolyPolyline** function draws multiple series of connected line segments.

Syntax

```
BOOL PolyPolyline(
    [in] HDC         hdc,
    [in] const POINT *apt,
    [in] const DWORD *asz,
    [in] DWORD       csz
);
```

Parameters

[in] *hdc*

A handle to the device context.

[in] *apt*

A pointer to an array of **POINT** structures that contains the vertices of the polylines, in logical units. The polylines are specified consecutively.

[in] *asz*

A pointer to an array of variables specifying the number of points in the *lptt* array for the corresponding polyline. Each entry must be greater than or equal to two.

[in] *csz*

The total number of entries in the *lpdwPolyPoints* array.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The line segments are drawn by using the current pen. The figures formed by the segments are not filled.

The current position is neither used nor updated by this function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

[POINT](#)

[Polyline](#)

[PolylineTo](#)

POLYTEXTA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The POLYTEXT structure describes how the [PolyTextOut](#) function should draw a string of text.

Syntax

```
typedef struct tagPOLYTEXTA {
    int      x;
    int      y;
    UINT     n;
    LPCSTR   lpstr;
    UINT     uiFlags;
    RECT    rcl;
    int     *pdx;
} POLYTEXTA, *PPOLYTEXTA, *NPPOLYTEXTA, *LPPOLYTEXTA;
```

Members

x

The horizontal reference point for the string. The string is aligned to this point using the current text-alignment mode.

y

The vertical reference point for the string. The string is aligned to this point using the current text-alignment mode.

n

The [length of the string](#) pointed to by `lpstr`.

lpstr

Pointer to a string of text to be drawn by the [PolyTextOut](#) function. This string need not be null-terminated, since `n` specifies the length of the string.

uiFlags

Specifies whether the string is to be opaque or clipped and whether the string is accompanied by an array of character-width values. This member can be one or more of the following values.

VALUE	MEANING
ETO_OPAQUE	The rectangle for each string is to be opaqued with the current background color.
ETO_CLIPPED	Each string is to be clipped to its specified rectangle.

rcl

A rectangle structure that contains the dimensions of the opaquing or clipping rectangle. This member is ignored if neither of the ETO_OPAQUE nor the ETO_CLIPPED value is specified for the `uiFlags` member.

Pointer to an array containing the width value for each character in the string.

Remarks

NOTE

The wingdi.h header defines POLYTEXT as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[PolyTextOut](#)

PolyTextOutA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PolyTextOut** function draws several strings using the font and text colors currently selected in the specified device context.

Syntax

```
BOOL PolyTextOutA(
    [in] HDC           hdc,
    [in] const POLYTEXTA *ppt,
    [in] int            nstrings
);
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **ppt**

A pointer to an array of **POLYTEXT** structures describing the strings to be drawn. The array contains one structure for each string to be drawn.

[in] **nstrings**

The number of **POLYTEXT** structures in the *ppetxt* array.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Each **POLYTEXT** structure contains the coordinates of a reference point that Windows uses to align the corresponding string of text. An application can specify how the reference point is used by calling the **SetTextAlign** function. An application can determine the current text-alignment setting for the specified device context by calling the **GetTextAlign** function.

To draw a single string of text, the application should call the **ExtTextOut** function.

PolyTextOut will not handle international scripting support automatically. To get international scripting support, use **ExtTextOut** instead. **ExtTextOut** will use **Uniscribe** when necessary resulting in font fallback. Additionally, **ExtTextOut** will perform internal batching of calls before transitioning to kernel mode, mitigating some of the performance concerns when weighing usage of **PolyTextOut** versus **ExtTextOut**.

TIP

ExtTextOut is strongly recommended over **PolyTextOut** for modern development due to its ability to handle display of different languages.

NOTE

The wingdi.h header defines PolyTextOut as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[POLYTEXT](#)

[SetTextAlign](#)

PolyTextOutW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PolyTextOut** function draws several strings using the font and text colors currently selected in the specified device context.

Syntax

```
BOOL PolyTextOutW(
    [in] HDC           hdc,
    [in] const POLYTEXTW *ppt,
    [in] int            nstrings
);
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **ppt**

A pointer to an array of **POLYTEXT** structures describing the strings to be drawn. The array contains one structure for each string to be drawn.

[in] **nstrings**

The number of **POLYTEXT** structures in the *ppetxt* array.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Each **POLYTEXT** structure contains the coordinates of a reference point that Windows uses to align the corresponding string of text. An application can specify how the reference point is used by calling the **SetTextAlign** function. An application can determine the current text-alignment setting for the specified device context by calling the **GetTextAlign** function.

To draw a single string of text, the application should call the **ExtTextOut** function.

PolyTextOut will not handle international scripting support automatically. To get international scripting support, use **ExtTextOut** instead. **ExtTextOut** will use **Uniscribe** when necessary resulting in font fallback. Additionally, **ExtTextOut** will perform internal batching of calls before transitioning to kernel mode, mitigating some of the performance concerns when weighing usage of **PolyTextOut** versus **ExtTextOut**.

TIP

ExtTextOut is strongly recommended over **PolyTextOut** for modern development due to its ability to handle display of different languages.

NOTE

The wingdi.h header defines PolyTextOut as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[POLYTEXT](#)

[SetTextAlign](#)

POLYTEXTW structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The POLYTEXT structure describes how the [PolyTextOut](#) function should draw a string of text.

Syntax

```
typedef struct tagPOLYTEXTW {  
    int      x;  
    int      y;  
    UINT     n;  
    LPCWSTR  lpstr;  
    UINT     uiFlags;  
    RECT    rcl;  
    int      *pdx;  
} POLYTEXTW, *PPOLYTEXTW, *NPPOLYTEXTW, *LPPOLYTEXTW;
```

Members

x

The horizontal reference point for the string. The string is aligned to this point using the current text-alignment mode.

y

The vertical reference point for the string. The string is aligned to this point using the current text-alignment mode.

n

The [length of the string](#) pointed to by `lpstr`.

lpstr

Pointer to a string of text to be drawn by the [PolyTextOut](#) function. This string need not be null-terminated, since `n` specifies the length of the string.

uiFlags

Specifies whether the string is to be opaque or clipped and whether the string is accompanied by an array of character-width values. This member can be one or more of the following values.

VALUE	MEANING
ETO_OPAQUE	The rectangle for each string is to be opaqued with the current background color.
ETO_CLIPPED	Each string is to be clipped to its specified rectangle.

rcl

A rectangle structure that contains the dimensions of the opaquing or clipping rectangle. This member is ignored if neither of the ETO_OPAQUE nor the ETO_CLIPPED value is specified for the `uiFlags` member.

Pointer to an array containing the width value for each character in the string.

Remarks

NOTE

The wingdi.h header defines POLYTEXT as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[PolyTextOut](#)

PtInRegion function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PtInRegion** function determines whether the specified point is inside the specified region.

Syntax

```
BOOL PtInRegion(
    [in] Hrgn hrgn,
    [in] int x,
    [in] int y
);
```

Parameters

[in] **hrgn**

Handle to the region to be examined.

[in] **x**

Specifies the x-coordinate of the point in logical units.

[in] **y**

Specifies the y-coordinate of the point in logical units.

Return value

If the specified point is in the region, the return value is nonzero.

If the specified point is not in the region, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[RectInRegion](#)

[Region Functions](#)

[Regions Overview](#)

PtVisible function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PtVisible** function determines whether the specified point is within the clipping region of a device context.

Syntax

```
BOOL PtVisible(
    [in] HDC hdc,
    [in] int x,
    [in] int y
);
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **x**

The x-coordinate, in logical units, of the point.

[in] **y**

The y-coordinate, in logical units, of the point.

Return value

If the specified point is within the clipping region of the device context, the return value is **TRUE(1)**.

If the specified point is not within the clipping region of the device context, the return value is **FALSE(0)**.

If the **HDC** is not valid, the return value is **(BOOL)-1**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Clipping Functions](#)

[Clipping Overview](#)

[RectVisible](#)

RASTERIZER_STATUS structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RASTERIZER_STATUS** structure contains information about whether TrueType is installed. This structure is filled when an application calls the [GetRasterizerCaps](#) function.

Syntax

```
typedef struct _RASTERIZER_STATUS {
    short nSize;
    short wFlags;
    short nLanguageID;
} RASTERIZER_STATUS, *LPRASTERIZER_STATUS;
```

Members

nSize

The size, in bytes, of the **RASTERIZER_STATUS** structure.

wFlags

Specifies whether at least one TrueType font is installed and whether TrueType is enabled. This value is TT_AVAILABLE, TT_ENABLED, or both if TrueType is on the system.

nLanguageID

The language in the system's Setup.inf file.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetRasterizerCaps](#)

RealizePalette function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RealizePalette** function maps palette entries from the current logical palette to the system palette.

Syntax

```
UINT RealizePalette(  
    [in] HDC hdc  
>);
```

Parameters

[in] hdc

A handle to the device context into which a logical palette has been selected.

Return value

If the function succeeds, the return value is the number of entries in the logical palette mapped to the system palette.

If the function fails, the return value is GDI_ERROR.

Remarks

An application can determine whether a device supports palette operations by calling the [GetDeviceCaps](#) function and specifying the RASTERCAPS constant.

The **RealizePalette** function modifies the palette for the device associated with the specified device context. If the device context is a memory DC, the color table for the bitmap selected into the DC is modified. If the device context is a display DC, the physical palette for that device is modified.

A logical palette is a buffer between color-intensive applications and the system, allowing these applications to use as many colors as needed without interfering with colors displayed by other windows.

When an application's window has the focus and it calls the **RealizePalette** function, the system attempts to realize as many of the requested colors as possible. The same is also true for applications with inactive windows.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[CreatePalette](#)

[GetDeviceCaps](#)

[SelectPalette](#)

Rectangle function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **Rectangle** function draws a rectangle. The rectangle is outlined by using the current pen and filled by using the current brush.

Syntax

```
BOOL Rectangle(
    [in] HDC hdc,
    [in] int left,
    [in] int top,
    [in] int right,
    [in] int bottom
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `left`

The x-coordinate, in logical coordinates, of the upper-left corner of the rectangle.

[in] `top`

The y-coordinate, in logical coordinates, of the upper-left corner of the rectangle.

[in] `right`

The x-coordinate, in logical coordinates, of the lower-right corner of the rectangle.

[in] `bottom`

The y-coordinate, in logical coordinates, of the lower-right corner of the rectangle.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The current position is neither used nor updated by **Rectangle**.

The rectangle that is drawn excludes the bottom and right edges.

If a PS_NULL pen is used, the dimensions of the rectangle are 1 pixel less in height and 1 pixel less in width.

Examples

For an example, see [Using Filled Shapes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

[RoundRect](#)

RectInRegion function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RectInRegion** function determines whether any part of the specified rectangle is within the boundaries of a region.

Syntax

```
BOOL RectInRegion(
    [in] HRGN      hrgn,
    [in] const RECT *lprect
);
```

Parameters

[in] `hrgn`

Handle to the region.

[in] `lprect`

Pointer to a [RECT](#) structure containing the coordinates of the rectangle in logical units. The lower and right edges of the rectangle are not included.

Return value

If any part of the specified rectangle lies within the boundaries of the region, the return value is nonzero.

If no part of the specified rectangle lies within the boundaries of the region, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[PtInRegion](#)

RECT

[Region Functions](#)

[Regions Overview](#)

RectVisible function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RectVisible** function determines whether any part of the specified rectangle lies within the clipping region of a device context.

Syntax

```
BOOL RectVisible(
    [in] HDC      hdc,
    [in] const RECT *lprect
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lprect`

A pointer to a [RECT](#) structure that contains the logical coordinates of the specified rectangle.

Return value

If the current transform does not have a rotation and the rectangle lies within the clipping region, the return value is **TRUE** (1).

If the current transform does not have a rotation and the rectangle does not lie within the clipping region, the return value is **FALSE** (0).

If the current transform has a rotation and the rectangle lies within the clipping region, the return value is 2.

If the current transform has a rotation and the rectangle does not lie within the clipping region, the return value is 1.

All other return values are considered error codes. If the any parameter is not valid, the return value is undefined.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Clipping Functions](#)

[Clipping Overview](#)

[CreateRectRgn](#)

[PtVisible](#)

[RECT](#)

[SelectClipRgn](#)

RemoveFontMemResourceEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RemoveFontMemResourceEx** function removes the fonts added from a memory image file.

Syntax

```
BOOL RemoveFontMemResourceEx(
    [in] HANDLE h
);
```

Parameters

[in] *h*

A handle to the font-resource. This handle is returned by the [AddFontMemResourceEx](#) function.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. No extended error information is available.

Remarks

This function removes a font that was added by the [AddFontMemResourceEx](#) function. To remove the font, specify the same path and flags as were used in [AddFontMemResourceEx](#). This function will only remove the font that is specified by *fh*.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AddFontMemResourceEx](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[SendMessage](#)

RemoveFontResourceA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RemoveFontResource** function removes the fonts in the specified file from the system font table.

If the font was added using the [AddFontResourceEx](#) function, you must use the [RemoveFontResourceEx](#) function.

Syntax

```
BOOL RemoveFontResourceA(
    [in] LPCSTR lpFileName
);
```

Parameters

[in] *lpFileName*

A pointer to a null-terminated string that names a font resource file.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

We recommend that if an app adds or removes fonts from the system font table that it notify other windows of the change by sending a [WM_FONTCCHANGE](#) message to all top-level windows in the system. The app sends this message by calling the [SendMessage](#) function with the *hwnd* parameter set to [HWND_BROADCAST](#).

If there are outstanding references to a font, the associated resource remains loaded until no device context is using it. Furthermore, if the font is listed in the font registry

(**HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts**) and is installed to any location other than the %windir%\fonts\ folder, it may be loaded into other active sessions (including session 0).

When you try to replace an existing font file that contains a font with outstanding references to it, you might get an error that indicates that the original font can't be deleted because it's in use even after you call **RemoveFontResource**. If your app requires that the font file be replaced, to reduce the resource count of the original font to zero, call **RemoveFontResource** in a loop as shown in this example code. If you continue to get errors, this is an indication that the font file remains loaded in other sessions. Make sure the font isn't listed in the font registry and restart the system to ensure the font is unloaded from all sessions.

Note Apps where the original font file is in use will still be able to access the original file and won't use the new font until the font reloads. Call [AddFontResource](#) to reload the font. We recommend that you call [AddFontResource](#) the same number of times as the call to **RemoveFontResource** succeeded as shown in this example code.

```

int i = 0;
while( RemoveFontResource( FontFile ) )
{
    i++;
}

// TODO: Replace font file

while( i-- )
{
    AddFontResource( FontFile );
}

```

NOTE

The wingdi.h header defines RemoveFontResource as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AddFontResource](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[RemoveFontResourceEx](#)

[SendMessage](#)

RemoveFontResourceExA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RemoveFontResourceEx** function removes the fonts in the specified file from the system font table.

Syntax

```
BOOL RemoveFontResourceExA(
    [in] LPCSTR name,
    [in] DWORD  fl,
    [in] PVOID  pdv
);
```

Parameters

[in] **name**

A pointer to a null-terminated string that names a font resource file.

[in] **fl**

The characteristics of the font to be removed from the system. In order for the font to be removed, the flags used must be the same as when the font was added with the [AddFontResourceEx](#) function. See the [AddFontResourceEx](#) function for more information.

[in] **pdv**

Reserved. Must be zero.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. No extended error information is available.

Remarks

This function will only remove the font if the flags specified are the same as when the font was added with the [AddFontResourceEx](#) function.

When you try to replace an existing font file that contains a font with outstanding references to it, you might get an error that indicates that the original font can't be deleted because it's in use even after you call **RemoveFontResourceEx**. If your app requires that the font file be replaced, to reduce the resource count of the original font to zero, call **RemoveFontResourceEx** in a loop as shown in this example code. If you continue to get errors, this is an indication that the font file remains loaded in other sessions. Make sure the font isn't listed in the font registry and restart the system to ensure the font is unloaded from all sessions.

Note Apps where the original font file is in use will still be able to access the original file and won't use the new font until the font reloads. Call [AddFontResourceEx](#) to reload the font. We recommend that you call [AddFontResourceEx](#) the same number of times as the call to **RemoveFontResourceEx** succeeded as shown in this example code.

```

int i = 0;
while( RemoveFontResourceEx( FontFile, FR_PRIVATE, 0 ) )
{
    i++;
}

// TODO: Replace font file

while( i-- )
{
    AddFontResourceEx( FontFile, FR_PRIVATE, 0 );
}

```

NOTE

The wingdi.h header defines RemoveFontResourceEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AddFontResourceEx](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[SendMessage](#)

RemoveFontResourceExW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RemoveFontResourceEx** function removes the fonts in the specified file from the system font table.

Syntax

```
BOOL RemoveFontResourceExW(
    [in] LPCWSTR name,
    [in] DWORD   f1,
    [in] PVOID   pdv
);
```

Parameters

[in] **name**

A pointer to a null-terminated string that names a font resource file.

[in] **f1**

The characteristics of the font to be removed from the system. In order for the font to be removed, the flags used must be the same as when the font was added with the [AddFontResourceEx](#) function. See the [AddFontResourceEx](#) function for more information.

[in] **pdv**

Reserved. Must be zero.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. No extended error information is available.

Remarks

This function will only remove the font if the flags specified are the same as when the font was added with the [AddFontResourceEx](#) function.

When you try to replace an existing font file that contains a font with outstanding references to it, you might get an error that indicates that the original font can't be deleted because it's in use even after you call **RemoveFontResourceEx**. If your app requires that the font file be replaced, to reduce the resource count of the original font to zero, call **RemoveFontResourceEx** in a loop as shown in this example code. If you continue to get errors, this is an indication that the font file remains loaded in other sessions. Make sure the font isn't listed in the font registry and restart the system to ensure the font is unloaded from all sessions.

Note Apps where the original font file is in use will still be able to access the original file and won't use the new font until the font reloads. Call [AddFontResourceEx](#) to reload the font. We recommend that you call [AddFontResourceEx](#) the same number of times as the call to **RemoveFontResourceEx** succeeded as shown in this example code.

```

int i = 0;
while( RemoveFontResourceEx( FontFile, FR_PRIVATE, 0 ) )
{
    i++;
}

// TODO: Replace font file

while( i-- )
{
    AddFontResourceEx( FontFile, FR_PRIVATE, 0 );
}

```

NOTE

The wingdi.h header defines RemoveFontResourceEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AddFontResourceEx](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[SendMessage](#)

RemoveFontResourceW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RemoveFontResource** function removes the fonts in the specified file from the system font table.

If the font was added using the [AddFontResourceEx](#) function, you must use the [RemoveFontResourceEx](#) function.

Syntax

```
BOOL RemoveFontResource(
    [in] LPCWSTR lpFileName
);
```

Parameters

[in] *lpFileName*

A pointer to a null-terminated string that names a font resource file.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

We recommend that if an app adds or removes fonts from the system font table that it notify other windows of the change by sending a [WM_FONTCCHANGE](#) message to all top-level windows in the system. The app sends this message by calling the [SendMessage](#) function with the *hwnd* parameter set to [HWND_BROADCAST](#).

If there are outstanding references to a font, the associated resource remains loaded until no device context is using it. Furthermore, if the font is listed in the font registry

([HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts](#)) and is installed to any location other than the %windir%\fonts\ folder, it may be loaded into other active sessions (including session 0).

When you try to replace an existing font file that contains a font with outstanding references to it, you might get an error that indicates that the original font can't be deleted because it's in use even after you call **RemoveFontResource**. If your app requires that the font file be replaced, to reduce the resource count of the original font to zero, call **RemoveFontResource** in a loop as shown in this example code. If you continue to get errors, this is an indication that the font file remains loaded in other sessions. Make sure the font isn't listed in the font registry and restart the system to ensure the font is unloaded from all sessions.

Note Apps where the original font file is in use will still be able to access the original file and won't use the new font until the font reloads. Call [AddFontResource](#) to reload the font. We recommend that you call [AddFontResource](#) the same number of times as the call to **RemoveFontResource** succeeded as shown in this example code.

```

int i = 0;
while( RemoveFontResource( FontFile ) )
{
    i++;
}

// TODO: Replace font file

while( i-- )
{
    AddFontResource( FontFile );
}

```

NOTE

The wingdi.h header defines RemoveFontResource as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[AddFontResource](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[RemoveFontResourceEx](#)

[SendMessage](#)

ResetDCA function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ResetDC** function updates the specified printer or plotter device context (DC) using the specified information.

Syntax

```
HDC ResetDCA(
    [in] HDC           hdc,
    [in] const DEVMODEA *lpdm
);
```

Parameters

[in] `hdc`

A handle to the DC to update.

[in] `lpdm`

A pointer to a [DEVMODE](#) structure containing information about the new DC.

Return value

If the function succeeds, the return value is a handle to the original DC.

If the function fails, the return value is **NULL**.

Remarks

An application will typically use the **ResetDC** function when a window receives a [WM_DEVMODECHANGE](#) message. **ResetDC** can also be used to change the paper orientation or paper bins while printing a document.

The **ResetDC** function cannot be used to change the driver name, device name, or the output port. When the user changes the port connection or device name, the application must delete the original DC and create a new DC with the new information.

An application can pass an information DC to the **ResetDC** function. In that situation, **ResetDC** will always return a printer DC.

ICM: The color profile of the DC specified by the `hdc` parameter will be reset based on the information contained in the `lpInitData` member of the [DEVMODE](#) structure.

NOTE

The wingdi.h header defines **ResetDC** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DEVMODE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[DeviceCapabilities](#)

[Escape](#)

ResetDCW function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ResetDC** function updates the specified printer or plotter device context (DC) using the specified information.

Syntax

```
HDC ResetDCW(
    [in] HDC           hdc,
    [in] const DEVMODEW *lpdm
);
```

Parameters

[in] `hdc`

A handle to the DC to update.

[in] `lpdm`

A pointer to a [DEVMODE](#) structure containing information about the new DC.

Return value

If the function succeeds, the return value is a handle to the original DC.

If the function fails, the return value is **NULL**.

Remarks

An application will typically use the **ResetDC** function when a window receives a [WM_DEVMODECHANGE](#) message. **ResetDC** can also be used to change the paper orientation or paper bins while printing a document.

The **ResetDC** function cannot be used to change the driver name, device name, or the output port. When the user changes the port connection or device name, the application must delete the original DC and create a new DC with the new information.

An application can pass an information DC to the **ResetDC** function. In that situation, **ResetDC** will always return a printer DC.

ICM: The color profile of the DC specified by the `hdc` parameter will be reset based on the information contained in the `lpInitData` member of the [DEVMODE](#) structure.

NOTE

The wingdi.h header defines **ResetDC** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DEVMODE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[DeviceCapabilities](#)

[Escape](#)

ResizePalette function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ResizePalette** function increases or decreases the size of a logical palette based on the specified value.

Syntax

```
BOOL ResizePalette(
    [in] HPALETTE hpal,
    [in] UINT      n
);
```

Parameters

[in] `hpal`

A handle to the palette to be changed.

[in] `n`

The number of entries in the palette after it has been resized.

The number of entries is limited to 1024.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

An application can determine whether a device supports palette operations by calling the [GetDeviceCaps](#) function and specifying the RASTERCAPS constant.

If an application calls **ResizePalette** to reduce the size of the palette, the entries remaining in the resized palette are unchanged. If the application calls **ResizePalette** to enlarge the palette, the additional palette entries are set to black (the red, green, and blue values are all 0) and their flags are set to zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[GetDeviceCaps](#)

RestoreDC function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RestoreDC** function restores a device context (DC) to the specified state. The DC is restored by popping state information off a stack created by earlier calls to the [SaveDC](#) function.

Syntax

```
BOOL RestoreDC(
    [in] HDC hdc,
    [in] int nSavedDC
);
```

Parameters

[in] `hdc`

A handle to the DC.

[in] `nSavedDC`

The saved state to be restored. If this parameter is positive, `nSavedDC` represents a specific instance of the state to be restored. If this parameter is negative, `nSavedDC` represents an instance relative to the current state. For example, -1 restores the most recently saved state.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Each DC maintains a stack of saved states. The [SaveDC](#) function pushes the current state of the DC onto its stack of saved states. That state can be restored only to the same DC from which it was created. After a state is restored, the saved state is destroyed and cannot be reused. Furthermore, any states saved after the restored state was created are also destroyed and cannot be used. In other words, the **RestoreDC** function pops the restored state (and any subsequent states) from the state information stack.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Device Context Functions](#)

[Device Contexts Overview](#)

[SaveDC](#)

RGB macro (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RGB** macro selects a red, green, blue (RGB) color based on the arguments supplied and the color capabilities of the output device.

Syntax

```
void RGB(  
    r,  
    g,  
    b  
) ;
```

Parameters

r

The intensity of the red color.

g

The intensity of the green color.

b

The intensity of the blue color.

Return value

None

Remarks

The intensity for each argument is in the range 0 through 255. If all three intensities are zero, the result is black. If all three intensities are 255, the result is white.

To extract the individual values for the red, green, and blue components of a **COLORREF** color value, use the [GetRValue](#), [GetGValue](#), and [GetBValue](#) macros, respectively.

When creating or examining a logical palette, use the **RGBQUAD** structure to define color values and examine individual component values. For more information about using color values in a color palette, see the descriptions of the [PALETTEINDEX](#) and [PALETTERGB](#) macros.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[COLORREF](#)

[Color Macros](#)

[Colors Overview](#)

[GetBValue](#)

[GetGValue](#)

[GetRValue](#)

[PALETTEINDEX](#)

[PALETTERGB](#)

[RGBQUAD](#)

RGBQUAD structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The RGBQUAD structure describes a color consisting of relative intensities of red, green, and blue.

Syntax

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

Members

`rgbBlue`

The intensity of blue in the color.

`rgbGreen`

The intensity of green in the color.

`rgbRed`

The intensity of red in the color.

`rgbReserved`

This member is reserved and must be zero.

Remarks

The `bmiColors` member of the [BITMAPINFO](#) structure consists of an array of RGBQUAD structures.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPINFO](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

[CreateDIBSection](#)

[CreateDIBitmap](#)

[GetDIBits](#)

[SetDIBits](#)

[SetDIBitsToDevice](#)

[StretchDIBits](#)

RGBTRIPLE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RGBTRIPLE** structure describes a color consisting of relative intensities of red, green, and blue. The **bmciColors** member of the [BITMAPCOREINFO](#) structure consists of an array of **RGBTRIPLE** structures.

Syntax

```
typedef struct tagRGBTRIPLE {  
    BYTE rgbtBlue;  
    BYTE rgbtGreen;  
    BYTE rgbtRed;  
} RGBTRIPLE, *PRGBTRIPLE, *NPRGBTRIPLE, *LPRGBTRIPLE;
```

Members

rgbtBlue

The intensity of blue in the color.

rgbtGreen

The intensity of green in the color.

rgbtRed

The intensity of red in the color.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[BITMAPCOREINFO](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

RGNDATA structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RGNDATA** structure contains a header and an array of rectangles that compose a region. The rectangles are sorted top to bottom, left to right. They do not overlap.

Syntax

```
typedef struct _RGNDATA {  
    RGNDATAHEADER rdh;  
    char         Buffer[1];  
} RGNDATA, *PRGNDATA, *NPRGNDATA, *LPRGNDATA;
```

Members

rdh

A [RGNDATAHEADER](#) structure. The members of this structure specify the type of region (whether it is rectangular or trapezoidal), the number of rectangles that make up the region, the size of the buffer that contains the rectangle structures, and so on.

Buffer

Specifies an arbitrary-size buffer that contains the [RECT](#) structures that make up the region.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[RECT](#)

[RGNDATAHEADER](#)

[Region Structures](#)

[Regions Overview](#)

RGNDATAHEADER structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RGNDATAHEADER** structure describes the data returned by the [GetRegionData](#) function.

Syntax

```
typedef struct _RGNDATAHEADER {
    DWORD dwSize;
    DWORD iType;
    DWORD nCount;
    DWORD nRgnSize;
    RECT  rcBound;
} RGNDATAHEADER, *PRGNDATAHEADER;
```

Members

dwSize

The size, in bytes, of the header.

iType

The type of region. This value must be RDH_RECTANGLES.

nCount

The number of rectangles that make up the region.

nRgnSize

The size of the [RGNDATA](#) buffer required to receive the [RECT](#) structures that make up the region. If the size is not known, this member can be zero.

rcBound

A bounding rectangle for the region in logical units.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[GetRegionData](#)

[RECT](#)

RGNDATA

Region Structures

Regions Overview

RoundRect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **RoundRect** function draws a rectangle with rounded corners. The rectangle is outlined by using the current pen and filled by using the current brush.

Syntax

```
BOOL RoundRect(
    [in] HDC hdc,
    [in] int left,
    [in] int top,
    [in] int right,
    [in] int bottom,
    [in] int width,
    [in] int height
);
```

Parameters

[in] hdc

A handle to the device context.

[in] left

The x-coordinate, in logical coordinates, of the upper-left corner of the rectangle.

[in] top

The y-coordinate, in logical coordinates, of the upper-left corner of the rectangle.

[in] right

The x-coordinate, in logical coordinates, of the lower-right corner of the rectangle.

[in] bottom

The y-coordinate, in logical coordinates, of the lower-right corner of the rectangle.

[in] width

The width, in logical coordinates, of the ellipse used to draw the rounded corners.

[in] height

The height, in logical coordinates, of the ellipse used to draw the rounded corners.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The current position is neither used nor updated by this function.

Examples

For an example, see [Using Filled Shapes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

[Rectangle](#)

SaveDC function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SaveDC** function saves the current state of the specified device context (DC) by copying data describing selected objects and graphic modes (such as the bitmap, brush, palette, font, pen, region, drawing mode, and mapping mode) to a context stack.

Syntax

```
int SaveDC(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

A handle to the DC whose state is to be saved.

Return value

If the function succeeds, the return value identifies the saved state.

If the function fails, the return value is zero.

Remarks

The **SaveDC** function can be used any number of times to save any number of instances of the DC state.

A saved state can be restored by using the [RestoreDC](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Device Context Functions](#)

[Device Contexts Overview](#)

[RestoreDC](#)

ScaleViewportExtEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ScaleViewportExtEx** function modifies the viewport for a device context using the ratios formed by the specified multiplicands and divisors.

Syntax

```
BOOL ScaleViewportExtEx(
    [in]  HDC      hdc,
    [in]  int      xn,
    [in]  int      dx,
    [in]  int      yn,
    [in]  int      yd,
    [out] LPSIZE  lpsz
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `xn`

The amount by which to multiply the current horizontal extent.

[in] `dx`

The amount by which to divide the current horizontal extent.

[in] `yn`

The amount by which to multiply the current vertical extent.

[in] `yd`

The amount by which to divide the current vertical extent.

[out] `lpsz`

A pointer to a `SIZE` structure that receives the previous viewport extents, in device units. If `/pSize` is `NULL`, this parameter is not used.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The viewport extents are modified as follows:

```
xNewVE = (xOldVE * Xnum) / Xdenom  
yNewVE = (yOldVE * Ynum) / Ydenom
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetViewportExtEx](#)

[SIZE](#)

ScaleWindowExtEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ScaleWindowExtEx** function modifies the window for a device context using the ratios formed by the specified multiplicands and divisors.

Syntax

```
BOOL ScaleWindowExtEx(
    [in]  HDC      hdc,
    [in]  int      xn,
    [in]  int      xd,
    [in]  int      yn,
    [in]  int      yd,
    [out] LPSIZE  lpsz
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `xn`

The amount by which to multiply the current horizontal extent.

[in] `xd`

The amount by which to divide the current horizontal extent.

[in] `yn`

The amount by which to multiply the current vertical extent.

[in] `yd`

The amount by which to divide the current vertical extent.

[out] `lpsz`

A pointer to a `SIZE` structure that receives the previous window extents, in logical units. If `/pSize` is `NULL`, this parameter is not used.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The window extents are modified as follows:

```
xNewWE = (xOldWE * Xnum) / Xdenom  
yNewWE = (yOldWE * Ynum) / Ydenom
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetWindowExtEx](#)

[SIZE](#)

SelectClipPath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SelectClipPath** function selects the current path as a clipping region for a device context, combining the new region with any existing clipping region using the specified mode.

Syntax

```
BOOL SelectClipPath(  
    [in] HDC hdc,  
    [in] int mode  
)
```

Parameters

[in] *hdc*

A handle to the device context of the path.

[in] *mode*

The way to use the path. This parameter can be one of the following values.

VALUE	MEANING
RGN_AND	The new clipping region includes the intersection (overlapping areas) of the current clipping region and the current path.
RGN_COPY	The new clipping region is the current path.
RGN_DIFF	The new clipping region includes the areas of the current clipping region with those of the current path excluded.
RGN_OR	The new clipping region includes the union (combined areas) of the current clipping region and the current path.
RGN_XOR	The new clipping region includes the union of the current clipping region and the current path but without the overlapping areas.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The device context identified by the *hdc* parameter must contain a closed path.

Examples

For an example, see [Using Clipping](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[Clipping Functions](#)

[Clipping Overview](#)

[EndPath](#)

SelectClipRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SelectClipRgn** function selects a region as the current clipping region for the specified device context.

Syntax

```
int SelectClipRgn(  
    [in] HDC hdc,  
    [in] HRGN hrgn  
)
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **hrgn**

A handle to the region to be selected.

Return value

The return value specifies the region's complexity and can be one of the following values.

RETURN CODE	DESCRIPTION
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred. (The previous clipping region is unaffected.)

Remarks

Only a copy of the selected region is used. The region itself can be selected for any number of other device contexts or it can be deleted.

The **SelectClipRgn** function assumes that the coordinates for a region are specified in device units.

To remove a device-context's clipping region, specify a **NULL** region handle.

Examples

For an example, see [Clipping Output](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Clipping Functions](#)

[Clipping Overview](#)

[ExtSelectClipRgn](#)

SelectObject function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SelectObject** function selects an object into the specified device context (DC). The new object replaces the previous object of the same type.

Syntax

```
HGDIOBJ SelectObject(  
    [in] HDC      hdc,  
    [in] HGDIOBJ h  
)
```

Parameters

[in] hdc

A handle to the DC.

[in] h

A handle to the object to be selected. The specified object must have been created by using one of the following functions.

OBJECT	FUNCTIONS
Bitmap	CreateBitmap , CreateBitmapIndirect , CreateCompatibleBitmap , CreateDIBitmap , CreateDIBSection Bitmaps can only be selected into memory DC's. A single bitmap cannot be selected into more than one DC at the same time.
Brush	CreateBrushIndirect , CreateDIBPatternBrush , CreateDIBPatternBrushPt , CreateHatchBrush , CreatePatternBrush , CreateSolidBrush
Font	CreateFont , CreateFontIndirect
Pen	CreatePen , CreatePenIndirect
Region	CombineRgn , CreateEllipticRgn , CreateEllipticRgnIndirect , CreatePolygonRgn , CreateRectRgn , CreateRectRgnIndirect

Return value

If the selected object is not a region and the function succeeds, the return value is a handle to the object being replaced. If the selected object is a region and the function succeeds, the return value is one of the following values.

VALUE	MEANING
SIMPLEREGION	Region consists of a single rectangle.
COMPLEXREGION	Region consists of more than one rectangle.
NULLRGN	Region is empty.

If an error occurs and the selected object is not a region, the return value is **NULL**. Otherwise, it is **HGDI_ERROR**.

Remarks

This function returns the previously selected object of the specified type. An application should always replace a new object with the original, default object after it has finished drawing with the new object.

An application cannot select a single bitmap into more than one DC at a time.

ICM: If the object being selected is a brush or a pen, color management is performed.

Examples

For an example, see [Setting the Pen or Brush Color](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CombineRgn](#)

[CreateBitmap](#)

[CreateBitmapIndirect](#)

[CreateBrushIndirect](#)

[CreateCompatibleBitmap](#)

[CreateDIBPatternBrush](#)

[CreateDIBitmap](#)

[CreateEllipticRgn](#)

[CreateEllipticRgnIndirect](#)

[CreateFont](#)

[CreateFontIndirect](#)

[CreateHatchBrush](#)

[CreatePatternBrush](#)

[CreatePen](#)

[CreatePenIndirect](#)

[CreatePolygonRgn](#)

[CreateRectRgn](#)

[CreateRectRgnIndirect](#)

[CreateSolidBrush](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[SelectClipRgn](#)

[SelectPalette](#)

SelectPalette function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SelectPalette** function selects the specified logical palette into a device context.

Syntax

```
HPALETTE SelectPalette(
    [in] HDC      hdc,
    [in] HPALETTE hPal,
    [in] BOOL     bForceBkgd
);
```

Parameters

[in] *hdc*

A handle to the device context.

[in] *hPal*

A handle to the logical palette to be selected.

[in] *bForceBkgd*

Specifies whether the logical palette is forced to be a background palette. If this value is **TRUE**, the [RealizePalette](#) function causes the logical palette to be mapped to the colors already in the physical palette in the best possible way. This is always done, even if the window for which the palette is realized belongs to a thread without active focus.

If this value is **FALSE**, [RealizePalette](#) causes the logical palette to be copied into the device palette when the application is in the foreground. (If the *hdc* parameter is a memory device context, this parameter is ignored.)

Return value

If the function succeeds, the return value is a handle to the device context's previous logical palette.

If the function fails, the return value is **NULL**.

Remarks

An application can determine whether a device supports palette operations by calling the [GetDeviceCaps](#) function and specifying the **RASTERCAPS** constant.

An application can select a logical palette into more than one device context only if device contexts are compatible. Otherwise **SelectPalette** fails. To create a device context that is compatible with another device context, call [CreateCompatibleDC](#) with the first device context as the parameter. If a logical palette is selected into more than one device context, changes to the logical palette will affect all device contexts for which it is selected.

An application might call the **SelectPalette** function with the *bForceBackground* parameter set to **TRUE** if the child windows of a top-level window each realize their own palettes. However, only the child window that needs to realize its palette must set *bForceBackground* to **TRUE**; other child windows must set this value to **FALSE**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[CreateCompatibleDC](#)

[CreatePalette](#)

[GetDeviceCaps](#)

[RealizePalette](#)

SetArcDirection function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetArcDirection** sets the drawing direction to be used for arc and rectangle functions.

Syntax

```
int SetArcDirection(  
    [in] HDC hdc,  
    [in] int dir  
>;
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `dir`

The new arc direction. This parameter can be one of the following values.

VALUE	MEANING
<code>AD_COUNTERCLOCKWISE</code>	Figures drawn counterclockwise.
<code>AD_CLOCKWISE</code>	Figures drawn clockwise.

Return value

If the function succeeds, the return value specifies the old arc direction.

If the function fails, the return value is zero.

Remarks

The default direction is counterclockwise.

The **SetArcDirection** function specifies the direction in which the following functions draw:

- [Arc](#)
- [ArcTo](#)
- [Chord](#)
- [Ellipse](#)
- [Pie](#)
- [Rectangle](#)
- [RoundRect](#)

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

SetBitmapBits function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetBitmapBits** function sets the bits of color data for a bitmap to the specified values.

Note This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the [SetDIBits](#) function.

Syntax

```
LONG SetBitmapBits(
    [in] HBITMAP    hbm,
    [in] DWORD      cb,
    [in] const VOID *pvBits
);
```

Parameters

[in] *hbm*

A handle to the bitmap to be set. This must be a compatible bitmap (DDB).

[in] *cb*

The number of bytes pointed to by the */pBits* parameter.

[in] *pvBits*

A pointer to an array of bytes that contain color data for the specified bitmap.

Return value

If the function succeeds, the return value is the number of bytes used in setting the bitmap bits.

If the function fails, the return value is zero.

Remarks

The array identified by */pBits* must be WORD aligned.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[GetBitmapBits](#)

[SetDIBits](#)

SetBitmapDimensionEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetBitmapDimensionEx** function assigns preferred dimensions to a bitmap. These dimensions can be used by applications; however, they are not used by the system.

Syntax

```
BOOL SetBitmapDimensionEx(
    [in] HBITMAP hbm,
    [in] int      w,
    [in] int      h,
    [out] LPSIZE  lpsz
);
```

Parameters

[in] **hbm**

A handle to the bitmap. The bitmap cannot be a DIB-section bitmap.

[in] **w**

The width, in 0.1-millimeter units, of the bitmap.

[in] **h**

The height, in 0.1-millimeter units, of the bitmap.

[out] **lpsz**

A pointer to a **SIZE** structure to receive the previous dimensions of the bitmap. This pointer can be NULL.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

An application can retrieve the dimensions assigned to a bitmap with the **SetBitmapDimensionEx** function by calling the [GetBitmapDimensionEx](#) function.

The bitmap identified by *hBitmap* cannot be a DIB section, which is a bitmap created by the [CreateDIBSection](#) function. If the bitmap is a DIB section, the **SetBitmapDimensionEx** function fails.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateDIBSection](#)

[GetBitmapDimensionEx](#)

[SIZE](#)

SetBkColor function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetBkColor** function sets the current background color to the specified color value, or to the nearest physical color if the device cannot represent the specified color value.

Syntax

```
COLORREF SetBkColor(  
    [in] HDC      hdc,  
    [in] COLORREF color  
)
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `color`

The new background color. To make a `COLORREF` value, use the `RGB` macro.

Return value

If the function succeeds, the return value specifies the previous background color as a `COLORREF` value.

If the function fails, the return value is `CLR_INVALID`.

Remarks

This function fills the gaps between styled lines drawn using a pen created by the `CreatePen` function; it does not fill the gaps between styled lines drawn using a pen created by the `ExtCreatePen` function. The `SetBkColor` function also sets the background colors for `TextOut` and `ExtTextOut`.

If the background mode is `OPAQUE`, the background color is used to fill gaps between styled lines, gaps between hatched lines in brushes, and character cells. The background color is also used when converting bitmaps from color to monochrome and vice versa.

Examples

For an example, see "Example of Owner-Drawn Menu Items" in [Using Menus](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORREF](#)

[CreatePen](#)

[ExtCreatePen](#)

[GetBKColor](#)

[GetBkMode](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[SetBkMode](#)

SetBkMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetBkMode** function sets the background mix mode of the specified device context. The background mix mode is used with text, hatched brushes, and pen styles that are not solid lines.

Syntax

```
int SetBkMode(
    [in] HDC hdc,
    [in] int mode
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `mode`

The background mode. This parameter can be one of the following values.

VALUE	MEANING
OPAQUE	Background is filled with the current background color before the text, hatched brush, or pen is drawn.
TRANSPARENT	Background remains untouched.

Return value

If the function succeeds, the return value specifies the previous background mode.

If the function fails, the return value is zero.

Remarks

The **SetBkMode** function affects the line styles for lines drawn using a pen created by the [CreatePen](#) function. **SetBkMode** does not affect lines drawn using a pen created by the [ExtCreatePen](#) function.

Examples

To see how to make the background of a hatch brush transparent or opaque, refer to the example shown in the [CreateHatchBrush](#) topic.

The next example draws a string 36 times, rotating it 10 degrees counterclockwise each time. It also sets the background mode to transparent to make the text visible.

```
#include "strsafe.h"
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
```

```
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;

    switch (message)
    {

        case WM_PAINT:
        {
            hdc = BeginPaint(hWnd, &ps);
            RECT rc;
            int angle;
            HGDIOBJ hfnt, hfntPrev;
            WCHAR lpszRotate[22] = TEXT("String to be rotated.");
            HRESULT hr;
            size_t pcch = 22;

            // Allocate memory for a LOGFONT structure.

            PLOGFONT plf = (PLOGFONT) LocalAlloc(LPTR, sizeof(LOGFONT));

            // Specify a font typeface name and weight.

            hr = StringCchCopy(plf->lfFaceName, 6, TEXT("Arial"));
            if (FAILED(hr))
            {
                // TODO: write error handler
            }

            plf->lfWeight = FW_NORMAL;

            // Retrieve the client-rectangle dimensions.

            GetClientRect(hWnd, &rc);

            // Set the background mode to transparent for the
            // text-output operation.

            SetBkMode(hdc, TRANSPARENT);

            // Draw the string 36 times, rotating 10 degrees
            // counter-clockwise each time.

            for (angle = 0; angle < 3600; angle += 100)
            {
                plf->lfEscapement = angle;
                hfnt = CreateFontIndirect(plf);
                hfntPrev = SelectObject(hdc, hfnt);

                //
                // The StringCchLength call is fitted to the lpszRotate string
                //
                hr = StringCchLength(lpszRotate, 22, &pcch);
                if (FAILED(hr))
                {
                    // TODO: write error handler
                }
                TextOut(hdc, rc.right / 2, rc.bottom / 2,
                        lpszRotate, pcch);
                SelectObject(hdc, hfntPrev);
                DeleteObject(hfnt);
            }

            // Reset the background mode to its default.

            SetBkMode(hdc, OPAQUE);

            // Free the memory allocated for the LOGFONT structure.
        }
    }
}
```

```

// Free the memory allocated for the LOGFONT structure.

LocalFree((LOCALHANDLE) plf);
    EndPaint(hWnd, &ps);
    break;
}
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreatePen](#)

[ExtCreatePen](#)

[GetBkMode](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

SetBoundsRect function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetBoundsRect** function controls the accumulation of bounding rectangle information for the specified device context. The system can maintain a bounding rectangle for all drawing operations. An application can examine and set this rectangle. The drawing boundaries are useful for invalidating bitmap caches.

Syntax

```
UINT SetBoundsRect(
    [in] HDC      hdc,
    [in] const RECT *lprect,
    [in] UINT      flags
);
```

Parameters

[in] `hdc`

A handle to the device context for which to accumulate bounding rectangles.

[in] `lprect`

A pointer to a [RECT](#) structure used to set the bounding rectangle. Rectangle dimensions are in logical coordinates. This parameter can be **NULL**.

[in] `flags`

Specifies how the new rectangle will be combined with the accumulated rectangle. This parameter can be one of more of the following values.

VALUE	MEANING
<code>DCB_ACCUMULATE</code>	Adds the rectangle specified by the <i>lprcBounds</i> parameter to the bounding rectangle (using a rectangle union operation). Using both <code>DCB_RESET</code> and <code>DCB_ACCUMULATE</code> sets the bounding rectangle to the rectangle specified by the <i>lprcBounds</i> parameter.
<code>DCB_DISABLE</code>	Turns off boundary accumulation.
<code>DCB_ENABLE</code>	Turns on boundary accumulation, which is disabled by default.
<code>DCB_RESET</code>	Clears the bounding rectangle.

Return value

If the function succeeds, the return value specifies the previous state of the bounding rectangle. This state can be

a combination of the following values.

VALUE	MEANING
DCB_DISABLE	Boundary accumulation is off.
DCB_ENABLE	Boundary accumulation is on. DCB_ENABLE and DCB_DISABLE are mutually exclusive.
DCB_RESET	Bounding rectangle is empty.
DCB_SET	Bounding rectangle is not empty. DCB_SET and DCB_RESET are mutually exclusive.

If the function fails, the return value is zero.

Remarks

The DCB_SET value is a combination of the bit values DCB_ACCUMULATE and DCB_RESET. Applications that check the DCB_RESET bit to determine whether the bounding rectangle is empty must also check the DCB_ACCUMULATE bit. The bounding rectangle is empty only if the DCB_RESET bit is 1 and the DCB_ACCUMULATE bit is 0.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GetBoundsRect](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

SetBrushOrgEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetBrushOrgEx** function sets the brush origin that GDI assigns to the next brush an application selects into the specified device context.

Syntax

```
BOOL SetBrushOrgEx(
    [in]  HDC      hdc,
    [in]  int      x,
    [in]  int      y,
    [out] LPPOINT  lppt
);
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **x**

The x-coordinate, in device units, of the new brush origin. If this value is greater than the brush width, its value is reduced using the modulus operator ($nXOrg \bmod$ brush width).

[in] **y**

The y-coordinate, in device units, of the new brush origin. If this value is greater than the brush height, its value is reduced using the modulus operator ($nYOrg \bmod$ brush height).

[out] **lppt**

A pointer to a [POINT](#) structure that receives the previous brush origin.

This parameter can be **NULL** if the previous brush origin is not required.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

A brush is a bitmap that the system uses to paint the interiors of filled shapes.

The brush origin is a pair of coordinates specifying the location of one pixel in the bitmap. The default brush origin coordinates are (0,0). For horizontal coordinates, the value 0 corresponds to the leftmost column of pixels; the width corresponds to the rightmost column. For vertical coordinates, the value 0 corresponds to the uppermost row of pixels; the height corresponds to the lowermost row.

The system automatically tracks the origin of all window-managed device contexts and adjusts their brushes as necessary to maintain an alignment of patterns on the surface. The brush origin that is set with this call is

relative to the upper-left corner of the client area.

An application should call [SetBrushOrgEx](#) after setting the bitmap stretching mode to HALFTONE by using [SetStretchBltMode](#). This must be done to avoid brush misalignment.

The system automatically tracks the origin of all window-managed device contexts and adjusts their brushes as necessary to maintain an alignment of patterns on the surface.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Brush Functions](#)

[Brushes Overview](#)

[GetBrushOrgEx](#)

[POINT](#)

[SelectObject](#)

[SetStretchBltMode](#)

[UnrealizeObject](#)

SetColorAdjustment function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetColorAdjustment** function sets the color adjustment values for a device context (DC) using the specified values.

Syntax

```
BOOL SetColorAdjustment(
    [in] HDC             hdc,
    [in] const COLORADJUSTMENT *lPCA
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lPCA`

A pointer to a [COLORADJUSTMENT](#) structure containing the color adjustment values.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The color adjustment values are used to adjust the input color of the source bitmap for calls to the [StretchBlt](#) and [StretchDIBits](#) functions when HALFTONE mode is set.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORADJUSTMENT](#)

[Color Functions](#)

[Colors Overview](#)

[GetColorAdjustment](#)

[SetStretchBltMode](#)

[StretchBlt](#)

[StretchDIBits](#)

SetDCBrushColor function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

SetDCBrushColor function sets the current device context (DC) brush color to the specified color value. If the device cannot represent the specified color value, the color is set to the nearest physical color.

Syntax

```
COLORREF SetDCBrushColor(  
    [in] HDC      hdc,  
    [in] COLORREF color  
)
```

Parameters

[in] `hdc`

A handle to the DC.

[in] `color`

The new brush color.

Return value

If the function succeeds, the return value specifies the previous DC brush color as a [COLORREF](#) value.

If the function fails, the return value is `CLR_INVALID`.

Remarks

When the stock `DC_BRUSH` is selected in a DC, all the subsequent drawings will be done using the DC brush color until the stock brush is deselected. The default `DC_BRUSH` color is `WHITE`.

The function returns the previous `DC_BRUSH` color, even if the stock brush `DC_BRUSH` is not selected in the DC; however, this will not be used in drawing operations until the stock `DC_BRUSH` is selected in the DC.

The [GetStockObject](#) function with an argument of `DC_BRUSH` or `DC_PEN` can be used interchangeably with the [SetDCPenColor](#) and [SetDCBrushColor](#) functions.

ICM: Color management is performed if ICM is enabled.

Examples

For an example of setting colors, see [Setting the Pen or Brush Color](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORREF](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetDCBrushColor](#)

SetDCPenColor function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

SetDCPenColor function sets the current device context (DC) pen color to the specified color value. If the device cannot represent the specified color value, the color is set to the nearest physical color.

Syntax

```
COLORREF SetDCPenColor(  
    [in] HDC      hdc,  
    [in] COLORREF color  
)
```

Parameters

[in] `hdc`

A handle to the DC.

[in] `color`

The new pen color.

Return value

If the function succeeds, the return value specifies the previous DC pen color as a [COLORREF](#) value. If the function fails, the return value is [CLR_INVALID](#).

Remarks

The function returns the previous DC_PEN color, even if the stock pen DC_PEN is not selected in the DC; however, this will not be used in drawing operations until the stock DC_PEN is selected in the DC.

The [GetStockObject](#) function with an argument of DC_BRUSH or DC_PEN can be used interchangeably with the [SetDCPenColor](#) and [SetDCBrushColor](#) functions.

ICM: Color management is performed if ICM is enabled.

Examples

For an example of setting colors, see [Setting the Pen or Brush Color](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORREF](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetDCPenColor](#)

SetDIBColorTable function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetDIBColorTable** function sets RGB (red, green, blue) color values in a range of entries in the color table of the DIB that is currently selected into a specified device context.

Syntax

```
UINT SetDIBColorTable(
    [in] HDC         hdc,
    [in] UINT        iStart,
    [in] UINT        cEntries,
    [in] const RGBQUAD *prgbq
);
```

Parameters

[in] `hdc`

A device context. A DIB must be selected into this device context.

[in] `iStart`

A zero-based color table index that specifies the first color table entry to set.

[in] `cEntries`

The number of color table entries to set.

[in] `prgbq`

A pointer to an array of **RGBQUAD** structures containing new color information for the DIB's color table.

Return value

If the function succeeds, the return value is the number of color table entries that the function sets.

If the function fails, the return value is zero.

Remarks

This function should be called to set the color table for DIBs that use 1, 4, or 8 bpp. The **BitCount** member of a bitmap's associated bitmap information header structure.

BITMAPINFOHEADER structure specifies the number of bits-per-pixel. Device-independent bitmaps with a **biBitCount** value greater than 8 do not have a color table.

The **bV5BitCount** member of a bitmap's associated **BITMAPV5HEADER** structure specifies the number of bits-per-pixel. Device-independent bitmaps with a **bV5BitCount** value greater than 8 do not have a color table.

ICM: No color management is performed.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPINFOHEADER](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateDIBSection](#)

[DIBSECTION](#)

[GetDIBColorTable](#)

[GetObject](#)

[RGBQUAD](#)

SetDIBits function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetDIBits** function sets the pixels in a compatible bitmap (DDB) using the color data found in the specified DIB.

Syntax

```
int SetDIBits(
    [in] HDC          hdc,
    [in] HBITMAP      hbm,
    [in] UINT         start,
    [in] UINT         cLines,
    [in] const VOID   *lpBits,
    [in] const BITMAPINFO *lpbmi,
    [in] UINT         ColorUse
);
```

Parameters

[in] `hdc`

A handle to a device context.

[in] `hbm`

A handle to the compatible bitmap (DDB) that is to be altered using the color data from the specified DIB.

[in] `start`

The starting scan line for the device-independent color data in the array pointed to by the *lpvBits* parameter.

[in] `cLines`

The number of scan lines found in the array containing device-independent color data.

[in] `lpBits`

A pointer to the DIB color data, stored as an array of bytes. The format of the bitmap values depends on the **biBitCount** member of the **BITMAPINFO** structure pointed to by the *lpbmi* parameter.

[in] `lpbmi`

A pointer to a **BITMAPINFO** structure that contains information about the DIB.

[in] `ColorUse`

Indicates whether the **bmiColors** member of the **BITMAPINFO** structure was provided and, if so, whether **bmiColors** contains explicit red, green, blue (RGB) values or palette indexes. The *fuColorUse* parameter must be one of the following values.

VALUE	MEANING
-------	---------

DIB_PAL_COLORS	The color table consists of an array of 16-bit indexes into the logical palette of the device context identified by the <i>hdc</i> parameter.
DIB_RGB_COLORS	The color table is provided and contains literal RGB values.

Return value

If the function succeeds, the return value is the number of scan lines copied.

If the function fails, the return value is zero.

This can be the following value.

RETURN CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	One or more of the input parameters is invalid.

Remarks

Optimal bitmap drawing speed is obtained when the bitmap bits are indexes into the system palette.

Applications can retrieve the system palette colors and indexes by calling the [GetSystemPaletteEntries](#) function. After the colors and indexes are retrieved, the application can create the DIB. For more information, see [System Palette](#).

The device context identified by the *hdc* parameter is used only if the DIB_PAL_COLORS constant is set for the *fuColorUse* parameter; otherwise it is ignored.

The bitmap identified by the *hbmp* parameter must not be selected into a device context when the application calls this function.

The scan lines must be aligned on a **DWORD** except for RLE-compressed bitmaps.

The origin for bottom-up DIBs is the lower-left corner of the bitmap; the origin for top-down DIBs is the upper-left corner of the bitmap.

ICM: Color management is performed if color management has been enabled with a call to [SetICMMode](#) with the *iEnableCM* parameter set to ICM_ON. If the bitmap specified by *lpbmi* has a [BITMAPV4HEADER](#) that specifies the gamma and endpoints members, or a [BITMAPV5HEADER](#) that specifies either the gamma and endpoints members or the profileData and profileSize members, then the call treats the bitmap's pixels as being expressed in the color space described by those members, rather than in the device context's source color space.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPINFO](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[GetDIBits](#)

[GetSystemPaletteEntries](#)

SetDIBitsToDevice function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **SetDIBitsToDevice** function sets the pixels in the specified rectangle on the device that is associated with the destination device context using color data from a DIB, JPEG, or PNG image.

Syntax

```
int SetDIBitsToDevice(
    [in] HDC         hdc,
    [in] int          xDest,
    [in] int          yDest,
    [in] DWORD        w,
    [in] DWORD        h,
    [in] int          xSrc,
    [in] int          ySrc,
    [in] UINT         StartScan,
    [in] UINT         cLines,
    [in] const VOID   *lpvBits,
    [in] const BITMAPINFO *lpbmi,
    [in] UINT         ColorUse
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `xDest`

The x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] `yDest`

The y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] `w`

The width, in logical units, of the image.

[in] `h`

The height, in logical units, of the image.

[in] `xSrc`

The x-coordinate, in logical units, of the lower-left corner of the image.

[in] `ySrc`

The y-coordinate, in logical units, of the lower-left corner of the image.

[in] `StartScan`

The starting scan line in the image.

[in] cLines

The number of DIB scan lines contained in the array pointed to by the *lpvBits* parameter.

[in] lpvBits

A pointer to the color data stored as an array of bytes. For more information, see the following Remarks section.

[in] lpbmi

A pointer to a [BITMAPINFO](#) structure that contains information about the DIB.

[in] ColorUse

Indicates whether the *bmiColors* member of the [BITMAPINFO](#) structure contains explicit red, green, blue (RGB) values or indexes into a palette. For more information, see the following Remarks section.

The *fuColorUse* parameter must be one of the following values.

VALUE	MEANING
DIB_PAL_COLORS	The color table consists of an array of 16-bit indexes into the currently selected logical palette.
DIB_RGB_COLORS	The color table contains literal RGB values.

Return value

If the function succeeds, the return value is the number of scan lines set.

If zero scan lines are set (such as when *dwHeight* is 0) or the function fails, the function returns zero.

If the driver cannot support the JPEG or PNG file image passed to [SetDIBitsToDevice](#), the function will fail and return GDI_ERROR. If failure does occur, the application must fall back on its own JPEG or PNG support to decompress the image into a bitmap, and then pass the bitmap to [SetDIBitsToDevice](#).

Remarks

Optimal bitmap drawing speed is obtained when the bitmap bits are indexes into the system palette.

Applications can retrieve the system palette colors and indexes by calling the [GetSystemPaletteEntries](#) function. After the colors and indexes are retrieved, the application can create the DIB. For more information about the system palette, see [Colors](#).

The scan lines must be aligned on a **DWORD** except for RLE-compressed bitmaps.

The origin of a bottom-up DIB is the lower-left corner of the bitmap; the origin of a top-down DIB is the upper-left corner.

To reduce the amount of memory required to set bits from a large DIB on a device surface, an application can band the output by repeatedly calling [SetDIBitsToDevice](#), placing a different portion of the bitmap into the *lpvBits* array each time. The values of the *uStartScan* and *cScanLines* parameters identify the portion of the bitmap contained in the *lpvBits* array.

The [SetDIBitsToDevice](#) function returns an error if it is called by a process that is running in the background while a full-screen MS-DOS session runs in the foreground.

- If the *biCompression* member of [BITMAPINFOHEADER](#) is BI_JPEG or BI_PNG, *lpvBits* points to a buffer

containing a JPEG or PNG image. The **biSizeImage** member of specifies the size of the buffer. The **fuColorUse** parameter must be set to DIB_RGB_COLORS.

- To ensure proper metafile spooling while printing, applications must call the **CHECKJPEGFORMAT** or **CHECKPNGFORMAT** escape to verify that the printer recognizes the JPEG or PNG image, respectively, before calling **SetDIBitsToDevice**.

ICM: Color management is performed if color management has been enabled with a call to **SetICMMODE** with the *iEnableCM* parameter set to ICM_ON. If the bitmap specified by *lpbmi* has a **BITMAPV4HEADER** that specifies the gamma and endpoints members, or a **BITMAPV5HEADER** that specifies either the gamma and endpoints members or the profileData and profileSize members, then the call treats the bitmap's pixels as being expressed in the color space described by those members, rather than in the device context's source color space.

Examples

For an example, see [Testing a Printer for JPEG or PNG Support](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPINFO](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[GetSystemPaletteEntries](#)

[SetDIBits](#)

[StretchDIBits](#)

SetEnhMetaFileBits function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetEnhMetaFileBits** function creates a memory-based enhanced-format metafile from the specified data.

Syntax

```
HENHMETAFILE SetEnhMetaFileBits(
    [in] UINT      nSize,
    [in] const BYTE *pb
);
```

Parameters

[in] **nSize**

Specifies the size, in bytes, of the data provided.

[in] **pb**

Pointer to a buffer that contains enhanced-metafile data. (It is assumed that the data in the buffer was obtained by calling the [GetEnhMetaFileBits](#) function.)

Return value

If the function succeeds, the return value is a handle to a memory-based enhanced metafile.

If the function fails, the return value is **NULL**.

Remarks

When the application no longer needs the enhanced-metafile handle, it should delete the handle by calling the [DeleteEnhMetaFile](#) function.

The **SetEnhMetaFileBits** function does not accept metafile data in the Windows format. To import Windows-format metafiles, use the [SetWinMetaFileBits](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[DeleteEnhMetaFile](#)

[GetEnhMetaFileBits](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SetWinMetaFileBits](#)

SetGraphicsMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetGraphicsMode** function sets the graphics mode for the specified device context.

Syntax

```
int SetGraphicsMode(  
    [in] HDC hdc,  
    [in] int iMode  
>;
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **iMode**

The graphics mode. This parameter can be one of the following values.

VALUE	MEANING
GM_COMPATIBLE	Sets the graphics mode that is compatible with 16-bit Windows. This is the default mode. If this value is specified, the application can only modify the world-to-device transform by calling functions that set window and viewport extents and origins, but not by using SetWorldTransform or ModifyWorldTransform ; calls to those functions will fail. Examples of functions that set window and viewport extents and origins are SetViewportExtEx and SetWindowExtEx .
GM_ADVANCED	Sets the advanced graphics mode that allows world transformations. This value must be specified if the application will set or modify the world transformation for the specified device context. In this mode all graphics, including text output, fully conform to the world-to-device transformation specified in the device context.

Return value

If the function succeeds, the return value is the old graphics mode.

If the function fails, the return value is zero.

Remarks

There are three areas in which graphics output differs according to the graphics mode:

1. Text Output: In the GM_COMPATIBLE mode, TrueType (or vector font) text output behaves much the same way as raster font text output with respect to the world-to-device transformations in the DC. The TrueType text is always written from left to right and right side up, even if the rest of the graphics will be flipped on the x or y

axis. Only the height of the TrueType (or vector font) text is scaled. The only way to write text that is not horizontal in the GM_COMPATIBLE mode is to specify nonzero escapement and orientation for the logical font selected in this device context.

In the GM_ADVANCED mode, TrueType (or vector font) text output fully conforms to the world-to-device transformation in the device context. The raster fonts only have very limited transformation capabilities (stretching by some integer factors). Graphics device interface (GDI) tries to produce the best output it can with raster fonts for nontrivial transformations.

2. Rectangle Exclusion: If the default GM_COMPATIBLE graphics mode is set, the system excludes bottom and rightmost edges when it draws rectangles.

The GM_ADVANCED graphics mode is required if applications want to draw rectangles that are lower-right inclusive.

3. Arc Drawing: If the default GM_COMPATIBLE graphics mode is set, GDI draws arcs using the current arc direction in the device space. With this convention, arcs do not respect page-to-device transforms that require a flip along the x or y axis.

If the GM_ADVANCED graphics mode is set, GDI always draws arcs in the counterclockwise direction in logical space. This is equivalent to the statement that, in the GM_ADVANCED graphics mode, both arc control points and arcs themselves fully respect the device context's world-to-device transformation.

Examples

For an example, see [Using Coordinate Spaces and Transformations](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[CreateDC](#)

[GetArcDirection](#)

[GetDC](#)

[GetGraphicsMode](#)

[ModifyWorldTransform](#)

[SetArcDirection](#)

[SetViewportExtEx](#)

[SetViewportExtent](#)

[SetWindowExtEx](#)

[SetWindowExtent](#)

[SetWorldTransform](#)

SetLayout function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetLayout** function changes the layout of a device context (DC).

Syntax

```
DWORD SetLayout(
    [in] HDC    hdc,
    [in] DWORD  l
);
```

Parameters

[in] **hdc**

A handle to the DC.

[in] **l**

The DC layout. This parameter can be one or more of the following values.

VALUE	MEANING
LAYOUT_BITMAPORIENTATIONPRESERVED	Disables any reflection during BitBlt and StretchBlt operations.
LAYOUT_RTL	Sets the default horizontal layout to be right to left.

Return value

If the function succeeds, it returns the previous layout of the DC.

If the function fails, it returns GDI_ERROR.

Remarks

The layout specifies the order in which text and graphics are revealed in a window or a device context. The default is left to right. The **SetLayout** function changes this to be right to left, which is the standard in Arabic and Hebrew cultures.

Once the LAYOUT_RTL flag is selected, flags normally specifying right or left are reversed. To avoid confusion, consider defining alternate words for standard flags, such as those in the following table.

STANDARD FLAG	SUGGESTED ALTERNATE NAME
WS_EX_RIGHT	WS_EX_TRAILING
WS_EX_RTLREADING	WS_EX_REVERSEREADING

WS_EX_LEFTSCROLLBAR	WS_EX_LEADSCROLLBAR
ES_LEFT	ES_LEAD
ES_RIGHT	ES_TRAIL
EC_LEFTMARGIN	EC_LEADMARGIN
EC_RIGHTMARGIN	EC_TRAILMARGIN

SetLayout cannot modify drawing directly into the bits of a DIB.

For more information, see "Window Layout and Mirroring" in [Window Features](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetLayout](#)

SetMapMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetMapMode** function sets the mapping mode of the specified device context. The mapping mode defines the unit of measure used to transform page-space units into device-space units, and also defines the orientation of the device's x and y axes.

Syntax

```
int SetMapMode(  
    [in] HDC hdc,  
    [in] int iMode  
>;
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `iMode`

The new mapping mode. This parameter can be one of the following values.

VALUE	MEANING
<code>MM_ANISOTROPIC</code>	Logical units are mapped to arbitrary units with arbitrarily scaled axes. Use the SetWindowExtEx and SetViewportExtEx functions to specify the units, orientation, and scaling.
<code>MM_HIENGLISH</code>	Each logical unit is mapped to 0.001 inch. Positive x is to the right; positive y is up.
<code>MM HIMETRIC</code>	Each logical unit is mapped to 0.01 millimeter. Positive x is to the right; positive y is up.
<code>MM_ISOTROPIC</code>	Logical units are mapped to arbitrary units with equally scaled axes; that is, one unit along the x-axis is equal to one unit along the y-axis. Use the SetWindowExtEx and SetViewportExtEx functions to specify the units and the orientation of the axes. Graphics device interface (GDI) makes adjustments as necessary to ensure the x and y units remain the same size (When the window extent is set, the viewport will be adjusted to keep the units isotropic).
<code>MM LOENGLISH</code>	Each logical unit is mapped to 0.01 inch. Positive x is to the right; positive y is up.
<code>MM LOMETRIC</code>	Each logical unit is mapped to 0.1 millimeter. Positive x is to the right; positive y is up.

MM_TEXT	Each logical unit is mapped to one device pixel. Positive x is to the right; positive y is down.
MM_TWIPS	Each logical unit is mapped to one twentieth of a printer's point (1/1440 inch, also called a twip). Positive x is to the right; positive y is up.

Return value

If the function succeeds, the return value identifies the previous mapping mode.

If the function fails, the return value is zero.

Remarks

The MM_TEXT mode allows applications to work in device pixels, whose size varies from device to device.

The MM_HIENGLISH, MM_HIMETRIC, MM_LOENGLISH, MM_LOMETRIC, and MM_TWIPS modes are useful for applications drawing in physically meaningful units (such as inches or millimeters).

The MM_ISOTROPIC mode ensures a 1:1 aspect ratio.

The MM_ANISOTROPIC mode allows the x-coordinates and y-coordinates to be adjusted independently.

Examples

For an example, see [Using Coordinate Spaces and Transformations](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetMapMode](#)

[SetViewportExtEx](#)

[SetViewportOrgEx](#)

[SetWindowExtEx](#)

[SetWindowOrgEx](#)

SetMapperFlags function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetMapperFlags** function alters the algorithm the font mapper uses when it maps logical fonts to physical fonts.

Syntax

```
DWORD SetMapperFlags(
    [in] HDC    hdc,
    [in] DWORD flags
);
```

Parameters

[in] `hdc`

A handle to the device context that contains the font-mapper flag.

[in] `flags`

Specifies whether the font mapper should attempt to match a font's aspect ratio to the current device's aspect ratio. If bit zero is set, the mapper selects only matching fonts.

Return value

If the function succeeds, the return value is the previous value of the font-mapper flag.

If the function fails, the return value is GDI_ERROR.

Remarks

If the *dwFlag* parameter is set and no matching fonts exist, Windows chooses a new aspect ratio and retrieves a font that matches this ratio.

The remaining bits of the *dwFlag* parameter must be zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetAspectRatioFilterEx](#)

SetMetaFileBitsEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetMetaFileBitsEx** function creates a memory-based Windows-format metafile from the supplied data.

Note This function is provided only for compatibility with Windows-format metafiles. Enhanced-format metafiles provide superior functionality and are recommended for new applications. The corresponding function for an enhanced-format metafile is [SetEnhMetaFileBits](#).

Syntax

```
HMETAFILE SetMetaFileBitsEx(  
    [in] UINT      cbBuffer,  
    [in] const BYTE *lpData  
)
```

Parameters

[in] cbBuffer

Specifies the size, in bytes, of the Windows-format metafile.

[in] lpData

Pointer to a buffer that contains the Windows-format metafile. (It is assumed that the data was obtained by using the [GetMetaFileBitsEx](#) function.)

Return value

If the function succeeds, the return value is a handle to a memory-based Windows-format metafile.

If the function fails, the return value is **NULL**.

Remarks

To convert a Windows-format metafile into an enhanced-format metafile, use the [SetWinMetaFileBits](#) function.

When the application no longer needs the metafile handle returned by **SetMetaFileBitsEx**, it should delete it by calling the [DeleteMetaFile](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteMetaFile](#)

[GetMetaFileBitsEx](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[SetEnhMetaFileBits](#)

[SetWinMetaFileBits](#)

SetMetaRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetMetaRgn** function intersects the current clipping region for the specified device context with the current metaregion and saves the combined region as the new metaregion for the specified device context. The clipping region is reset to a null region.

Syntax

```
int SetMetaRgn(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

A handle to the device context.

Return value

The return value specifies the new clipping region's complexity and can be one of the following values.

RETURN CODE	DESCRIPTION
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred. (The previous clipping region is unaffected.)

Remarks

The current clipping region of a device context is defined by the intersection of its clipping region and its metaregion.

The **SetMetaRgn** function should only be called after an application's original device context was saved by calling the [SaveDC](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Clipping Functions](#)

[Clipping Overview](#)

[GetMetaRgn](#)

[SaveDC](#)

SetMiterLimit function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetMiterLimit** function sets the limit for the length of miter joins for the specified device context.

Syntax

```
BOOL SetMiterLimit(
    [in]    HDC     hdc,
    [in]    FLOAT   limit,
    [out]   PFLOAT  old
);
```

Parameters

[in] **hdc**

Handle to the device context.

[in] **limit**

Specifies the new miter limit for the device context.

[out] **old**

Pointer to a floating-point value that receives the previous miter limit. If this parameter is **NULL**, the previous miter limit is not returned.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The miter length is defined as the distance from the intersection of the line walls on the inside of the join to the intersection of the line walls on the outside of the join. The miter limit is the maximum allowed ratio of the miter length to the line width.

The default miter limit is 10.0.

Note Setting *eNewLimit* to a float value less than 1.0f will cause the function to fail.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtCreatePen](#)

[GetMiterLimit](#)

[Path Functions](#)

[Paths Overview](#)

SetPaletteEntries function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetPaletteEntries** function sets RGB (red, green, blue) color values and flags in a range of entries in a logical palette.

Syntax

```
UINT SetPaletteEntries(
    [in] HPALETTE          hpal,
    [in] UINT               iStart,
    [in] UINT               cEntries,
    [in] const PALETTEENTRY *pPalEntries
);
```

Parameters

[in] `hpal`

A handle to the logical palette.

[in] `iStart`

The first logical-palette entry to be set.

[in] `cEntries`

The number of logical-palette entries to be set.

[in] `pPalEntries`

A pointer to the first member of an array of **PALETTEENTRY** structures containing the RGB values and flags.

Return value

If the function succeeds, the return value is the number of entries that were set in the logical palette.

If the function fails, the return value is zero.

Remarks

An application can determine whether a device supports palette operations by calling the **GetDeviceCaps** function and specifying the RASTERCAPS constant.

Even if a logical palette has been selected and realized, changes to the palette do not affect the physical palette in the surface. **RealizePalette** must be called again to set the new logical palette into the surface.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[GetDeviceCaps](#)

[GetPaletteEntries](#)

[PALETTEENTRY](#)

[RealizePalette](#)

SetPixel function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetPixel** function sets the pixel at the specified coordinates to the specified color.

Syntax

```
COLORREF SetPixel(
    [in] HDC      hdc,
    [in] int      x,
    [in] int      y,
    [in] COLORREF color
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The x-coordinate, in logical units, of the point to be set.

[in] `y`

The y-coordinate, in logical units, of the point to be set.

[in] `color`

The color to be used to paint the point. To create a **COLORREF** color value, use the **RGB** macro.

Return value

If the function succeeds, the return value is the RGB value that the function sets the pixel to. This value may differ from the color specified by *crColor*; that occurs when an exact match for the specified color cannot be found.

If the function fails, the return value is -1.

This can be the following value.

RETURN CODE	DESCRIPTION
<code>ERROR_INVALID_PARAMETER</code>	One or more of the input parameters is invalid.

Remarks

The function fails if the pixel coordinates lie outside of the current clipping region.

Not all devices support the **SetPixel** function. For more information, see [GetDeviceCaps](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[COLORREF](#)

[GetDeviceCaps](#)

[GetPixel](#)

[RGB](#)

[SetPixelV](#)

SetPixelV function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetPixelV** function sets the pixel at the specified coordinates to the closest approximation of the specified color. The point must be in the clipping region and the visible part of the device surface.

Syntax

```
BOOL SetPixelV(
    [in] HDC      hdc,
    [in] int      x,
    [in] int      y,
    [in] COLORREF color
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The x-coordinate, in logical units, of the point to be set.

[in] `y`

The y-coordinate, in logical units, of the point to be set.

[in] `color`

The color to be used to paint the point. To create a `COLORREF` color value, use the `RGB` macro.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Not all devices support the **SetPixelV** function. For more information, see the description of the `RC_BITBLT` capability in the [GetDeviceCaps](#) function.

SetPixelV is faster than [SetPixel](#) because it does not need to return the color value of the point actually painted.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[COLORREF](#)

[GetDeviceCaps](#)

[RGB](#)

[SetPixel](#)

SetPolyFillMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetPolyFillMode** function sets the polygon fill mode for functions that fill polygons.

Syntax

```
int SetPolyFillMode(  
    [in] HDC hdc,  
    [in] int mode  
>;
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **mode**

The new fill mode. This parameter can be one of the following values.

VALUE	MEANING
ALTERNATE	Selects alternate mode (fills the area between odd-numbered and even-numbered polygon sides on each scan line).
WINDING	Selects winding mode (fills any region with a nonzero winding value).

Return value

The return value specifies the previous filling mode. If an error occurs, the return value is zero.

Remarks

In general, the modes differ only in cases where a complex, overlapping polygon must be filled (for example, a five-sided polygon that forms a five-pointed star with a pentagon in the center). In such cases, ALTERNATE mode fills every other enclosed region within the polygon (that is, the points of the star), but WINDING mode fills all regions (that is, the points and the pentagon).

When the fill mode is ALTERNATE, GDI fills the area between odd-numbered and even-numbered polygon sides on each scan line. That is, GDI fills the area between the first and second side, between the third and fourth side, and so on.

When the fill mode is WINDING, GDI fills any region that has a nonzero winding value. This value is defined as the number of times a pen used to draw the polygon would go around the region. The direction of each edge of the polygon is important.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GetPolyFillMode](#)

[Region Functions](#)

[Regions Overview](#)

SetRectRgn function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetRectRgn** function converts a region into a rectangular region with the specified coordinates.

Syntax

```
BOOL SetRectRgn(
    [in] HRGN hrgn,
    [in] int left,
    [in] int top,
    [in] int right,
    [in] int bottom
);
```

Parameters

[in] `hrgn`

Handle to the region.

[in] `left`

Specifies the x-coordinate of the upper-left corner of the rectangular region in logical units.

[in] `top`

Specifies the y-coordinate of the upper-left corner of the rectangular region in logical units.

[in] `right`

Specifies the x-coordinate of the lower-right corner of the rectangular region in logical units.

[in] `bottom`

Specifies the y-coordinate of the lower-right corner of the rectangular region in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The region does not include the lower and right boundaries of the rectangle.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateRectRgn](#)

[Region Functions](#)

[Regions Overview](#)

SetROP2 function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetROP2** function sets the current foreground mix mode. GDI uses the foreground mix mode to combine pens and interiors of filled objects with the colors already on the screen. The foreground mix mode defines how colors from the brush or pen and the colors in the existing image are to be combined.

Syntax

```
int SetROP2(
    [in] HDC hdc,
    [in] int rop2
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `rop2`

The mix mode. This parameter can be one of the following values.

MIX MODE	MEANING
R2_BLACK	Pixel is always 0.
R2_COPYPEN	Pixel is the pen color.
R2_MASKNOTPEN	Pixel is a combination of the colors common to both the screen and the inverse of the pen.
R2_MASKPEN	Pixel is a combination of the colors common to both the pen and the screen.
R2_MASKPENNOD	Pixel is a combination of the colors common to both the pen and the inverse of the screen.
R2_MERGENOTPEN	Pixel is a combination of the screen color and the inverse of the pen color.
R2_MERGEOPEN	Pixel is a combination of the pen color and the screen color.
R2_MERGEOPENNOD	Pixel is a combination of the pen color and the inverse of the screen color.

R2_NOP	Pixel remains unchanged.
R2_NOT	Pixel is the inverse of the screen color.
R2_NOTCOPYPEN	Pixel is the inverse of the pen color.
R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN color.
R2_NOTMERGESEN	Pixel is the inverse of the R2_MERGESEN color.
R2_NOTXORPEN	Pixel is the inverse of the R2_XORPEN color.
R2_WHITE	Pixel is always 1.
R2_XORPEN	Pixel is a combination of the colors in the pen and in the screen, but not in both.

Return value

If the function succeeds, the return value specifies the previous mix mode.

If the function fails, the return value is zero.

Remarks

Mix modes define how GDI combines source and destination colors when drawing with the current pen. The mix modes are binary raster operation codes, representing all possible Boolean functions of two variables, using the binary operations AND, OR, and XOR (exclusive OR), and the unary operation NOT. The mix mode is for raster devices only; it is not available for vector devices.

Examples

For an example, see [Using Rectangles](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GetROP2](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

SetStretchBltMode function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetStretchBltMode** function sets the bitmap stretching mode in the specified device context.

Syntax

```
int SetStretchBltMode(  
    [in] HDC hdc,  
    [in] int mode  
>;
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **mode**

The stretching mode. This parameter can be one of the following values.

VALUE	MEANING
BLACKONWHITE	Performs a Boolean AND operation using the color values for the eliminated and existing pixels. If the bitmap is a monochrome bitmap, this mode preserves black pixels at the expense of white pixels.
COLORONCOLOR	Deletes the pixels. This mode deletes all eliminated lines of pixels without trying to preserve their information.
HALFTONE	Maps pixels from the source rectangle into blocks of pixels in the destination rectangle. The average color over the destination block of pixels approximates the color of the source pixels. After setting the HALFTONE stretching mode, an application must call the SetBrushOrgEx function to set the brush origin. If it fails to do so, brush misalignment occurs.
STRETCH_ANDSCANS	Same as BLACKONWHITE.
STRETCH_DELETESCANS	Same as COLORONCOLOR.
STRETCH_HALFTONE	Same as HALFTONE.

STRETCH_ORSCANS	Same as WHITEONBLACK.
WHITEONBLACK	Performs a Boolean OR operation using the color values for the eliminated and existing pixels. If the bitmap is a monochrome bitmap, this mode preserves white pixels at the expense of black pixels.

Return value

If the function succeeds, the return value is the previous stretching mode.

If the function fails, the return value is zero.

This function can return the following value.

RETURN CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	One or more of the input parameters is invalid.

Remarks

The stretching mode defines how the system combines rows or columns of a bitmap with existing pixels on a display device when an application calls the [StretchBlt](#) function.

The BLACKONWHITE (STRETCH_ANDSCANS) and WHITEONBLACK (STRETCH_ORSCANS) modes are typically used to preserve foreground pixels in monochrome bitmaps. The COLORONCOLOR (STRETCH_DELETERCANS) mode is typically used to preserve color in color bitmaps.

The HALFTONE mode is slower and requires more processing of the source image than the other three modes; but produces higher quality images. Also note that [SetBrushOrgEx](#) must be called after setting the HALFTONE mode to avoid brush misalignment.

Additional stretching modes might also be available depending on the capabilities of the device driver.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[GetStretchBltMode](#)

[SetBrushOrgEx](#)

[StretchBlt](#)

SetSystemPaletteUse function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetSystemPaletteUse** function allows an application to specify whether the system palette contains 2 or 20 static colors. The default system palette contains 20 static colors. (Static colors cannot be changed when an application realizes a logical palette.)

Syntax

```
UINT SetSystemPaletteUse(
    [in] HDC hdc,
    [in] UINT use
);
```

Parameters

[in] hdc

A handle to the device context. This device context must refer to a device that supports color palettes.

[in] use

The new use of the system palette. This parameter can be one of the following values.

VALUE	MEANING
SYSPAL_NOSTATIC	The system palette contains two static colors (black and white).
SYSPAL_NOSTATIC256	The system palette contains no static colors.
SYSPAL_STATIC	The system palette contains static colors that will not change when an application realizes its logical palette.

Return value

If the function succeeds, the return value is the previous system palette. It can be either SYSPAL_NOSTATIC, SYSPAL_NOSTATIC256, or SYSPAL_STATIC.

If the function fails, the return value is SYSPAL_ERROR.

Remarks

An application can determine whether a device supports palette operations by calling the [GetDeviceCaps](#) function and specifying the RASTERCAPS constant.

When an application window moves to the foreground and the SYSPAL_NOSTATIC value is set, the application must call the [GetSysColor](#) function to save the current system colors setting. It must also call [SetSysColors](#) to set reasonable values using only black and white. When the application returns to the background or terminates,

the previous system colors must be restored.

If the function returns SYSPAL_ERROR, the specified device context is invalid or does not support color palettes.

An application must call this function only when its window is maximized and has the input focus.

If an application calls **SetSystemPaletteUse** with *uUsage* set to SYSPAL_NOSTATIC, the system continues to set aside two entries in the system palette for pure white and pure black, respectively.

After calling this function with *uUsage* set to SYSPAL_NOSTATIC, an application must take the following steps:

1. Realize the logical palette.
2. Call the [GetSysColor](#) function to save the current system-color settings.
3. Call the [SetSysColors](#) function to set the system colors to reasonable values using black and white. For example, adjacent or overlapping items (such as window frames and borders) should be set to black and white, respectively.
4. Send the [WM_SYSCOLORCHANGE](#) message to other top-level windows to allow them to be redrawn with the new system colors.

When the application's window loses focus or closes, the application must perform the following steps:

1. Call **SetSystemPaletteUse** with the *uUsage* parameter set to SYSPAL_STATIC.
2. Realize the logical palette.
3. Restore the system colors to their previous values.
4. Send the [WM_SYSCOLORCHANGE](#) message.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[GetDeviceCaps](#)

[GetSysColor](#)

[GetSystemPaletteUse](#)

[SetSysColors](#)

SetTextAlign function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetTextAlign** function sets the text-alignment flags for the specified device context.

Syntax

```
UINT SetTextAlign(
    [in] HDC hdc,
    [in] UINT align
);
```

Parameters

[in] **hdc**

A handle to the device context.

[in] **align**

The text alignment by using a mask of the values in the following list. Only one flag can be chosen from those that affect horizontal and vertical alignment. In addition, only one of the two flags that alter the current position can be chosen.

VALUE	MEANING
TA_BASELINE	The reference point will be on the base line of the text.
TA_BOTTOM	The reference point will be on the bottom edge of the bounding rectangle.
TA_TOP	The reference point will be on the top edge of the bounding rectangle.
TA_CENTER	The reference point will be aligned horizontally with the center of the bounding rectangle.
TA_LEFT	The reference point will be on the left edge of the bounding rectangle.
TA_RIGHT	The reference point will be on the right edge of the bounding rectangle.
TA_NOUPDATECP	The current position is not updated after each text output call. The reference point is passed to the text output function.

TA_RTLREADING	Middle East language edition of Windows: The text is laid out in right to left reading order, as opposed to the default left to right order. This applies only when the font selected into the device context is either Hebrew or Arabic.
TA_UPDATECP	The current position is updated after each text output call. The current position is used as the reference point.

When the current font has a vertical default base line, as with Kanji, the following values must be used instead of TA_BASELINE and TA_CENTER.

VALUE	MEANING
VTA_BASELINE	The reference point will be on the base line of the text.
VTA_CENTER	The reference point will be aligned vertically with the center of the bounding rectangle.

The default values are TA_LEFT, TA_TOP, and TA_NOUPDATECP.

Return value

If the function succeeds, the return value is the previous text-alignment setting.

If the function fails, the return value is GDI_ERROR.

Remarks

The [TextOut](#) and [ExtTextOut](#) functions use the text-alignment flags to position a string of text on a display or other device. The flags specify the relationship between a reference point and a rectangle that bounds the text. The reference point is either the current position or a point passed to a text output function.

The rectangle that bounds the text is formed by the character cells in the text string.

The best way to get left-aligned text is to use either

```
SetTextAlign (hdc, GetTextAlign(hdc) & (~TA_CENTER))
```

or

```
SetTextAlign (hdc, TA_LEFT | <other flags>)
```

You can also use [SetTextAlign](#) (hdc, TA_LEFT) for this purpose, but this loses any vertical or right-to-left settings.

Note You should not use [SetTextAlign](#) with TA_UPDATECP when you are using [ScriptStringOut](#), because selected text is not rendered correctly. If you must use this flag, you can unset and reset it as necessary to avoid the problem.

Examples

For an example, see [Setting the Text Alignment](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextAlign](#)

[ScriptStringOut](#)

[TextOut](#)

SetTextCharacterExtra function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetTextCharacterExtra** function sets the intercharacter spacing. Intercharacter spacing is added to each character, including break characters, when the system writes a line of text.

Syntax

```
int SetTextCharacterExtra(
    [in] HDC hdc,
    [in] int extra
);
```

Parameters

[in] *hdc*

A handle to the device context.

[in] *extra*

The amount of extra space, in logical units, to be added to each character. If the current mapping mode is not MM_TEXT, the *nCharExtra* parameter is transformed and rounded to the nearest pixel.

Return value

If the function succeeds, the return value is the previous intercharacter spacing.

If the function fails, the return value is 0x80000000.

Remarks

This function is supported mainly for compatibility with existing applications. New applications should generally avoid calling this function, because it is incompatible with complex scripts (scripts that require text shaping; Arabic script is an example of this).

The recommended approach is that instead of calling this function and then [TextOut](#), applications should call [ExtTextOut](#) and use its *lpDx* parameter to supply widths.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DrawText](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextCharacterExtra](#)

[TextOut](#)

SetTextColor function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetTextColor** function sets the text color for the specified device context to the specified color.

Syntax

```
COLORREF SetTextColor(  
    [in] HDC      hdc,  
    [in] COLORREF color  
)
```

Parameters

[in] hdc

A handle to the device context.

[in] color

The color of the text.

Return value

If the function succeeds, the return value is a color reference for the previous text color as a [COLORREF](#) value.

If the function fails, the return value is [CLR_INVALID](#).

Remarks

The text color is used to draw the face of each character written by the [TextOut](#) and [ExtTextOut](#) functions. The text color is also used in converting bitmaps from color to monochrome and vice versa.

Examples

For an example, see "Setting Fonts for Menu-Item Text Strings" in [Using Menus](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[BitBlt](#)

[COLORREF](#)

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextColor](#)

[RGB](#)

[SetBkColor](#)

[StretchBlt](#)

[TextOut](#)

SetTextJustification function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetTextJustification** function specifies the amount of space the system should add to the break characters in a string of text. The space is added when an application calls the [TextOut](#) or [ExtTextOut](#) functions.

Syntax

```
BOOL SetTextJustification(
    [in] HDC hdc,
    [in] int extra,
    [in] int count
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `extra`

The total extra space, in logical units, to be added to the line of text. If the current mapping mode is not MM_TEXT, the value identified by the *nBreakExtra* parameter is transformed and rounded to the nearest pixel.

[in] `count`

The number of break characters in the line.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The break character is usually the space character (ASCII 32), but it may be defined by a font as some other character. The [GetTextMetrics](#) function can be used to retrieve a font's break character.

The [TextOut](#) function distributes the specified extra space evenly among the break characters in the line.

The [GetTextExtentPoint32](#) function is always used with the **SetTextJustification** function. Sometimes the [GetTextExtentPoint32](#) function takes justification into account when computing the width of a specified line before justification, and sometimes it does not. For more details on this, see [GetTextExtentPoint32](#). This width must be known before an appropriate *nBreakExtra* value can be computed.

SetTextJustification can be used to justify a line that contains multiple strings in different fonts. In this case, each string must be justified separately.

Because rounding errors can occur during justification, the system keeps a running error term that defines the current error value. When justifying a line that contains multiple runs, [GetTextExtentPoint](#) automatically uses this error term when it computes the extent of the next run, allowing [TextOut](#) to blend the error into the new run.

After each line has been justified, this error term must be cleared to prevent it from being incorporated into the next line. The term can be cleared by calling [SetTextJustification](#) with *nBreakExtra* set to zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ExtTextOut](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

[TextOut](#)

SetViewportExtEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetViewportExtEx** function sets the horizontal and vertical extents of the viewport for a device context by using the specified values.

Syntax

```
BOOL SetViewportExtEx(
    [in]  HDC      hdc,
    [in]  int       x,
    [in]  int       y,
    [out] LPSIZE  lpsz
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The horizontal extent, in device units, of the viewport.

[in] `y`

The vertical extent, in device units, of the viewport.

[out] `lpsz`

A pointer to a [SIZE](#) structure that receives the previous viewport extents, in device units. If `/pSize` is **NULL**, this parameter is not used.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The *viewport* refers to the device coordinate system of the device space. The *extent* is the maximum value of an axis. This function sets the maximum values for the horizontal and vertical axes of the viewport in device coordinates (or pixels). When mapping between page space and device space, [SetWindowExtEx](#) and [SetViewportExtEx](#) determine the scaling factor between the window and the viewport. For more information, see [Transformation of Coordinate Spaces](#).

When the following mapping modes are set, calls to the [SetWindowExtEx](#) and [SetViewportExtEx](#) functions are ignored.

- `MM_HIENGLISH`
- `MM_HIMETRIC`

- MM_LOENGLISH
- MM_LOMETRIC
- MM_TEXT
- MM_TWIPS

When MM_ISOTROPIC mode is set, an application must call the [SetWindowExtEx](#) function before it calls [SetViewportExtEx](#). Note that for the MM_ISOTROPIC mode certain portions of a nonsquare screen may not be available for display because the logical units on both axes represent equal physical distances.

Examples

For an example, see [Invalidating the Client Area](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetViewportExtEx](#)

[SIZE](#)

[SetWindowExtEx](#)

SetViewportOrgEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetViewportOrgEx** function specifies which device point maps to the window origin (0,0).

Syntax

```
BOOL SetViewportOrgEx(
    [in]  HDC      hdc,
    [in]  int      x,
    [in]  int      y,
    [out] LPPOINT lppt
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The x-coordinate, in device units, of the new viewport origin.

[in] `y`

The y-coordinate, in device units, of the new viewport origin.

[out] `lppt`

A pointer to a **POINT** structure that receives the previous viewport origin, in device coordinates. If *lpPoint* is **NULL**, this parameter is not used.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

This function (along with **SetViewportExtEx** and **SetWindowExtEx**) helps define the mapping from the logical coordinate space (also known as a *window*) to the device coordinate space (the *viewport*). **SetViewportOrgEx** specifies which device point maps to the logical point (0,0). It has the effect of shifting the axes so that the logical point (0,0) no longer refers to the upper-left corner.

```
//map the logical point (0,0) to the device point (xViewOrg, yViewOrg)
SetViewportOrgEx ( hdc, xViewOrg, yViewOrg, NULL )
```

This is related to the **SetWindowOrgEx** function. Generally, you will use one function or the other, but not both. Regardless of your use of **SetWindowOrgEx** and **SetViewportOrgEx**, the device point (0,0) is always the

upper-left corner.

Examples

For an example, see [Redrawing in the Update Region](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetViewportOrgEx](#)

[POINT](#)

[SetWindowOrgEx](#)

SetWindowExtEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetWindowExtEx** function sets the horizontal and vertical extents of the window for a device context by using the specified values.

Syntax

```
BOOL SetWindowExtEx(
    [in]  HDC      hdc,
    [in]  int       x,
    [in]  int       y,
    [out] LPSIZE lpsz
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The window's horizontal extent in logical units.

[in] `y`

The window's vertical extent in logical units.

[out] `lpsz`

A pointer to a [SIZE](#) structure that receives the previous window extents, in logical units. If `lpSize` is **NULL**, this parameter is not used.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The *window* refers to the logical coordinate system of the page space. The *extent* is the maximum value of an axis. This function sets the maximum values for the horizontal and vertical axes of the window (in logical coordinates). When mapping between page space and device space, [SetViewportExtEx](#) and [SetWindowExtEx](#) determine the scaling factor between the window and the viewport. For more information, see [Transformation of Coordinate Spaces](#).

When the following mapping modes are set, calls to the [SetWindowExtEx](#) and [SetViewportExtEx](#) functions are ignored:

- `MM_HIENGLISH`
- `MM_HIMETRIC`

- MM_LOENGLISH
- MM_LOMETRIC
- MM_TEXT
- MM_TWIPS

When MM_ISOTROPIC mode is set, an application must call the [SetWindowExtEx](#) function before calling [SetViewportExtEx](#). Note that for the MM_ISOTROPIC mode, certain portions of a nonsquare screen may not be available for display because the logical units on both axes represent equal physical distances.

Examples

For an example, see [Invalidating the Client Area](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetWindowExtEx](#)

[SIZE](#)

[SetViewportExtEx](#)

SetWindowOrgEx function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetWindowOrgEx** function specifies which window point maps to the viewport origin (0,0).

Syntax

```
BOOL SetWindowOrgEx(
    [in]  HDC      hdc,
    [in]  int      x,
    [in]  int      y,
    [out] LPPOINT lppt
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The x-coordinate, in logical units, of the new window origin.

[in] `y`

The y-coordinate, in logical units, of the new window origin.

[out] `lppt`

A pointer to a **POINT** structure that receives the previous origin of the window, in logical units. If *lpPoint* is `NULL`, this parameter is not used.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

This helps define the mapping from the logical coordinate space (also known as a *window*) to the device coordinate space (the *viewport*). **SetWindowOrgEx** specifies which logical point maps to the device point (0,0). It has the effect of shifting the axes so that the logical point (0,0) no longer refers to the upper-left corner.

```
//map the logical point (xWinOrg, yWinOrg) to the device point (0,0)
SetWindowOrgEx (hdc, xWinOrg, yWinOrg, NULL)
```

This is related to the **SetViewportOrgEx** function. Generally, you will use one function or the other, but not both. Regardless of your use of **SetWindowOrgEx** and **SetViewportOrgEx**, the device point (0,0) is always the upper-left corner.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetViewportOrgEx](#)

[GetWindowOrgEx](#)

[POINT](#)

[SetViewportOrgEx](#)

SetWinMetaFileBits function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetWinMetaFileBits** function converts a metafile from the older Windows format to the new enhanced format and stores the new metafile in memory.

Syntax

```
HENHMETAFILE SetWinMetaFileBits(
    [in] UINT             nSize,
    [in] const BYTE       *lpMeta16Data,
    [in] HDC              hdcRef,
    [in] const METAFILEPICT *lpMFP
);
```

Parameters

[in] **nSize**

The size, in bytes, of the buffer that contains the Windows-format metafile.

[in] **lpMeta16Data**

A pointer to a buffer that contains the Windows-format metafile data. (It is assumed that the data was obtained by using the [GetMetaFileBitsEx](#) or [GetWinMetaFileBits](#) function.)

[in] **hdcRef**

A handle to a reference device context.

[in] **lpMFP**

A pointer to a [METAFILEPICT](#) structure that contains the suggested size of the metafile picture and the mapping mode that was used when the picture was created.

Return value

If the function succeeds, the return value is a handle to a memory-based enhanced metafile.

If the function fails, the return value is **NULL**.

Remarks

Windows uses the reference device context's resolution data and the data in the [METAFILEPICT](#) structure to scale a picture. If the *hdcRef* parameter is **NULL**, the system uses resolution data for the current output device. If the *lpMfp* parameter is **NULL**, the system uses the **MM_ANISOTROPIC** mapping mode to scale the picture so that it fits the entire device surface. The **hMF** member of the [METAFILEPICT](#) structure is not used.

When the application no longer needs the enhanced metafile handle, it should delete it by calling the [DeleteEnhMetaFile](#) function.

The handle returned by this function can be used with other enhanced-metafile functions.

If the reference device context is not identical to the device in which the metafile was originally created, some

GDI functions that use device units may not draw the picture correctly.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[DeleteEnhMetaFile](#)

[GetMetaFileBitsEx](#)

[GetWinMetaFileBits](#)

[METAFILEPICT](#)

[Metafile Functions](#)

[Metafiles Overview](#)

[PlayEnhMetaFile](#)

SetWorldTransform function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetWorldTransform** function sets a two-dimensional linear transformation between world space and page space for the specified device context. This transformation can be used to scale, rotate, shear, or translate graphics output.

Syntax

```
BOOL SetWorldTransform(
    [in] HDC      hdc,
    [in] const XFORM *lpxf
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpxf`

A pointer to an **XFORM** structure that contains the transformation data.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Below is the transformation matrix (note that the digits in the element notation are 1-based column number followed by 1-based row number, rather than the reverse).

eM11	eM21	eDx
eM12	eM22	eDy
0	0	1

So for any coordinates (x, y) in world space, the transformed coordinates in page space (x', y') can be determined in the way shown below.

x'		eM11	eM21	eDx		x				
y'		=	eM12	eM22	eDy		.		y	
1			0	0	1			1		

$$\begin{aligned}x' &= x * eM11 + y * eM21 + eDx \\y' &= x * eM12 + y * eM22 + eDy\end{aligned}$$

This function uses logical units.

The world transformation is usually used to scale or rotate logical images in a device-independent way.

The default world transformation is the identity matrix with zero offset.

The **SetWorldTransform** function will fail unless the graphics mode for the given device context has been set to GM_ADVANCED by previously calling the [SetGraphicsMode](#) function. Likewise, it will not be possible to reset the graphics mode for the device context to the default GM_COMPATIBLE mode, unless the world transformation has first been reset to the default identity transformation by calling [SetWorldTransform](#) or [ModifyWorldTransform](#).

Examples

For an example, see [Using Coordinate Spaces and Transformations](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetWorldTransform](#)

[ModifyWorldTransform](#)

[SetGraphicsMode](#)

[SetMapMode](#)

[SetViewportExtEx](#)

[SetViewportOrgEx](#)

[SetWindowExtEx](#)

[SetWindowOrgEx](#)

[XFORM](#)

StretchBlt function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **StretchBlt** function copies a bitmap from a source rectangle into a destination rectangle, stretching or compressing the bitmap to fit the dimensions of the destination rectangle, if necessary. The system stretches or compresses the bitmap according to the stretching mode currently set in the destination device context.

Syntax

```
BOOL StretchBlt(
    [in] HDC    hdcDest,
    [in] int     xDest,
    [in] int     yDest,
    [in] int     wDest,
    [in] int     hDest,
    [in] HDC    hdcSrc,
    [in] int     xSrc,
    [in] int     ySrc,
    [in] int     wSrc,
    [in] int     hSrc,
    [in] DWORD   rop
);
```

Parameters

[in] `hdcDest`

A handle to the destination device context.

[in] `xDest`

The x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] `yDest`

The y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] `wDest`

The width, in logical units, of the destination rectangle.

[in] `hDest`

The height, in logical units, of the destination rectangle.

[in] `hdcSrc`

A handle to the source device context.

[in] `xSrc`

The x-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] `ySrc`

The y-coordinate, in logical units, of the upper-left corner of the source rectangle.

[in] *wSrc*

The width, in logical units, of the source rectangle.

[in] *hSrc*

The height, in logical units, of the source rectangle.

[in] *rop*

The raster operation to be performed. Raster operation codes define how the system combines colors in output operations that involve a brush, a source bitmap, and a destination bitmap.

See [BitBlt](#) for a list of common raster operation codes (ROPs). Note that the CAPTUREBLT ROP generally cannot be used for printing device contexts.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

StretchBlt stretches or compresses the source bitmap in memory and then copies the result to the destination rectangle. This bitmap can be either a compatible bitmap (DDB) or the output from [CreateDIBSection](#). The color data for pattern or destination pixels is merged after the stretching or compression occurs.

When an enhanced metafile is being recorded, an error occurs (and the function returns FALSE) if the source device context identifies an enhanced-metafile device context.

If the specified raster operation requires a brush, the system uses the brush currently selected into the destination device context.

The destination coordinates are transformed by using the transformation currently specified for the destination device context; the source coordinates are transformed by using the transformation currently specified for the source device context.

If the source transformation has a rotation or shear, an error occurs.

If destination, source, and pattern bitmaps do not have the same color format, **StretchBlt** converts the source and pattern bitmaps to match the destination bitmap.

If **StretchBlt** must convert a monochrome bitmap to a color bitmap, it sets white bits (1) to the background color and black bits (0) to the foreground color. To convert a color bitmap to a monochrome bitmap, it sets pixels that match the background color to white (1) and sets all other pixels to black (0). The foreground and background colors of the device context with color are used.

StretchBlt creates a mirror image of a bitmap if the signs of the *nWidthSrc* and *nWidthDest* parameters or if the *nHeightSrc* and *nHeightDest* parameters differ. If *nWidthSrc* and *nWidthDest* have different signs, the function creates a mirror image of the bitmap along the x-axis. If *nHeightSrc* and *nHeightDest* have different signs, the function creates a mirror image of the bitmap along the y-axis.

Not all devices support the **StretchBlt** function. For more information, see the [GetDeviceCaps](#).

ICM: No color management is performed when a blit operation occurs.

When used in a multiple monitor system, both *hdcSrc* and *hdcDest* must refer to the same device or the function will fail. To transfer data between DCs for different devices, convert the memory bitmap to a DIB by calling [GetDIBits](#). To display the DIB to the second device, call [SetDIBits](#) or [StretchDIBits](#).

Examples

For an example, see [Scaling an Image](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BitBlt](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateDIBSection](#)

[GetDIBits](#)

[GetDeviceCaps](#)

[MaskBlt](#)

[PlgBlt](#)

[SetDIBits](#)

[SetStretchBltMode](#)

[StretchDIBits](#)

StretchDIBits function (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **StretchDIBits** function copies the color data for a rectangle of pixels in a DIB, JPEG, or PNG image to the specified destination rectangle. If the destination rectangle is larger than the source rectangle, this function stretches the rows and columns of color data to fit the destination rectangle. If the destination rectangle is smaller than the source rectangle, this function compresses the rows and columns by using the specified raster operation.

Syntax

```
int StretchDIBits(
    [in] HDC             hdc,
    [in] int              xDest,
    [in] int              yDest,
    [in] int              DestWidth,
    [in] int              DestHeight,
    [in] int              xSrc,
    [in] int              ySrc,
    [in] int              SrcWidth,
    [in] int              SrcHeight,
    [in] const VOID        *lpBits,
    [in] const BITMAPINFO *lpbmi,
    [in] UINT              iUsage,
    [in] DWORD             rop
);
```

Parameters

[in] **hdc**

A handle to the destination device context.

[in] **xDest**

The x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] **yDest**

The y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] **DestWidth**

The width, in logical units, of the destination rectangle.

[in] **DestHeight**

The height, in logical units, of the destination rectangle.

[in] **xSrc**

The x-coordinate, in pixels, of the source rectangle in the image.

[in] **ySrc**

The y-coordinate, in pixels, of the source rectangle in the image.

[in] *SrcWidth*

The width, in pixels, of the source rectangle in the image.

[in] *SrcHeight*

The height, in pixels, of the source rectangle in the image.

[in] *lpBits*

A pointer to the image bits, which are stored as an array of bytes. For more information, see the Remarks section.

[in] *lpbmi*

A pointer to a [BITMAPINFO](#) structure that contains information about the DIB.

[in] *iUsage*

Specifies whether the *bmiColors* member of the [BITMAPINFO](#) structure was provided and, if so, whether *bmiColors* contains explicit red, green, blue (RGB) values or indexes. The *iUsage* parameter must be one of the following values.

VALUE	MEANING
DIB_PAL_COLORS	The array contains 16-bit indexes into the logical palette of the source device context.
DIB_RGB_COLORS	The color table contains literal RGB values.

For more information, see the Remarks section.

[in] *rop*

A raster-operation code that specifies how the source pixels, the destination device context's current brush, and the destination pixels are to be combined to form the new image. For a list of some common raster operation codes, see [BitBlt](#).

Return value

If the function succeeds, the return value is the number of scan lines copied. Note that this value can be negative for mirrored content.

If the function fails, or no scan lines are copied, the return value is 0.

If the driver cannot support the JPEG or PNG file image passed to [StretchDIBits](#), the function will fail and return [GDI_ERROR](#). If failure does occur, the application must fall back on its own JPEG or PNG support to decompress the image into a bitmap, and then pass the bitmap to [StretchDIBits](#).

Remarks

The origin of a bottom-up DIB is the lower-left corner; the origin of a top-down DIB is the upper-left corner.

[StretchDIBits](#) creates a mirror image of a bitmap if the signs of the *nSrcWidth* and *nDestWidth* parameters, or if the *nSrcHeight* and *nDestHeight* parameters differ. If *nSrcWidth* and *nDestWidth* have different signs, the function creates a mirror image of the bitmap along the x-axis. If *nSrcHeight* and *nDestHeight* have different

signs, the function creates a mirror image of the bitmap along the y-axis.

StretchDIBits creates a top-down image if the sign of the **biHeight** member of the **BITMAPINFOHEADER** structure for the DIB is negative. For a code example, see [Sizing a JPEG or PNG Image](#).

This function allows a JPEG or PNG image to be passed as the source image. How each parameter is used remains the same, except:

- If the **biCompression** member of **BITMAPINFOHEADER** is BI_JPEG or BI_PNG, *lpBits* points to a buffer containing a JPEG or PNG image, respectively. The **biSizeImage** member of the **BITMAPINFOHEADER** structure specifies the size of the buffer. The *iUsage* parameter must be set to DIB_RGB_COLORS. The *dwRop* parameter must be set to SRCCOPY.
- To ensure proper metafile spooling while printing, applications must call the **CHECKJPEGFORMAT** or **CHECKPNGFORMAT** escape to verify that the printer recognizes the JPEG or PNG image, respectively, before calling **StretchDIBits**.

ICM: Color management is performed if color management has been enabled with a call to **SetICMMode** with the *iEnableICM* parameter set to ICM_ON. If the bitmap specified by *lpBitsInfo* has a **BITMAPV4HEADER** that specifies the gamma and endpoints members, or a **BITMAPV5HEADER** that specifies either the gamma and endpoints members or the profileData and profileSize members, then the call treats the bitmap's pixels as being expressed in the color space described by those members, rather than in the device context's source color space.

Examples

For an example, see [Sizing a JPEG or PNG Image](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BITMAPINFO](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[SetMapMode](#)

[SetStretchBltMode](#)

StrokeAndFillPath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **StrokeAndFillPath** function closes any open figures in a path, strokes the outline of the path by using the current pen, and fills its interior by using the current brush.

Syntax

```
BOOL StrokeAndFillPath(  
    [in] HDC hdc  
>;
```

Parameters

[in] *hdc*

A handle to the device context.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The device context identified by the *hdc* parameter must contain a closed path.

The **StrokeAndFillPath** function has the same effect as closing all the open figures in the path, and stroking and filling the path separately, except that the filled region will not overlap the stroked region even if the pen is wide.

Examples

For an example, see [Drawing a Pie Chart](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[FillPath](#)

[Path Functions](#)

[Paths Overview](#)

[SetPolyFillMode](#)

[StrokePath](#)

StrokePath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **StrokePath** function renders the specified path by using the current pen.

Syntax

```
BOOL StrokePath(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

Handle to a device context that contains the completed path.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The path, if it is to be drawn by **StrokePath**, must have been completed through a call to [EndPath](#). Calling this function on a path for which [EndPath](#) has not been called will cause this function to fail and return zero. Unlike other path drawing functions such as [StrokeAndFillPath](#), **StrokePath** will not attempt to close the path by drawing a straight line from the first point on the path to the last point on the path.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[EndPath](#)

[ExtCreatePen](#)

[Path Functions](#)

[Paths Overview](#)

TEXTMETRICA structure (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **TEXTMETRIC** structure contains basic information about a physical font. All sizes are specified in logical units; that is, they depend on the current mapping mode of the display context.

Syntax

```
typedef struct tagTEXTMETRICA {  
    LONG tmHeight;  
    LONG tmAscent;  
    LONG tmDescent;  
    LONG tmInternalLeading;  
    LONG tmExternalLeading;  
    LONG tmAveCharWidth;  
    LONG tmMaxCharWidth;  
    LONG tmWeight;  
    LONG tmOverhang;  
    LONG tmDigitizedAspectX;  
    LONG tmDigitizedAspectY;  
    BYTE tmFirstChar;  
    BYTE tmLastChar;  
    BYTE tmDefaultChar;  
    BYTE tmBreakChar;  
    BYTE tmItalic;  
    BYTE tmUnderlined;  
    BYTE tmStruckOut;  
    BYTE tmPitchAndFamily;  
    BYTE tmCharSet;  
} TEXTMETRICA, *PTEXTMETRICA, *NPTEXTMETRICA, *LPTEXTMETRICA;
```

Members

tmHeight

The height (ascent + descent) of characters.

tmAscent

The ascent (units above the base line) of characters.

tmDescent

The descent (units below the base line) of characters.

tmInternalLeading

The amount of leading (space) inside the bounds set by the **tmHeight** member. Accent marks and other diacritical characters may occur in this area. The designer may set this member to zero.

tmExternalLeading

The amount of extra leading (space) that the application adds between rows. Since this area is outside the font, it contains no marks and is not altered by text output calls in either OPAQUE or TRANSPARENT mode. The designer may set this member to zero.

tmAveCharWidth

The average width of characters in the font (generally defined as the width of the letter *x*). This value does not include the overhang required for bold or italic characters.

`tmMaxCharWidth`

The width of the widest character in the font.

`tmWeight`

The weight of the font.

`tmOverhang`

The extra width per string that may be added to some synthesized fonts. When synthesizing some attributes, such as bold or italic, graphics device interface (GDI) or a device may have to add width to a string on both a per-character and per-string basis. For example, GDI makes a string bold by expanding the spacing of each character and overstriking by an offset value; it italicizes a font by shearing the string. In either case, there is an overhang past the basic string. For bold strings, the overhang is the distance by which the overstrike is offset. For italic strings, the overhang is the amount the top of the font is sheared past the bottom of the font.

The **tmOverhang** member enables the application to determine how much of the character width returned by a [GetTextExtentPoint32](#) function call on a single character is the actual character width and how much is the per-string extra width. The actual width is the extent minus the overhang.

`tmDigitizedAspectX`

The horizontal aspect of the device for which the font was designed.

`tmDigitizedAspectY`

The vertical aspect of the device for which the font was designed. The ratio of the **tmDigitizedAspectX** and **tmDigitizedAspectY** members is the aspect ratio of the device for which the font was designed.

`tmFirstChar`

The value of the first character defined in the font.

`tmLastChar`

The value of the last character defined in the font.

`tmDefaultChar`

The value of the character to be substituted for characters not in the font.

`tmBreakChar`

The value of the character that will be used to define word breaks for text justification.

`tmItalic`

Specifies an italic font if it is nonzero.

`tmUnderlined`

Specifies an underlined font if it is nonzero.

`tmStruckOut`

A strikeout font if it is nonzero.

`tmPitchAndFamily`

Specifies information about the pitch, the technology, and the family of a physical font.

The four low-order bits of this member specify information about the pitch and the technology of the font. A constant is defined for each of the four bits.

CONSTANT	MEANING
TMPF_FIXED_PITCH	If this bit is set the font is a variable pitch font. If this bit is clear the font is a fixed pitch font. Note very carefully that those meanings are the opposite of what the constant name implies.
TMPF_VECTOR	If this bit is set the font is a vector font.
TMPF_TRUETYPE	If this bit is set the font is a TrueType font.
TMPF_DEVICE	If this bit is set the font is a device font.

An application should carefully test for qualities encoded in these low-order bits, making no arbitrary assumptions. For example, besides having their own bits set, TrueType and PostScript fonts set the TMPF_VECTOR bit. A monospace bitmap font has all of these low-order bits clear; a proportional bitmap font sets the TMPF_FIXED_PITCH bit. A Postscript printer device font sets the TMPF_DEVICE, TMPF_VECTOR, and TMPF_FIXED_PITCH bits.

The four high-order bits of **tmPitchAndFamily** designate the font's font family. An application can use the value 0xF0 and the bitwise AND operator to mask out the four low-order bits of **tmPitchAndFamily**, thus obtaining a value that can be directly compared with font family names to find an identical match. For information about font families, see the description of the [LOGFONT](#) structure.

tmCharSet

The character set of the font. The character set can be one of the following values.

- ANSI_CHARSET
- BALTIC_CHARSET
- CHINESEBIG5_CHARSET
- DEFAULT_CHARSET
- EASTEUROPE_CHARSET
- GB2312_CHARSET
- GREEK_CHARSET
- HANGUL_CHARSET
- MAC_CHARSET
- OEM_CHARSET
- RUSSIAN_CHARSET
- SHIFTJIS_CHARSET
- SYMBOL_CHARSET
- TURKISH_CHARSET
- VIETNAMESE_CHARSET

Korean language edition of Windows:

- JOHAB_CHARSET

Middle East language edition of Windows:

- ARABIC_CHARSET
- HEBREW_CHARSET

Thai language edition of Windows:

- THAI_CHARSET

Remarks

NOTE

The wingdi.h header defines TEXTMETRIC as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

[LOGFONT](#)

TEXTMETRICW structure (wingdi.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **TEXTMETRIC** structure contains basic information about a physical font. All sizes are specified in logical units; that is, they depend on the current mapping mode of the display context.

Syntax

```
typedef struct tagTEXTMETRICW {  
    LONG tmHeight;  
    LONG tmAscent;  
    LONG tmDescent;  
    LONG tmInternalLeading;  
    LONG tmExternalLeading;  
    LONG tmAveCharWidth;  
    LONG tmMaxCharWidth;  
    LONG tmWeight;  
    LONG tmOverhang;  
    LONG tmDigitizedAspectX;  
    LONG tmDigitizedAspectY;  
    WCHAR tmFirstChar;  
    WCHAR tmLastChar;  
    WCHAR tmDefaultChar;  
    WCHAR tmBreakChar;  
    BYTE tmItalic;  
    BYTE tmUnderlined;  
    BYTE tmStruckOut;  
    BYTE tmPitchAndFamily;  
    BYTE tmCharSet;  
} TEXTMETRICW, *PTEXTMETRICW, *NPTEXTMETRICW, *LPTEXTMETRICW;
```

Members

tmHeight

The height (ascent + descent) of characters.

tmAscent

The ascent (units above the base line) of characters.

tmDescent

The descent (units below the base line) of characters.

tmInternalLeading

The amount of leading (space) inside the bounds set by the **tmHeight** member. Accent marks and other diacritical characters may occur in this area. The designer may set this member to zero.

tmExternalLeading

The amount of extra leading (space) that the application adds between rows. Since this area is outside the font, it contains no marks and is not altered by text output calls in either OPAQUE or TRANSPARENT mode. The designer may set this member to zero.

tmAveCharWidth

The average width of characters in the font (generally defined as the width of the letter *x*). This value does not include the overhang required for bold or italic characters.

`tmMaxCharWidth`

The width of the widest character in the font.

`tmWeight`

The weight of the font.

`tmOverhang`

The extra width per string that may be added to some synthesized fonts. When synthesizing some attributes, such as bold or italic, graphics device interface (GDI) or a device may have to add width to a string on both a per-character and per-string basis. For example, GDI makes a string bold by expanding the spacing of each character and overstriking by an offset value; it italicizes a font by shearing the string. In either case, there is an overhang past the basic string. For bold strings, the overhang is the distance by which the overstrike is offset. For italic strings, the overhang is the amount the top of the font is sheared past the bottom of the font.

The **tmOverhang** member enables the application to determine how much of the character width returned by a [GetTextExtentPoint32](#) function call on a single character is the actual character width and how much is the per-string extra width. The actual width is the extent minus the overhang.

`tmDigitizedAspectX`

The horizontal aspect of the device for which the font was designed.

`tmDigitizedAspectY`

The vertical aspect of the device for which the font was designed. The ratio of the **tmDigitizedAspectX** and **tmDigitizedAspectY** members is the aspect ratio of the device for which the font was designed.

`tmFirstChar`

The value of the first character defined in the font.

`tmLastChar`

The value of the last character defined in the font.

`tmDefaultChar`

The value of the character to be substituted for characters not in the font.

`tmBreakChar`

The value of the character that will be used to define word breaks for text justification.

`tmItalic`

Specifies an italic font if it is nonzero.

`tmUnderlined`

Specifies an underlined font if it is nonzero.

`tmStruckOut`

A strikeout font if it is nonzero.

`tmPitchAndFamily`

Specifies information about the pitch, the technology, and the family of a physical font.

The four low-order bits of this member specify information about the pitch and the technology of the font. A constant is defined for each of the four bits.

CONSTANT	MEANING
TMPF_FIXED_PITCH	If this bit is set the font is a variable pitch font. If this bit is clear the font is a fixed pitch font. Note very carefully that those meanings are the opposite of what the constant name implies.
TMPF_VECTOR	If this bit is set the font is a vector font.
TMPF_TRUETYPE	If this bit is set the font is a TrueType font.
TMPF_DEVICE	If this bit is set the font is a device font.

An application should carefully test for qualities encoded in these low-order bits, making no arbitrary assumptions. For example, besides having their own bits set, TrueType and PostScript fonts set the TMPF_VECTOR bit. A monospace bitmap font has all of these low-order bits clear; a proportional bitmap font sets the TMPF_FIXED_PITCH bit. A Postscript printer device font sets the TMPF_DEVICE, TMPF_VECTOR, and TMPF_FIXED_PITCH bits.

The four high-order bits of **tmPitchAndFamily** designate the font's font family. An application can use the value 0xF0 and the bitwise AND operator to mask out the four low-order bits of **tmPitchAndFamily**, thus obtaining a value that can be directly compared with font family names to find an identical match. For information about font families, see the description of the [LOGFONT](#) structure.

tmCharSet

The character set of the font. The character set can be one of the following values.

- ANSI_CHARSET
- BALTIC_CHARSET
- CHINESEBIG5_CHARSET
- DEFAULT_CHARSET
- EASTEUROPE_CHARSET
- GB2312_CHARSET
- GREEK_CHARSET
- HANGUL_CHARSET
- MAC_CHARSET
- OEM_CHARSET
- RUSSIAN_CHARSET
- SHIFTJIS_CHARSET
- SYMBOL_CHARSET
- TURKISH_CHARSET
- VIETNAMESE_CHARSET

Korean language edition of Windows:

- JOHAB_CHARSET

Middle East language edition of Windows:

- ARABIC_CHARSET
- HEBREW_CHARSET

Thai language edition of Windows:

- THAI_CHARSET

Remarks

NOTE

The wingdi.h header defines TEXTMETRIC as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

[LOGFONT](#)

TextOutA function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **TextOut** function writes a character string at the specified location, using the currently selected font, background color, and text color.

Syntax

```
BOOL TextOutA(
    [in] HDC     hdc,
    [in] int     x,
    [in] int     y,
    [in] LPCSTR  lpString,
    [in] int     c
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The x-coordinate, in logical coordinates, of the reference point that the system uses to align the string.

[in] `y`

The y-coordinate, in logical coordinates, of the reference point that the system uses to align the string.

[in] `lpString`

A pointer to the string to be drawn. The string does not need to be zero-terminated, because `cchString` specifies the length of the string.

[in] `c`

The [length of the string](#) pointed to by `lpString`, in characters.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The interpretation of the reference point depends on the current text-alignment mode. An application can retrieve this mode by calling the [GetTextAlign](#) function; an application can alter this mode by calling the [SetTextAlign](#) function. You can use the following values for text alignment. Only one flag can be chosen from those that affect horizontal and vertical alignment. In addition, only one of the two flags that alter the current position can be chosen.

TERM	DESCRIPTION
TA_BASELINE	The reference point will be on the base line of the text.
TA_BOTTOM	The reference point will be on the bottom edge of the bounding rectangle.
TA_TOP	The reference point will be on the top edge of the bounding rectangle.
TA_CENTER	The reference point will be aligned horizontally with the center of the bounding rectangle.
TA_LEFT	The reference point will be on the left edge of the bounding rectangle.
TA_RIGHT	The reference point will be on the right edge of the bounding rectangle.
TA_NOUPDATECP	The current position is not updated after each text output call. The reference point is passed to the text output function.
TA_RTLREADING	Middle East language edition of Windows: The text is laid out in right to left reading order, as opposed to the default left to right order. This applies only when the font selected into the device context is either Hebrew or Arabic.
TA_UPDATECP	The current position is updated after each text output call. The current position is used as the reference point.

By default, the current position is not used or updated by this function. However, an application can call the [SetTextAlign](#) function with the *fMode* parameter set to TA_UPDATECP to permit the system to use and update the current position each time the application calls **TextOut** for a specified device context. When this flag is set, the system ignores the *nXStart* and *nYStart* parameters on subsequent **TextOut** calls.

When the **TextOut** function is placed inside a path bracket, the system generates a path for the TrueType text that includes each character plus its character box. The region generated is the character box minus the text, rather than the text itself. You can obtain the region enclosed by the outline of the TrueType text by setting the background mode to transparent before placing the **TextOut** function in the path bracket. Following is sample code that demonstrates this procedure.

```

// Obtain the window's client rectangle
GetClientRect(hwnd, &r);

// THE FIX: by setting the background mode
// to transparent, the region is the text itself
// SetBkMode(hdc, TRANSPARENT);

// Bracket begin a path
BeginPath(hdc);

// Send some text out into the world
TCHAR text[ ] = "Defenestration can be hazardous";
TextOut(hdc,r.left,r.top,text, ARRSIZE(text));

// Bracket end a path
EndPath(hdc);

// Derive a region from that path
SelectClipPath(hdc, RGN_AND);

// This generates the same result as SelectClipPath()
// SelectClipRgn(hdc, PathToRegion(hdc));

// Fill the region with grayness
FillRect(hdc, &r, GetStockObject(GRAY_BRUSH));

```

Examples

For an example, see [Enumerating the Installed Fonts](#).

NOTE

The wingdi.h header defines TextOut as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

Fonts and Text Overview

[GetTextAlign](#)

[SelectObject](#)

[SetBkColor](#)

[SetTextAlign](#)

[SetTextColor](#)

[TabbedTextOut](#)

TextOutW function (wingdi.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **TextOut** function writes a character string at the specified location, using the currently selected font, background color, and text color.

Syntax

```
BOOL TextOutW(
    [in] HDC      hdc,
    [in] int      x,
    [in] int      y,
    [in] LPCWSTR lpString,
    [in] int      c
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The x-coordinate, in logical coordinates, of the reference point that the system uses to align the string.

[in] `y`

The y-coordinate, in logical coordinates, of the reference point that the system uses to align the string.

[in] `lpString`

A pointer to the string to be drawn. The string does not need to be zero-terminated, because `cchString` specifies the length of the string.

[in] `c`

The [length of the string](#) pointed to by `lpString`, in characters.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The interpretation of the reference point depends on the current text-alignment mode. An application can retrieve this mode by calling the [GetTextAlign](#) function; an application can alter this mode by calling the [SetTextAlign](#) function. You can use the following values for text alignment. Only one flag can be chosen from those that affect horizontal and vertical alignment. In addition, only one of the two flags that alter the current position can be chosen.

TERM	DESCRIPTION
TA_BASELINE	The reference point will be on the base line of the text.
TA_BOTTOM	The reference point will be on the bottom edge of the bounding rectangle.
TA_TOP	The reference point will be on the top edge of the bounding rectangle.
TA_CENTER	The reference point will be aligned horizontally with the center of the bounding rectangle.
TA_LEFT	The reference point will be on the left edge of the bounding rectangle.
TA_RIGHT	The reference point will be on the right edge of the bounding rectangle.
TA_NOUPDATECP	The current position is not updated after each text output call. The reference point is passed to the text output function.
TA_RTLREADING	Middle East language edition of Windows: The text is laid out in right to left reading order, as opposed to the default left to right order. This applies only when the font selected into the device context is either Hebrew or Arabic.
TA_UPDATECP	The current position is updated after each text output call. The current position is used as the reference point.

By default, the current position is not used or updated by this function. However, an application can call the [SetTextAlign](#) function with the *fMode* parameter set to TA_UPDATECP to permit the system to use and update the current position each time the application calls **TextOut** for a specified device context. When this flag is set, the system ignores the *nXStart* and *nYStart* parameters on subsequent **TextOut** calls.

When the **TextOut** function is placed inside a path bracket, the system generates a path for the TrueType text that includes each character plus its character box. The region generated is the character box minus the text, rather than the text itself. You can obtain the region enclosed by the outline of the TrueType text by setting the background mode to transparent before placing the **TextOut** function in the path bracket. Following is sample code that demonstrates this procedure.

```

// Obtain the window's client rectangle
GetClientRect(hwnd, &r);

// THE FIX: by setting the background mode
// to transparent, the region is the text itself
// SetBkMode(hdc, TRANSPARENT);

// Bracket begin a path
BeginPath(hdc);

// Send some text out into the world
TCHAR text[ ] = "Defenestration can be hazardous";
TextOut(hdc,r.left,r.top,text, ARRSIZE(text));

// Bracket end a path
EndPath(hdc);

// Derive a region from that path
SelectClipPath(hdc, RGN_AND);

// This generates the same result as SelectClipPath()
// SelectClipRgn(hdc, PathToRegion(hdc));

// Fill the region with grayness
FillRect(hdc, &r, GetStockObject(GRAY_BRUSH));

```

Examples

For an example, see [Enumerating the Installed Fonts](#).

NOTE

The wingdi.h header defines TextOut as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

Fonts and Text Overview

[GetTextAlign](#)

[SelectObject](#)

[SetBkColor](#)

[SetTextAlign](#)

[SetTextColor](#)

[TabbedTextOut](#)

TransparentBlt function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **TransparentBlt** function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

Syntax

```
BOOL TransparentBlt(
    [in] HDC hdcDest,
    [in] int xoriginDest,
    [in] int yoriginDest,
    [in] int wDest,
    [in] int hDest,
    [in] HDC hdcSrc,
    [in] int xoriginSrc,
    [in] int yoriginSrc,
    [in] int wSrc,
    [in] int hSrc,
    [in] UINT crTransparent
);
```

Parameters

[in] hdcDest

A handle to the destination device context.

[in] xoriginDest

The x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] yoriginDest

The y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

[in] wDest

The width, in logical units, of the destination rectangle.

[in] hDest

The height, in logical units, of the destination rectangle.

[in] hdcSrc

A handle to the source device context.

[in] xoriginSrc

The x-coordinate, in logical units, of the source rectangle.

[in] yoriginSrc

The y-coordinate, in logical units, of the source rectangle.

[in] wSrc

The width, in logical units, of the source rectangle.

[in] hSrc

The height, in logical units, of the source rectangle.

[in] crTransparent

The RGB color in the source bitmap to treat as transparent.

Return value

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**.

Remarks

The **TransparentBlt** function works with compatible bitmaps (DDBs).

The **TransparentBlt** function supports all formats of source bitmaps. However, for 32 bpp bitmaps, it just copies the alpha value over. Use [AlphaBlend](#) to specify 32 bits-per-pixel bitmaps with transparency.

If the source and destination rectangles are not the same size, the source bitmap is stretched to match the destination rectangle. When the [SetStretchBltMode](#) function is used, the *iStretchMode* modes of **BLACKONWHITE** and **WHITEONBLACK** are converted to **COLORONCOLOR** for the **TransparentBlt** function.

The destination device context specifies the transformation type for the destination coordinates. The source device context specifies the transformation type for the source coordinates.

TransparentBlt does not mirror a bitmap if either the width or height, of either the source or destination, is negative.

When used in a multiple monitor system, both *hdcSrc* and *hdcDest* must refer to the same device or the function will fail. To transfer data between DCs for different devices, convert the memory bitmap to a DIB by calling [GetDIBits](#). To display the DIB to the second device, call [SetDIBits](#) or [StretchDIBits](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Msimg32.lib
DLL	Msimg32.dll

See also

[AlphaBlend](#)

[Bitmap Functions](#)

[Bitmaps Overview](#)

[GetDIBits](#)

[SetDIBits](#)

[SetStretchBltMode](#)

[StretchDIBits](#)

TRIVERTEX structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **TRIVERTEX** structure contains color information and position information.

Syntax

```
typedef struct _TRIVERTEX {  
    LONG    x;  
    LONG    y;  
    COLOR16 Red;  
    COLOR16 Green;  
    COLOR16 Blue;  
    COLOR16 Alpha;  
} TRIVERTEX, *PTRIVERTEX, *LPTRIVERTEX;
```

Members

x

The x-coordinate, in logical units, of the upper-left corner of the rectangle.

y

The y-coordinate, in logical units, of the upper-left corner of the rectangle.

Red

The color information at the point of x, y.

Green

The color information at the point of x, y.

Blue

The color information at the point of x, y.

Alpha

The color information at the point of x, y.

Remarks

In the **TRIVERTEX** structure, x and y indicate position in the same manner as in the **POINTL** structure contained in the wtypes.h header file. Red, Green, Blue, and Alpha members indicate color information at the point x, y. The color information of each channel is specified as a value from 0x0000 to 0xff00. This allows higher color resolution for an object that has been split into small triangles for display. The **TRIVERTEX** structure contains information needed by the *pVertex* parameter of **GradientFill**.

Examples

For an example of the use of this structure, see [Drawing a Shaded Triangle](#) or [Drawing a Shaded Rectangle](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Bitmap Structures](#)

[Bitmaps Overview](#)

[GradientFill](#)

[POINTL](#)

TTPOLYCURVE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The TTPOLYCURVE structure contains information about a curve in the outline of a TrueType character.

Syntax

```
typedef struct tagTTPOLYCURVE {  
    WORD    wType;  
    WORD    cpfx;  
    POINTFX apfx[1];  
} TTPOLYCURVE, *LPTTPOLYCURVE;
```

Members

wType

The type of curve described by the structure. This member can be one of the following values.

VALUE	MEANING
TT_PRIM_LINE	Curve is a polyline.
TT_PRIM_QSPLINE	Curve is a quadratic Bézier spline.
TT_PRIM_CSPLINE	Curve is a cubic Bézier spline.

cpfx

The number of [POINTFX](#) structures in the array.

apfx

Specifies an array of [POINTFX](#) structures that define the polyline or Bézier spline.

Remarks

When an application calls the [GetGlyphOutline](#) function, a glyph outline for a TrueType character is returned in a [TTPOLYGONHEADER](#) structure, followed by as many TTPOLYCURVE structures as are required to describe the glyph. All points are returned as [POINTFX](#) structures and represent absolute positions, not relative moves. The starting point specified by the [pfxStart](#) member of the [TTPOLYGONHEADER](#) structure is the point at which the outline for a contour begins. The TTPOLYCURVE structures that follow can be either polyline records or spline records.

Polyline records are a series of points; lines drawn between the points describe the outline of the character. Spline records represent the quadratic curves (that is, quadratic b-splines) used by TrueType.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetGlyphOutline](#)

[POINTFX](#)

[TTPOLYGONHEADER](#)

TTPOLYGONHEADER structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **TTPOLYGONHEADER** structure specifies the starting position and type of a contour in a TrueType character outline.

Syntax

```
typedef struct tagTTPOLYGONHEADER {  
    DWORD    cb;  
    DWORD    dwType;  
    POINTFX pfxStart;  
} TTPOLYGONHEADER, *LPTTPOLYGONHEADER;
```

Members

cb

The number of bytes required by the **TTPOLYGONHEADER** structure and **TTPOLYCURVE** structure or structures required to describe the contour of the character.

dwType

The type of character outline returned. Currently, this value must be **TT_POLYGON_TYPE**.

pfxStart

The starting point of the contour in the character outline.

Remarks

Each **TTPOLYGONHEADER** structure is followed by one or more **TTPOLYCURVE** structures.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[POINTFX](#)

[TTPOLYCURVE](#)

UnrealizeObject function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **UnrealizeObject** function resets the origin of a brush or resets a logical palette. If the *hgdiobj* parameter is a handle to a brush, **UnrealizeObject** directs the system to reset the origin of the brush the next time it is selected. If the *hgdiobj* parameter is a handle to a logical palette, **UnrealizeObject** directs the system to realize the palette as though it had not previously been realized. The next time the application calls the [RealizePalette](#) function for the specified palette, the system completely remaps the logical palette to the system palette.

Syntax

```
BOOL UnrealizeObject(  
    [in] HGDIOBJ h  
);
```

Parameters

[in] *h*

A handle to the logical palette to be reset.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **UnrealizeObject** function should not be used with stock objects. For example, the default palette, obtained by calling [GetStockObject](#) (DEFAULT_PALETTE), is a stock object.

A palette identified by *hgdiobj* can be the currently selected palette of a device context.

If *hgdiobj* is a brush, **UnrealizeObject** does nothing, and the function returns TRUE. Use [SetBrushOrgEx](#) to set the origin of a brush.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

DLL	Gdi32.dll
-----	-----------

See also

[Color Functions](#)

[Colors Overview](#)

[GetStockObject](#)

[RealizePalette](#)

[SetBrushOrgEx](#)

UpdateColors function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **UpdateColors** function updates the client area of the specified device context by remapping the current colors in the client area to the currently realized logical palette.

Syntax

```
BOOL UpdateColors(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

A handle to the device context.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

An application can determine whether a device supports palette operations by calling the [GetDeviceCaps](#) function and specifying the RASTERCAPS constant.

An inactive window with a realized logical palette may call **UpdateColors** as an alternative to redrawing its client area when the system palette changes.

The **UpdateColors** function typically updates a client area faster than redrawing the area. However, because **UpdateColors** performs the color translation based on the color of each pixel before the system palette changed, each call to this function results in the loss of some color accuracy.

This function must be called soon after a [WM_PALETTECHANGED](#) message is received.

Requirements

Windows compatibility	
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)

Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Color Functions](#)

[Colors Overview](#)

[GetDeviceCaps](#)

[RealizePalette](#)

WCRANGE structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **WCRANGE** structure specifies a range of Unicode characters.

Syntax

```
typedef struct tagWCRANGE {  
    WCHAR  wcLow;  
    USHORT cGlyphs;  
} WCRANGE, *PWCRANGE, *LPWCRANGE;
```

Members

`wcLow`

Low Unicode code point in the range of supported Unicode code points.

`cGlyphs`

Number of supported Unicode code points in this range.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GLYPHSET](#)

WidenPath function (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **WidenPath** function redefines the current path as the area that would be painted if the path were stroked using the pen currently selected into the given device context.

Syntax

```
BOOL WidenPath(  
    [in] HDC hdc  
>;
```

Parameters

[in] *hdc*

A handle to a device context that contains a closed path.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **WidenPath** function is successful only if the current pen is a geometric pen created by the [ExtCreatePen](#) function, or if the pen is created with the [CreatePen](#) function and has a width, in device units, of more than one.

The device context identified by the *hdc* parameter must contain a closed path.

Any Bézier curves in the path are converted to sequences of straight lines approximating the widened curves. As such, no Bézier curves remain in the path after **WidenPath** is called.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[CreatePen](#)

[EndPath](#)

[ExtCreatePen](#)

[Path Functions](#)

[Paths Overview](#)

[SetMiterLimit](#)

XFORM structure (wingdi.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The XFORM structure specifies a world-space to page-space transformation.

Syntax

```
typedef struct tagXFORM {  
    FLOAT eM11;  
    FLOAT eM12;  
    FLOAT eM21;  
    FLOAT eM22;  
    FLOAT eDx;  
    FLOAT eDy;  
} XFORM, *PXFORM, *LPXFORM;
```

Members

eM11

The following.

OPERATION	MEANING
Scaling	Horizontal scaling component
Rotation	Cosine of rotation angle
Reflection	Horizontal component

eM12

The following.

OPERATION	MEANING
Shear	Horizontal proportionality constant
Rotation	Sine of the rotation angle

eM21

The following.

OPERATION	MEANING
Shear	Vertical proportionality constant
Rotation	Negative sine of the rotation angle

eM22

The following.

OPERATION	MEANING
Scaling	Vertical scaling component
Rotation	Cosine of rotation angle
Reflection	Vertical reflection component

eDx

The horizontal translation component, in logical units.

eDy

The vertical translation component, in logical units.

Remarks

The following list describes how the members are used for each operation.

OPERATION	EM11	EM12	EM21	EM22
Rotation	Cosine	Sine	Negative sine	Cosine
Scaling	Horizontal scaling component	Not used	Not used	Vertical Scaling Component
Shear	Not used	Horizontal Proportionality Constant	Vertical Proportionality Constant	Not used
Reflection	Horizontal Reflection Component	Not used	Not used	Vertical Reflection Component

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Coordinate Space and Transformation Structures](#)

[Coordinate Spaces and Transformations Overview](#)

[ExtCreateRegion](#)

[GetWorldTransform](#)

[ModifyWorldTransform](#)

[PlayEnhMetaFile](#)

[SetWorldTransform](#)

winuser.h header

4/21/2022 • 79 minutes to read • [Edit Online](#)

This header is used by multiple technologies. For more information, see:

- [Data Exchange](#)
- [Desktop Window Manager \(DWM\)](#)
- [Developer Notes](#)
- [Dialog Boxes](#)
- [Display Devices Reference](#)
- [High DPI](#)
- [Input Feedback Configuration](#)
- [Input Source Identification](#)
- [Internationalization for Windows Applications](#)
- [Keyboard and Mouse Input](#)
- [Menus and Other Resources](#)
- [Mobile Device Management Settings Provider](#)
- [Pointer Device Input Stack](#)
- [Pointer Input Messages and Notifications](#)
- [Remote Desktop Services](#)
- [Security and Identity](#)
- [System Services](#)
- [The Windows Shell](#)
- [Touch Hit Testing](#)
- [Touch Injection](#)
- [Touch Input](#)
- [Window Stations and Desktops](#)
- [Windows Accessibility Features](#)
- [Windows and Messages](#)
- [Windows Controls](#)
- [Windows GDI](#)

winuser.h contains the following programming interfaces:

Functions

[ActivateKeyboardLayout](#)

Sets the input locale identifier (formerly called the keyboard layout handle) for the calling thread or the current process. The input locale identifier specifies a locale as well as the physical layout of the keyboard.

[AddClipboardFormatListener](#)

Places the given window in the system-maintained clipboard format listener list.

[AdjustWindowRect](#)

Calculates the required size of the window rectangle, based on the desired client-rectangle size. The window rectangle can then be passed to the CreateWindow function to create a window whose client area is the desired size.

[AdjustWindowRectEx](#)

Calculates the required size of the window rectangle, based on the desired size of the client rectangle. The window rectangle can then be passed to the CreateWindowEx function to create a window whose client area is the desired size.

[AdjustWindowRectExForDpi](#)

Calculates the required size of the window rectangle, based on the desired size of the client rectangle and the provided DPI.

[AllowSetForegroundWindow](#)

Enables the specified process to set the foreground window using the SetForegroundWindow function. The calling process must already be able to set the foreground window. For more information, see Remarks later in this topic.

[AnimateWindow](#)

Enables you to produce special effects when showing or hiding windows. There are four types of animation:_roll, slide, collapse or expand, and alpha-blended fade.

[AnyPopup](#)

Indicates whether an owned, visible, top-level pop-up, or overlapped window exists on the screen. The function searches the entire screen, not just the calling application's client area.

[AppendMenuA](#)

Appends a new item to the end of the specified menu bar, drop-down menu, submenu, or shortcut menu. You can use this function to specify the content, appearance, and behavior of the menu item.

[AppendMenuW](#)

Appends a new item to the end of the specified menu bar, drop-down menu, submenu, or shortcut menu. You can use this function to specify the content, appearance, and behavior of the menu item.

[AreDpiAwarenessContextsEqual](#)

Determines whether two DPI_AWARENESS_CONTEXT values are identical.

[ArrangeIconicWindows](#)

Arranges all the minimized (iconic) child windows of the specified parent window.

[AttachThreadInput](#)

Attaches or detaches the input processing mechanism of one thread to that of another thread.

[BeginDeferWindowPos](#)

Allocates memory for a multiple-window- position structure and returns the handle to the structure.

[BeginPaint](#)

The BeginPaint function prepares the specified window for painting and fills a PAINTSTRUCT structure with information about the painting.

[BlockInput](#)

Blocks keyboard and mouse input events from reaching applications.

[BringWindowToTop](#)

Brings the specified window to the top of the Z order. If the window is a top-level window, it is activated. If the window is a child window, the top-level parent window associated with the child window is activated.

[BroadcastSystemMessage](#)

Sends a message to the specified recipients.

[BroadcastSystemMessageA](#)

Sends a message to the specified recipients.

[BroadcastSystemMessageExA](#)

Sends a message to the specified recipients.

[BroadcastSystemMessageExW](#)

Sends a message to the specified recipients.

[BroadcastSystemMessageW](#)

Sends a message to the specified recipients.

[CalculatePopupWindowPosition](#)

Calculates an appropriate pop-up window position using the specified anchor point, pop-up window size, flags, and the optional exclude rectangle.

[CallMsgFilterA](#)

Passes the specified message and hook code to the hook procedures associated with the WH_SYSMSGFILTER and WH_MSGFILTER hooks.

[CallMsgFilterW](#)

Passes the specified message and hook code to the hook procedures associated with the WH_SYSMSGFILTER and WH_MSGFILTER hooks.

[CallNextHookEx](#)

Passes the hook information to the next hook procedure in the current hook chain. A hook procedure can call this function either before or after processing the hook information.

[CallWindowProcA](#)

Passes message information to the specified window procedure.

[CallWindowProcW](#)

Passes message information to the specified window procedure.

[CascadeWindows](#)

Cascades the specified child windows of the specified parent window.

[ChangeClipboardChain](#)

Removes a specified window from the chain of clipboard viewers.

[ChangeDisplaySettingsA](#)

The ChangeDisplaySettings function changes the settings of the default display device to the specified graphics mode.

[ChangeDisplaySettingsExA](#)

The ChangeDisplaySettingsEx function changes the settings of the specified display device to the specified graphics mode.

[ChangeDisplaySettingsExW](#)

The ChangeDisplaySettingsEx function changes the settings of the specified display device to the specified graphics mode.

[ChangeDisplaySettingsW](#)

The ChangeDisplaySettings function changes the settings of the default display device to the specified graphics mode.

[ChangeWindowMessageFilter](#)

Adds or removes a message from the User Interface Privilege Isolation (UIPI) message filter.

[ChangeWindowMessageFilterEx](#)

Modifies the User Interface Privilege Isolation (UIPI) message filter for a specified window.

[CharLowerA](#)

Converts a character string or a single character to lowercase. If the operand is a character string, the function converts the characters in place.

[CharLowerBuffA](#)

Converts uppercase characters in a buffer to lowercase characters. The function converts the characters in place.

[CharLowerBuffW](#)

Converts uppercase characters in a buffer to lowercase characters. The function converts the characters in place.

[CharLowerW](#)

Converts a character string or a single character to lowercase. If the operand is a character string, the function converts the characters in place.

[CharNextA](#)

Retrieves a pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.

[CharNextExA](#)

Retrieves the pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.

[CharNextW](#)

Retrieves a pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.

[CharPrevA](#)

Retrieves a pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.

[CharPrevExA](#)

Retrieves the pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.

[CharPrevW](#)

Retrieves a pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.

[CharToOemA](#)

Translates a string into the OEM-defined character set. **Warning** Do not use.

[CharToOemBuffA](#)

Translates a specified number of characters in a string into the OEM-defined character set.

[CharToOemBuffW](#)

Translates a specified number of characters in a string into the OEM-defined character set.

[CharToOemW](#)

Translates a string into the OEM-defined character set. **Warning** Do not use.

[CharUpperA](#)

Converts a character string or a single character to uppercase. If the operand is a character string, the function converts the characters in place.

[CharUpperBuffA](#)

Converts lowercase characters in a buffer to uppercase characters. The function converts the characters in place.

[CharUpperBuffW](#)

Converts lowercase characters in a buffer to uppercase characters. The function converts the characters in place.

[CharUpperW](#)

Converts a character string or a single character to uppercase. If the operand is a character string, the function converts the characters in place.

[CheckDlgButton](#)

Changes the check state of a button control.

[CheckMenuItem](#)

Sets the state of the specified menu item's check-mark attribute to either selected or clear.

[CheckMenuItem](#)

Checks a specified menu item and makes it a radio item. At the same time, the function clears all other menu items in the associated group and clears the radio-item type flag for those items.

[CheckRadioButton](#)

Adds a check mark to (checks) a specified radio button in a group and removes a check mark from (clears) all other radio buttons in the group.

[ChildWindowFromPoint](#)

Determines which, if any, of the child windows belonging to a parent window contains the specified point. The search is restricted to immediate child windows. Grandchildren, and deeper descendant windows are not searched.

[ChildWindowFromPointEx](#)

Determines which, if any, of the child windows belonging to the specified parent window contains the specified point.

[ClientToScreen](#)

The ClientToScreen function converts the client-area coordinates of a specified point to screen coordinates.

[ClipCursor](#)

Confines the cursor to a rectangular area on the screen.

[CloseClipboard](#)

Closes the clipboard.

[CloseDesktop](#)

Closes an open handle to a desktop object.

[CloseGestureInfoHandle](#)

Closes resources associated with a gesture information handle.

[CloseTouchInputHandle](#)

Closes a touch input handle, frees process memory associated with it, and invalidates the handle.

[CloseWindow](#)

Minimizes (but does not destroy) the specified window.

[CloseWindowStation](#)

Closes an open window station handle.

[CopyAcceleratorTableA](#)

Copies the specified accelerator table. This function is used to obtain the accelerator-table data that corresponds to an accelerator-table handle, or to determine the size of the accelerator-table data.

[CopyAcceleratorTableW](#)

Copies the specified accelerator table. This function is used to obtain the accelerator-table data that corresponds to an accelerator-table handle, or to determine the size of the accelerator-table data.

[CopyCursor](#)

Copies the specified cursor.

[CopyIcon](#)

Copies the specified icon from another module to the current module.

[CopyImage](#)

Creates a new image (icon, cursor, or bitmap) and copies the attributes of the specified image to the new one. If necessary, the function stretches the bits to fit the desired size of the new image.

[CopyRect](#)

The CopyRect function copies the coordinates of one rectangle to another.

[CountClipboardFormats](#)

Retrieves the number of different data formats currently on the clipboard.

[CreateAcceleratorTableA](#)

Creates an accelerator table.

[CreateAcceleratorTableW](#)

Creates an accelerator table.

[CreateCaret](#)

Creates a new shape for the system caret and assigns ownership of the caret to the specified window. The caret shape can be a line, a block, or a bitmap.

[CreateCursor](#)

Creates a cursor having the specified size, bit patterns, and hot spot.

[CreateDesktopA](#)

Creates a new desktop, associates it with the current window station of the calling process, and assigns it to the calling thread.

[CreateDesktopExA](#)

Creates a new desktop with the specified heap, associates it with the current window station of the calling process, and assigns it to the calling thread.

[CreateDesktopExW](#)

Creates a new desktop with the specified heap, associates it with the current window station of the calling process, and assigns it to the calling thread.

[CreateDesktopW](#)

Creates a new desktop, associates it with the current window station of the calling process, and assigns it to the calling thread.

[CreateDialogA](#)

Creates a modeless dialog box from a dialog box template resource. The CreateDialog macro uses the CreateDialogParam function.

[CreateDialogIndirectA](#)

Creates a modeless dialog box from a dialog box template in memory. The CreateDialogIndirect macro uses the CreateDialogIndirectParam function.

[CreateDialogIndirectParamA](#)

Creates a modeless dialog box from a dialog box template in memory.

[CreateDialogIndirectParamW](#)

Creates a modeless dialog box from a dialog box template in memory.

[CreateDialogIndirectW](#)

Creates a modeless dialog box from a dialog box template in memory. The CreateDialogIndirect macro uses the CreateDialogIndirectParam function.

[CreateDialogParamA](#)

Creates a modeless dialog box from a dialog box template resource.

[CreateDialogParamW](#)

Creates a modeless dialog box from a dialog box template resource.

[CreateDialogW](#)

Creates a modeless dialog box from a dialog box template resource. The CreateDialog macro uses the CreateDialogParam function.

[CreateIcon](#)

Creates an icon that has the specified size, colors, and bit patterns.

[CreateIconFromResource](#)

Creates an icon or cursor from resource bits describing the icon.

[CreateIconFromResourceEx](#)

Creates an icon or cursor from resource bits describing the icon.

[CreateIconIndirect](#)

Creates an icon or cursor from an ICONINFO structure.

[CreateMDIWindowA](#)

Creates a multiple-document interface (MDI) child window.

[CreateMDIWindowW](#)

Creates a multiple-document interface (MDI) child window.

[CreateMenu](#)

Creates a menu. The menu is initially empty, but it can be filled with menu items by using the InsertMenuItem, AppendMenu, and InsertMenu functions.

[CreatePopupMenu](#)

Creates a drop-down menu, submenu, or shortcut menu.

[CreateSyntheticPointerDevice](#)

Configures the pointer injection device for the calling application, and initializes the maximum number of simultaneous pointers that the app can inject.

[CreateWindowA](#)

Creates an overlapped, pop-up, or child window.

[CreateWindowExA](#)

Creates an overlapped, pop-up, or child window with an extended window style; otherwise, this function is identical to the CreateWindow function.

[CreateWindowExW](#)

Creates an overlapped, pop-up, or child window with an extended window style; otherwise, this function is identical to the CreateWindow function.

[CreateWindowStationA](#)

Creates a window station object, associates it with the calling process, and assigns it to the current session.

[CreateWindowStationW](#)

Creates a window station object, associates it with the calling process, and assigns it to the current session.

[CreateWindowW](#)

Creates an overlapped, pop-up, or child window.

[DefDlgProcA](#)

Calls the default dialog box window procedure to provide default processing for any window messages that a dialog box with a private window class does not process.

[DefDlgProcW](#)

Calls the default dialog box window procedure to provide default processing for any window messages that a dialog box with a private window class does not process.

[DeferWindowPos](#)

Updates the specified multiple-window  position structure for the specified window.

[DefFrameProcA](#)

Provides default processing for any window messages that the window procedure of a multiple-document interface (MDI) frame window does not process.

[DefFrameProcW](#)

Provides default processing for any window messages that the window procedure of a multiple-document interface (MDI) frame window does not process.

[DefMDIChildProcA](#)

Provides default processing for any window message that the window procedure of a multiple-document interface (MDI) child window does not process.

[DefMDIChildProcW](#)

Provides default processing for any window message that the window procedure of a multiple-document interface (MDI) child window does not process.

[DefRawInputProc](#)

Verifies that the size of the RAWINPUTHEADER structure is correct.

[DefWindowProcA](#)

Calls the default window procedure to provide default processing for any window messages that an application does not process.

[DefWindowProcW](#)

Calls the default window procedure to provide default processing for any window messages that an application does not process.

[DeleteMenu](#)

Deletes an item from the specified menu. If the menu item opens a menu or submenu, this function destroys the handle to the menu or submenu and frees the memory used by the menu or submenu.

[DeregisterShellHookWindow](#)

Unregisters a specified Shell window that is registered to receive Shell hook messages.

[DestroyAcceleratorTable](#)

Destroys an accelerator table.

[DestroyCaret](#)

Destroys the caret's current shape, frees the caret from the window, and removes the caret from the screen.

[DestroyCursor](#)

Destroys a cursor and frees any memory the cursor occupied. Do not use this function to destroy a shared cursor.

[DestroyIcon](#)

Destroys an icon and frees any memory the icon occupied.

[DestroyMenu](#)

Destroys the specified menu and frees any memory that the menu occupies.

[DestroySyntheticPointerDevice](#)

Destroys the specified pointer injection device.

[DestroyWindow](#)

Destroys the specified window.

[DialogBoxA](#)

Creates a modal dialog box from a dialog box template resource. DialogBox does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.

[DialogBoxIndirectA](#)

Creates a modal dialog box from a dialog box template in memory. DialogBoxIndirect does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.

[DialogBoxIndirectParamA](#)

Creates a modal dialog box from a dialog box template in memory.

[DialogBoxIndirectParamW](#)

Creates a modal dialog box from a dialog box template in memory.

[DialogBoxIndirectW](#)

Creates a modal dialog box from a dialog box template in memory. DialogBoxIndirect does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.

[DialogBoxParamA](#)

Creates a modal dialog box from a dialog box template resource.

[DialogBoxParamW](#)

Creates a modal dialog box from a dialog box template resource.

[DialogBoxW](#)

Creates a modal dialog box from a dialog box template resource. DialogBox does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.

[DisableProcessWindowsGhosting](#)

Disables the window ghosting feature for the calling GUI process. Window ghosting is a Windows Manager feature that lets the user minimize, move, or close the main window of an application that is not responding.

[DispatchMessage](#)

Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the GetMessage function.

[DispatchMessageA](#)

Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the GetMessage function.

[DispatchMessageW](#)

Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the GetMessage function.

[DisplayConfigGetDeviceInfo](#)

The DisplayConfigGetDeviceInfo function retrieves display configuration information about the device.

[DisplayConfigSetDeviceInfo](#)

The DisplayConfigSetDeviceInfo function sets the properties of a target.

[DlgDirListA](#)

Replaces the contents of a list box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list can optionally include mapped drives.

DlgDirListComboBoxA

Replaces the contents of a combo box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list of names can include mapped drive letters.

DlgDirListComboBoxW

Replaces the contents of a combo box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list of names can include mapped drive letters.

DlgDirListW

Replaces the contents of a list box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list can optionally include mapped drives.

DlgDirSelectComboBoxExA

Retrieves the current selection from a combo box filled by using the DlgDirListComboBox function. The selection is interpreted as a drive letter, a file, or a directory name.

DlgDirSelectComboBoxExW

Retrieves the current selection from a combo box filled by using the DlgDirListComboBox function. The selection is interpreted as a drive letter, a file, or a directory name.

DlgDirSelectExA

Retrieves the current selection from a single-selection list box. It assumes that the list box has been filled by the DlgDirList function and that the selection is a drive letter, filename, or directory name.

DlgDirSelectExW

Retrieves the current selection from a single-selection list box. It assumes that the list box has been filled by the DlgDirList function and that the selection is a drive letter, filename, or directory name.

DragDetect

Captures the mouse and tracks its movement until the user releases the left button, presses the ESC key, or moves the mouse outside the drag rectangle around the specified point.

DrawAnimatedRects

Animates the caption of a window to indicate the opening of an icon or the minimizing or maximizing of a window.

DrawCaption

The DrawCaption function draws a window caption.

DrawEdge

The DrawEdge function draws one or more edges of rectangle.

DrawFocusRect

The DrawFocusRect function draws a rectangle in the style used to indicate that the rectangle has the focus.

[DrawFrameControl](#)

The `DrawFrameControl` function draws a frame control of the specified type and style.

[DrawIcon](#)

Draws an icon or cursor into the specified device context.

[DrawIconEx](#)

Draws an icon or cursor into the specified device context, performing the specified raster operations, and stretching or compressing the icon or cursor as specified.

[DrawMenuBar](#)

Redraws the menu bar of the specified window. If the menu bar changes after the system has created the window, this function must be called to draw the changed menu bar.

[DrawStateA](#)

The `DrawState` function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.

[DrawStateW](#)

The `DrawState` function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.

[DrawText](#)

The `DrawText` function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

[DrawTextA](#)

The `DrawText` function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

[DrawTextExA](#)

The `DrawTextEx` function draws formatted text in the specified rectangle.

[DrawTextExW](#)

The `DrawTextEx` function draws formatted text in the specified rectangle.

[DrawTextW](#)

The `DrawText` function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

[EmptyClipboard](#)

Empties the clipboard and frees handles to data in the clipboard. The function then assigns ownership of the clipboard to the window that currently has the clipboard open.

[EnableMenuItem](#)

Enables, disables, or grays the specified menu item.

[EnableMouseInPointer](#)

Enables the mouse to act as a pointer input device and send WM_POINTER messages.

[EnableNonClientDpiScaling](#)

In high-DPI displays, enables automatic display scaling of the non-client area portions of the specified top-level window. Must be called during the initialization of that window.

[EnableScrollBar](#)

The EnableScrollBar function enables or disables one or both scroll bar arrows.

[EnableWindow](#)

Enables or disables mouse and keyboard input to the specified window or control. When input is disabled, the window does not receive input such as mouse clicks and key presses. When input is enabled, the window receives all input.

[EndDeferWindowPos](#)

Simultaneously updates the position and size of one or more windows in a single screen-refreshing cycle.

[EndDialog](#)

Destroys a modal dialog box, causing the system to end any processing for the dialog box.

[EndMenu](#)

Ends the calling thread's active menu.

[EndPaint](#)

The EndPaint function marks the end of painting in the specified window. This function is required for each call to the BeginPaint function, but only after painting is complete.

[EndTask](#)

Forcibly closes the specified window.

[EnumChildWindows](#)

Enumerates the child windows that belong to the specified parent window by passing the handle to each child window, in turn, to an application-defined callback function.

[EnumClipboardFormats](#)

Enumerates the data formats currently available on the clipboard.

[EnumDesktopsA](#)

Enumerates all desktops associated with the specified window station of the calling process. The function passes the name of each desktop, in turn, to an application-defined callback function.

[EnumDesktopsW](#)

Enumerates all desktops associated with the specified window station of the calling process. The function passes the name of each desktop, in turn, to an application-defined callback function.

[EnumDesktopWindows](#)

Enumerates all top-level windows associated with the specified desktop. It passes the handle to each window, in turn, to an application-defined callback function.

[EnumDisplayDevicesA](#)

The `EnumDisplayDevices` function lets you obtain information about the display devices in the current session.

[EnumDisplayDevicesW](#)

The `EnumDisplayDevices` function lets you obtain information about the display devices in the current session.

[EnumDisplayMonitors](#)

The `EnumDisplayMonitors` function enumerates display monitors (including invisible pseudo-monitors associated with the mirroring drivers) that intersect a region formed by the intersection of a specified clipping rectangle and the visible region of a device context. `EnumDisplayMonitors` calls an application-defined `MonitorEnumProc` callback function once for each monitor that is enumerated. Note that `GetSystemMetrics` (`SM_CMONITORS`) counts only the display monitors.

[EnumDisplaySettingsA](#)

The `EnumDisplaySettings` function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.

[EnumDisplaySettingsExA](#)

The `EnumDisplaySettingsEx` function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.

[EnumDisplaySettingsExW](#)

The `EnumDisplaySettingsEx` function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.

[EnumDisplaySettingsW](#)

The `EnumDisplaySettings` function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.

[EnumPropsA](#)

Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. `EnumProps` continues until the last entry is enumerated or the callback function returns FALSE.

[EnumPropsExA](#)

Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. `EnumPropsEx` continues until the last entry is enumerated or the callback function returns FALSE.

[EnumPropsExW](#)

Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumPropsEx continues until the last entry is enumerated or the callback function returns FALSE.

[EnumPropsW](#)

Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumProps continues until the last entry is enumerated or the callback function returns FALSE.

[EnumThreadWindows](#)

Enumerates all nonchild windows associated with a thread by passing the handle to each window, in turn, to an application-defined callback function.

[EnumWindows](#)

Enumerates all top-level windows on the screen by passing the handle to each window, in turn, to an application-defined callback function. EnumWindows continues until the last top-level window is enumerated or the callback function returns FALSE.

[EnumWindowStationsA](#)

Enumerates all window stations in the current session. The function passes the name of each window station, in turn, to an application-defined callback function.

[EnumWindowStationsW](#)

Enumerates all window stations in the current session. The function passes the name of each window station, in turn, to an application-defined callback function.

[EqualRect](#)

The EqualRect function determines whether the two specified rectangles are equal by comparing the coordinates of their upper-left and lower-right corners.

[EvaluateProximityToPolygon](#)

Returns the score of a polygon as the probable touch target (compared to all other polygons that intersect the touch contact area) and an adjusted touch point within the polygon.

[EvaluateProximityToRect](#)

Returns the score of a rectangle as the probable touch target, compared to all other rectangles that intersect the touch contact area, and an adjusted touch point within the rectangle.

[ExcludeUpdateRgn](#)

The ExcludeUpdateRgn function prevents drawing within invalid areas of a window by excluding an updated region in the window from a clipping region.

[ExitWindows](#)

Calls the ExitWindowsEx function to log off the interactive user.

[ExitWindowsEx](#)

Logs off the interactive user, shuts down the system, or shuts down and restarts the system.

[FillRect](#)

The FillRect function fills a rectangle by using the specified brush. This function includes the left and top borders, but excludes the right and bottom borders of the rectangle.

[FindWindowA](#)

Retrieves a handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows. This function does not perform a case-sensitive search.

[FindWindowExA](#)

Retrieves a handle to a window whose class name and window name match the specified strings. The function searches child windows, beginning with the one following the specified child window. This function does not perform a case-sensitive search.

[FindWindowExW](#)

Retrieves a handle to a window whose class name and window name match the specified strings. The function searches child windows, beginning with the one following the specified child window. This function does not perform a case-sensitive search.

[FindWindowW](#)

Retrieves a handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows. This function does not perform a case-sensitive search.

[FlashWindow](#)

Flashes the specified window one time. It does not change the active state of the window.

[FlashWindowEx](#)

Flashes the specified window. It does not change the active state of the window.

[FrameRect](#)

The FrameRect function draws a border around the specified rectangle by using the specified brush. The width and height of the border are always one logical unit.

[GET_APPCOMMAND_LPARAM](#)

Retrieves the application command from the specified LPARAM value.

[GET_DEVICE_LPARAM](#)

Retrieves the input device type from the specified LPARAM value.

[GET_FLAGS_LPARAM](#)

Retrieves the state of certain virtual keys from the specified LPARAM value.

[GET_KEYSTATE_LPARAM](#)

Retrieves the state of certain virtual keys from the specified LPARAM value.

[GET_KEYSTATE_WPARAM](#)

Retrieves the state of certain virtual keys from the specified WPARAM value.

[GET_NCHITTEST_WPARAM](#)

Retrieves the hit-test value from the specified WPARAM value.

[GET_POINTERID_WPARAM](#)

Retrieves the pointer ID using the specified value.

[GET_RAWINPUT_CODE_WPARAM](#)

Retrieves the input code from wParam in WM_INPUT.

[GET_WHEEL_DELTA_WPARAM](#)

Retrieves the wheel-delta value from the specified WPARAM value.

[GET_XBUTTON_WPARAM](#)

Retrieves the state of certain buttons from the specified WPARAM value.

[GetActiveWindow](#)

Retrieves the window handle to the active window attached to the calling thread's message queue.

[GetAltTabInfoA](#)

Retrieves status information for the specified window if it is the application-switching (ALT+TAB) window.

[GetAltTabInfoW](#)

Retrieves status information for the specified window if it is the application-switching (ALT+TAB) window.

[GetAncestor](#)

Retrieves the handle to the ancestor of the specified window.

[GetAsyncKeyState](#)

Determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to GetAsyncKeyState.

[GetAutoRotationState](#)

Retrieves an AR_STATE value containing the state of screen auto-rotation for the system, for example whether auto-rotation is supported, and whether it is enabled by the user.

[GetAwarenessFromDpiAwarenessContext](#)

Retrieves the DPI_AWARENESS value from a DPI_AWARENESS_CONTEXT.

[GetCapture](#)

Retrieves a handle to the window (if any) that has captured the mouse. Only one window at a time can capture the mouse; this window receives mouse input whether or not the cursor is within its borders.

[GetCaretBlinkTime](#)

Retrieves the time required to invert the caret's pixels. The user can set this value.

[GetCaretPos](#)

Copies the caret's position to the specified POINT structure.

[GetCIMSSM](#)

Retrieves the source of the input message (GetCurrentInputMessageSourceInSendMessage).

[GetClassInfoA](#)

Retrieves information about a window class.

[GetClassInfoExA](#)

Retrieves information about a window class, including a handle to the small icon associated with the window class. The GetClassInfo function does not retrieve a handle to the small icon.

[GetClassInfoExW](#)

Retrieves information about a window class, including a handle to the small icon associated with the window class. The GetClassInfo function does not retrieve a handle to the small icon.

[GetClassInfoW](#)

Retrieves information about a window class.

[GetClassLongA](#)

Retrieves the specified 32-bit (DWORD) value from the WNDCLASSEX structure associated with the specified window.

[GetClassLongPtrA](#)

Retrieves the specified value from the WNDCLASSEX structure associated with the specified window.

[GetClassLongPtrW](#)

Retrieves the specified value from the WNDCLASSEX structure associated with the specified window.

[GetClassLongW](#)

Retrieves the specified 32-bit (DWORD) value from the WNDCLASSEX structure associated with the specified window.

[GetClassName](#)

Retrieves the name of the class to which the specified window belongs.

[GetClassNameA](#)

Retrieves the name of the class to which the specified window belongs.

[GetClassNameW](#)

Retrieves the name of the class to which the specified window belongs.

[GetClassWord](#)

Retrieves the 16-bit (WORD) value at the specified offset into the extra class memory for the window class to which the specified window belongs.

[GetClientRect](#)

Retrieves the coordinates of a window's client area.

[GetClipboardData](#)

Retrieves data from the clipboard in a specified format. The clipboard must have been opened previously.

[GetClipboardFormatNameA](#)

Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.

[GetClipboardFormatNameW](#)

Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.

[GetClipboardOwner](#)

Retrieves the window handle of the current owner of the clipboard.

[GetClipboardSequenceNumber](#)

Retrieves the clipboard sequence number for the current window station.

[GetClipboardViewer](#)

Retrieves the handle to the first window in the clipboard viewer chain.

[GetClipCursor](#)

Retrieves the screen coordinates of the rectangular area to which the cursor is confined.

[GetComboBoxInfo](#)

Retrieves information about the specified combo box.

[GetCurrentInputMessageSource](#)

Retrieves the source of the input message.

[GetCursor](#)

Retrieves a handle to the current cursor.

[GetCursorInfo](#)

Retrieves information about the global cursor.

[GetCursorPos](#)

Retrieves the position of the mouse cursor, in screen coordinates.

[GetDC](#)

The GetDC function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen.

[GetDCEx](#)

The GetDCEx function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen.

[GetDesktopWindow](#)

Retrieves a handle to the desktop window. The desktop window covers the entire screen. The desktop window is the area on top of which other windows are painted.

[GetDialogBaseUnits](#)

Retrieves the system's dialog base units, which are the average width and height of characters in the system font.

[GetDialogControlDpiChangeBehavior](#)

Retrieves and per-monitor DPI scaling behavior overrides of a child window in a dialog.

[GetDialogDpiChangeBehavior](#)

Returns the flags that might have been set on a given dialog by an earlier call to SetDialogDpiChangeBehavior.

[GetDisplayAutoRotationPreferences](#)

Retrieves the screen auto-rotation preferences for the current process.

[GetDisplayAutoRotationPreferencesByProcessId](#)

Retrieves the screen auto-rotation preferences for the process indicated by the dwProcessId parameter.

[GetDisplayConfigBufferSizes](#)

The GetDisplayConfigBufferSizes function retrieves the size of the buffers that are required to call the QueryDisplayConfig function.

[GetDlgCtrlID](#)

Retrieves the identifier of the specified control.

[GetDlgItem](#)

Retrieves a handle to a control in the specified dialog box.

[GetDlgItemInt](#)

Translates the text of a specified control in a dialog box into an integer value.

[GetDlgItemTextA](#)

Retrieves the title or text associated with a control in a dialog box.

[GetDlgItemTextW](#)

Retrieves the title or text associated with a control in a dialog box.

[GetDoubleClickTime](#)

Retrieves the current double-click time for the mouse.

[GetDpiForSystem](#)

Returns the system DPI.

[GetDpiForWindow](#)

Returns the dots per inch (dpi) value for the associated window.

[GetDpiFromDpiAwarenessContext](#)

Retrieves the DPI from a given DPI_AWARENESS_CONTEXT handle. This enables you to determine the DPI of a thread without needed to examine a window created within that thread.

[GetFocus](#)

Retrieves the handle to the window that has the keyboard focus, if the window is attached to the calling thread's message queue.

[GetForegroundWindow](#)

Retrieves a handle to the foreground window (the window with which the user is currently working). The system assigns a slightly higher priority to the thread that creates the foreground window than it does to other threads.

[GetGestureConfig](#)

Retrieves the configuration for which Windows Touch gesture messages are sent from a window.

[GetGestureExtraArgs](#)

Retrieves additional information about a gesture from its GESTUREINFO handle.

[GetGestureInfo](#)

Retrieves a GESTUREINFO structure given a handle to the gesture information.

[GetGuiResources](#)

Retrieves the count of handles to graphical user interface (GUI) objects in use by the specified process.

[GetGUIThreadInfo](#)

Retrieves information about the active window or a specified GUI thread.

[GetIconInfo](#)

Retrieves information about the specified icon or cursor.

[GetIconInfoExA](#)

Retrieves information about the specified icon or cursor. GetIconInfoEx extends GetIconInfo by using the newer ICONINFOEX structure.

[GetIconInfoExW](#)

Retrieves information about the specified icon or cursor. GetIconInfoEx extends GetIconInfo by using the newer ICONINFOEX structure.

[GetInputState](#)

Determines whether there are mouse-button or keyboard messages in the calling thread's message queue.

[GetKBCodePage](#)

Retrieves the current code page.

[GetKeyboardLayout](#)

Retrieves the active input locale identifier (formerly called the keyboard layout).

[GetKeyboardLayoutList](#)

Retrieves the input locale identifiers (formerly called keyboard layout handles) corresponding to the current set of input locales in the system. The function copies the identifiers to the specified buffer.

[GetKeyboardLayoutNameA](#)

Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.

[GetKeyboardLayoutNameW](#)

Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.

[GetKeyboardState](#)

Copies the status of the 256 virtual keys to the specified buffer.

[GetKeyboardType](#)

Retrieves information about the current keyboard.

[GetKeyNameTextA](#)

Retrieves a string that represents the name of a key.

[GetKeyNameTextW](#)

Retrieves a string that represents the name of a key.

[GetKeyState](#)

Retrieves the status of the specified virtual key. The status specifies whether the key is up, down, or toggled (on, off, alternating each time the key is pressed).

[GetLastActivePopup](#)

Determines which pop-up window owned by the specified window was most recently active.

[GetLastInputInfo](#)

Retrieves the time of the last input event.

[GetLayeredWindowAttributes](#)

Retrieves the opacity and transparency color key of a layered window.

[GetListBoxInfo](#)

Retrieves the number of items per column in a specified list box.

[GetMenu](#)

Retrieves a handle to the menu assigned to the specified window.

[GetMenuBarInfo](#)

Retrieves information about the specified menu bar.

[GetMenuCheckMarkDimensions](#)

Retrieves the dimensions of the default check-mark bitmap.

[GetMenuItemContextHelpId](#)

Retrieves the Help context identifier associated with the specified menu.

[GetMenuItemDefault](#)

Determines the default menu item on the specified menu.

[GetMenuItemInfo](#)

Retrieves information about a specified menu.

[GetMenuItemCount](#)

Determines the number of items in the specified menu.

[GetMenuItemID](#)

Retrieves the menu item identifier of a menu item located at the specified position in a menu.

[GetMenuItemInfoA](#)

Retrieves information about a menu item.

[GetMenuItemInfoW](#)

Retrieves information about a menu item.

[GetMenuItemRect](#)

Retrieves the bounding rectangle for the specified menu item.

[GetMenuState](#)

Retrieves the menu flags associated with the specified menu item.

[GetMenuItemStringA](#)

Copies the text string of the specified menu item into the specified buffer.

[GetMenuItemStringW](#)

Copies the text string of the specified menu item into the specified buffer.

[GetMessage](#)

Retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.

[GetMessageA](#)

Retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.

[GetMessageExtraInfo](#)

Retrieves the extra message information for the current thread. Extra message information is an application- or driver-defined value associated with the current thread's message queue.

[GetMessagePos](#)

Retrieves the cursor position for the last message retrieved by the GetMessage function.

[GetMessageTime](#)

Retrieves the message time for the last message retrieved by the GetMessage function.

[GetMessageW](#)

Retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.

[GetMonitorInfoA](#)

The GetMonitorInfo function retrieves information about a display monitor.

[GetMonitorInfoW](#)

The GetMonitorInfo function retrieves information about a display monitor.

[GetMouseMovePointsEx](#)

Retrieves a history of up to 64 previous coordinates of the mouse or pen.

[GetNextDlgGroupItem](#)

Retrieves a handle to the first control in a group of controls that precedes (or follows) the specified control in a dialog box.

[GetNextDlgTabItem](#)

Retrieves a handle to the first control that has the WS_TABSTOP style that precedes (or follows) the specified control.

[GetNextWindow](#)

Retrieves a handle to the next or previous window in the Z-Order. The next window is below the specified window; the previous window is above.

[GetOpenClipboardWindow](#)

Retrieves the handle to the window that currently has the clipboard open.

[GetParent](#)

Retrieves a handle to the specified window's parent or owner.

[GetPhysicalCursorPos](#)

Retrieves the position of the cursor in physical coordinates.

[GetPointerCursorId](#)

Retrieves the cursor identifier associated with the specified pointer.

[GetPointerDevice](#)

Gets information about the pointer device.

[GetPointerDeviceCursors](#)

Gets the cursor IDs that are mapped to the cursors associated with a pointer device.

[GetPointerDeviceProperties](#)

Gets device properties that aren't included in the `POINTER_DEVICE_INFO` structure.

[GetPointerDeviceRects](#)

Gets the x and y range for the pointer device (in himetric) and the x and y range (current resolution) for the display that the pointer device is mapped to.

[GetPointerDevices](#)

Gets information about the pointer devices attached to the system.

[GetPointerFrameInfo](#)

Gets the entire frame of information for the specified pointers associated with the current message.

[GetPointerFrameInfoHistory](#)

Gets the entire frame of information (including coalesced input frames) for the specified pointers associated with the current message.

[GetPointerFramePenInfo](#)

Gets the entire frame of pen-based information for the specified pointers (of type `PT_PEN`) associated with the current message.

[GetPointerFramePenInfoHistory](#)

Gets the entire frame of pen-based information (including coalesced input frames) for the specified pointers (of type `PT_PEN`) associated with the current message.

[GetPointerFrameTouchInfo](#)

Gets the entire frame of touch-based information for the specified pointers (of type `PT_TOUCH`) associated with the current message.

[GetPointerFrameTouchInfoHistory](#)

Gets the entire frame of touch-based information (including coalesced input frames) for the specified pointers (of type `PT_TOUCH`) associated with the current message.

[GetPointerInfo](#)

Gets the information for the specified pointer associated with the current message.

[GetPointerInfoHistory](#)

Gets the information associated with the individual inputs, if any, that were coalesced into the current message for the specified pointer.

GetPointerInputTransform	Gets one or more transforms for the pointer information coordinates associated with the current message.
GetPointerPenInfo	Gets the pen-based information for the specified pointer (of type PT_PEN) associated with the current message.
GetPointerPenInfoHistory	Gets the pen-based information associated with the individual inputs, if any, that were coalesced into the current message for the specified pointer (of type PT_PEN).
GetPointerTouchInfo	Gets the touch-based information for the specified pointer (of type PT_TOUCH) associated with the current message.
GetPointerTouchInfoHistory	Gets the touch-based information associated with the individual inputs, if any, that were coalesced into the current message for the specified pointer (of type PT_TOUCH).
GetPointerType	Retrieves the pointer type for a specified pointer.
GetPriorityClipboardFormat	Retrieves the first available clipboard format in the specified list.
GetProcessDefaultLayout	Retrieves the default layout that is used when windows are created with no parent or owner.
GetProcessWindowStation	Retrieves a handle to the current window station for the calling process.
GetPropA	Retrieves a data handle from the property list of the specified window. The character string identifies the handle to be retrieved. The string and handle must have been added to the property list by a previous call to the SetProp function.
GetPropW	Retrieves a data handle from the property list of the specified window. The character string identifies the handle to be retrieved. The string and handle must have been added to the property list by a previous call to the SetProp function.
GetQueueStatus	Retrieves the type of messages found in the calling thread's message queue.
GetRawInputBuffer	Performs a buffered read of the raw input data.

[GetRawInputData](#)

Retrieves the raw input from the specified device.

[GetRawInputDeviceInfoA](#)

Retrieves information about the raw input device.

[GetRawInputDeviceInfoW](#)

Retrieves information about the raw input device.

[GetRawInputDeviceList](#)

Enumerates the raw input devices attached to the system.

[GetRawPointerDeviceData](#)

Gets the raw input data from the pointer device.

[GetRegisteredRawInputDevices](#)

Retrieves the information about the raw input devices for the current application.

[GetScrollBarInfo](#)

The GetScrollBarInfo function retrieves information about the specified scroll bar.

[GetScrollInfo](#)

The GetScrollInfo function retrieves the parameters of a scroll bar, including the minimum and maximum scrolling positions, the page size, and the position of the scroll box (thumb).

[GetScrollPos](#)

The GetScrollPos function retrieves the current position of the scroll box (thumb) in the specified scroll bar.

[GetScrollRange](#)

The GetScrollRange function retrieves the current minimum and maximum scroll box (thumb) positions for the specified scroll bar.

[GetShellWindow](#)

Retrieves a handle to the Shell's desktop window.

[GetSubMenu](#)

Retrieves a handle to the drop-down menu or submenu activated by the specified menu item.

[GetSysColor](#)

Retrieves the current color of the specified display element.

[GetSysColorBrush](#)

The GetSysColorBrush function retrieves a handle identifying a logical brush that corresponds to the specified color index.

[GetSystemDpiForProcess](#)

Retrieves the system DPI associated with a given process. This is useful for avoiding compatibility issues that arise from sharing DPI-sensitive information between multiple system-aware processes with different system DPI values.

[GetSystemMenu](#)

Enables the application to access the window menu (also known as the system menu or the control menu) for copying and modifying.

[GetSystemMetrics](#)

Retrieves the specified system metric or system configuration setting.

[GetSystemMetricsForDpi](#)

Retrieves the specified system metric or system configuration setting taking into account a provided DPI.

[GetTabbedTextExtentA](#)

The GetTabbedTextExtent function computes the width and height of a character string.

[GetTabbedTextExtentW](#)

The GetTabbedTextExtent function computes the width and height of a character string.

[GetThreadDesktop](#)

Retrieves a handle to the desktop assigned to the specified thread.

[GetThreadDpiAwarenessContext](#)

Gets the DPI_AWARENESS_CONTEXT for the current thread.

[GetThreadDpiHostingBehavior](#)

Retrieves the DPI_HOSTING_BEHAVIOR from the current thread.

[GetTitleBarInfo](#)

Retrieves information about the specified title bar.

[GetTopWindow](#)

Examines the Z order of the child windows associated with the specified parent window and retrieves a handle to the child window at the top of the Z order.

[GetTouchInputInfo](#)

Retrieves detailed information about touch inputs associated with a particular touch input handle.

[GetUnpredictedMessagePos](#)

Gets pointer data before it has gone through touch prediction processing.

[GetUpdatedClipboardFormats](#)

Retrieves the currently supported clipboard formats.

[GetUpdateRect](#)

The GetUpdateRect function retrieves the coordinates of the smallest rectangle that completely encloses the update region of the specified window.

[GetUpdateRgn](#)

The GetUpdateRgn function retrieves the update region of a window by copying it into the specified region. The coordinates of the update region are relative to the upper-left corner of the window (that is, they are client coordinates).

 [GetUserObjectInformationA](#)

Retrieves information about the specified window station or desktop object.

 [GetUserObjectInformationW](#)

Retrieves information about the specified window station or desktop object.

 [GetUserObjectSecurity](#)

Retrieves security information for the specified user object.

[GetWindow](#)

Retrieves a handle to a window that has the specified relationship (Z-Order or owner) to the specified window.

[GetWindowContextHelpId](#)

Retrieves the Help context identifier, if any, associated with the specified window.

[GetWindowDC](#)

The GetWindowDC function retrieves the device context (DC) for the entire window, including title bar, menus, and scroll bars.

[GetWindowDisplayAffinity](#)

Retrieves the current display affinity setting, from any process, for a given window.

[GetWindowDpiAwarenessContext](#)

Returns the DPI_AWARENESS_CONTEXT associated with a window.

[GetWindowDpiHostingBehavior](#)

Returns the DPI_HOSTING_BEHAVIOR of the specified window.

[GetWindowFeedbackSetting](#)

Retrieves the feedback configuration for a window.

[GetWindowInfo](#)

Retrieves information about the specified window.

[GetWindowLongA](#)

Retrieves information about the specified window.

[GetWindowLongPtrA](#)

Retrieves information about the specified window. The function also retrieves the value at a specified offset into the extra window memory.

[GetWindowLongPtrW](#)

Retrieves information about the specified window. The function also retrieves the value at a specified offset into the extra window memory.

[GetWindowLongW](#)

Retrieves information about the specified window.

[GetWindowModuleFileNameA](#)

Retrieves the full path and file name of the module associated with the specified window handle.

[GetWindowModuleFileNameW](#)

Retrieves the full path and file name of the module associated with the specified window handle.

[GetWindowPlacement](#)

Retrieves the show state and the restored, minimized, and maximized positions of the specified window.

[GetWindowRect](#)

Retrieves the dimensions of the bounding rectangle of the specified window. The dimensions are given in screen coordinates that are relative to the upper-left corner of the screen.

[GetWindowRgn](#)

The GetWindowRgn function obtains a copy of the window region of a window.

[GetWindowRgnBox](#)

The GetWindowRgnBox function retrieves the dimensions of the tightest bounding rectangle for the window region of a window.

[GetWindowTextA](#)

Copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied. However, GetWindowText cannot retrieve the text of a control in another application.

[GetWindowTextLengthA](#)

Retrieves the length, in characters, of the specified window's title bar text (if the window has a title bar).

[GetWindowTextLengthW](#)

Retrieves the length, in characters, of the specified window's title bar text (if the window has a title bar).

[GetWindowTextW](#)

Copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied. However, GetWindowText cannot retrieve the text of a control in another application.

[GetWindowThreadProcessId](#)

Retrieves the identifier of the thread that created the specified window and, optionally, the identifier of the process that created the window.

[GID_ROTATE_ANGLE_FROM_ARGUMENT](#)

The GID_ROTATE_ANGLE_FROM_ARGUMENT macro is used to interpret the GID_ROTATE ullArgument value when receiving the value in the WM_GESTURE structure.

[GID_ROTATE_ANGLE_TO_ARGUMENT](#)

Converts a radian value to an argument for rotation gesture messages.

[GrayStringA](#)

The GrayString function draws gray text at the specified location.

[GrayStringW](#)

The GrayString function draws gray text at the specified location.

[HAS_POINTER_CONFIDENCE_WPARAM](#)

Checks whether the specified pointer message is considered intentional rather than accidental.

[HideCaret](#)

Removes the caret from the screen. Hiding a caret does not destroy its current shape or invalidate the insertion point.

[HiliteMenuItem](#)

Adds or removes highlighting from an item in a menu bar.

[InflateRect](#)

The InflateRect function increases or decreases the width and height of the specified rectangle.

[InitializeTouchInjection](#)

Configures the touch injection context for the calling application and initializes the maximum number of simultaneous contacts that the app can inject.

[InjectSyntheticPointerInput](#)

Simulates pointer input (pen or touch).

[InjectTouchInput](#)

Simulates touch input.

[InSendMessage](#)

Determines whether the current window procedure is processing a message that was sent from another thread (in the same process or a different process) by a call to the `SendMessage` function.

[InSendMessageEx](#)

Determines whether the current window procedure is processing a message that was sent from another thread (in the same process or a different process).

[InsertMenuA](#)

Inserts a new menu item into a menu, moving other items down the menu.

[InsertMenuItemA](#)

Inserts a new menu item at the specified position in a menu.

[InsertMenuItemW](#)

Inserts a new menu item at the specified position in a menu.

[InsertMenuW](#)

Inserts a new menu item into a menu, moving other items down the menu.

[InternalGetWindowText](#)

Copies the text of the specified window's title bar (if it has one) into a buffer.

[IntersectRect](#)

The `IntersectRect` function calculates the intersection of two source rectangles and places the coordinates of the intersection rectangle into the destination rectangle.

[InvalidateRect](#)

The `InvalidateRect` function adds a rectangle to the specified window's update region. The update region represents the portion of the window's client area that must be redrawn.

[InvalidateRgn](#)

The `InvalidateRgn` function invalidates the client area within the specified region by adding it to the current update region of a window.

[InvertRect](#)

The InvertRect function inverts a rectangle in a window by performing a logical NOT operation on the color values for each pixel in the rectangle's interior.

[IS_INTRESOURCE](#)

Determines whether a value is an integer identifier for a resource.

[IS_POINTER_CANCELED_WPARAM](#)

Checks whether the specified pointer input ended abruptly, or was invalid, indicating the interaction was not completed.

[IS_POINTER_FIFTHBUTTON_WPARAM](#)

Checks whether the specified pointer took fifth action.

[IS_POINTER_FIRSTBUTTON_WPARAM](#)

Checks whether the specified pointer took first action.

[IS_POINTER_FLAG_SET_WPARAM](#)

Checks whether a pointer macro sets the specified flag.

[IS_POINTER_FOURTHBUTTON_WPARAM](#)

Checks whether the specified pointer took fourth action.

[IS_POINTER_INCONTACT_WPARAM](#)

Checks whether the specified pointer is in contact.

[IS_POINTER_INRANGE_WPARAM](#)

Checks whether the specified pointer is in range.

[IS_POINTER_NEW_WPARAM](#)

Checks whether the specified pointer is a new pointer.

[IS_POINTER_SECONDBUTTON_WPARAM](#)

Checks whether the specified pointer took second action.

[IS_POINTER_THIRDBUTTON_WPARAM](#)

Checks whether the specified pointer took third action.

[IsCharAlphaA](#)

Determines whether a character is an alphabetical character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

[IsCharAlphaNumericA](#)

Determines whether a character is either an alphabetical or a numeric character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

[IsCharAlphaNumericW](#)

Determines whether a character is either an alphabetical or a numeric character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

[IsCharAlphaW](#)

Determines whether a character is an alphabetical character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

[IsCharLowerA](#)

Determines whether a character is lowercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

[IsCharLowerW](#)

[IsCharUpperA](#)

Determines whether a character is uppercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

[IsCharUpperW](#)

Determines whether a character is uppercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

[IsChild](#)

Determines whether a window is a child window or descendant window of a specified parent window.

[IsClipboardFormatAvailable](#)

Determines whether the clipboard contains data in the specified format.

[IsDialogMessageA](#)

Determines whether a message is intended for the specified dialog box and, if it is, processes the message.

[IsDialogMessageW](#)

Determines whether a message is intended for the specified dialog box and, if it is, processes the message.

[IsDlgButtonChecked](#)

The IsDlgButtonChecked function determines whether a button control is checked or whether a three-state button control is checked, unchecked, or indeterminate.

IsGUIThread	Determines whether the calling thread is already a GUI thread. It can also optionally convert the thread to a GUI thread.
IsHungAppWindow	Determines whether the system considers that a specified application is not responding.
IsIconic	Determines whether the specified window is minimized (iconic).
IsImmersiveProcess	Determines whether the process belongs to a Windows Store app.
IsMenu	Determines whether a handle is a menu handle.
IsMouseInPointerEnabled	Indicates whether EnableMouseInPointer is set for the mouse to act as a pointer input device and send WM_POINTER messages.
IsProcessDPIAware	IsProcessDPIAware may be altered or unavailable. Instead, use GetProcessDPIAwareness.
IsRectEmpty	The IsRectEmpty function determines whether the specified rectangle is empty.
IsTouchWindow	Checks whether a specified window is touch-capable and, optionally, retrieves the modifier flags set for the window's touch capability.
IsValidDpiAwarenessContext	Determines if a specified DPI_AWARENESS_CONTEXT is valid and supported by the current system.
IsWindow	Determines whether the specified window handle identifies an existing window.
IsWindowEnabled	Determines whether the specified window is enabled for mouse and keyboard input.
IsWindowUnicode	Determines whether the specified window is a native Unicode window.

IsWindowVisible	Determines the visibility state of the specified window.
IsWinEventHookInstalled	Determines whether there is an installed WinEvent hook that might be notified of a specified event.
IsWow64Message	Determines whether the last message read from the current thread's queue originated from a WOW64 process.
IsZoomed	Determines whether a window is maximized.
keybd_event	Synthesizes a keystroke.
KillTimer	Destroys the specified timer.
LoadAcceleratorsA	Loads the specified accelerator table.
LoadAcceleratorsW	Loads the specified accelerator table.
LoadBitmapA	The LoadBitmap function loads the specified bitmap resource from a module's executable file.
LoadBitmapW	The LoadBitmap function loads the specified bitmap resource from a module's executable file.
LoadCursorA	Loads the specified cursor resource from the executable (.EXE) file associated with an application instance.
LoadCursorFromFileA	Creates a cursor based on data contained in a file.
LoadCursorFromFileW	Creates a cursor based on data contained in a file.
LoadCursorW	Loads the specified cursor resource from the executable (.EXE) file associated with an application instance.

[LoadIconA](#)

Loads the specified icon resource from the executable (.exe) file associated with an application instance.

[LoadIconW](#)

Loads the specified icon resource from the executable (.exe) file associated with an application instance.

[LoadImageA](#)

Loads an icon, cursor, animated cursor, or bitmap.

[LoadImageW](#)

Loads an icon, cursor, animated cursor, or bitmap.

[LoadKeyboardLayoutA](#)

Loads a new input locale identifier (formerly called the keyboard layout) into the system.

[LoadKeyboardLayoutW](#)

Loads a new input locale identifier (formerly called the keyboard layout) into the system.

[LoadMenuA](#)

Loads the specified menu resource from the executable (.exe) file associated with an application instance.

[LoadMenuIndirectA](#)

Loads the specified menu template in memory.

[LoadMenuIndirectW](#)

Loads the specified menu template in memory.

[LoadMenuW](#)

Loads the specified menu resource from the executable (.exe) file associated with an application instance.

[LoadStringA](#)

Loads a string resource from the executable file associated with a specified module, copies the string into a buffer, and appends a terminating null character.

[LoadStringW](#)

Loads a string resource from the executable file associated with a specified module, copies the string into a buffer, and appends a terminating null character.

[LockSetForegroundWindow](#)

The foreground process can call the LockSetForegroundWindow function to disable calls to the SetForegroundWindow function.

[LockWindowUpdate](#)

The LockWindowUpdate function disables or enables drawing in the specified window. Only one window can be locked at a time.

[LockWorkStation](#)

Locks the workstation's display.

[LogicalToPhysicalPoint](#)

Converts the logical coordinates of a point in a window to physical coordinates.

[LogicalToPhysicalPointForPerMonitorDPI](#)

Converts a point in a window from logical coordinates into physical coordinates, regardless of the dots per inch (dpi) awareness of the caller.

[LookupIconIdFromDirectory](#)

Searches through icon or cursor data for the icon or cursor that best fits the current display device.

[LookupIconIdFromDirectoryEx](#)

Searches through icon or cursor data for the icon or cursor that best fits the current display device.

[MAKEINTRESOURCEA](#)

Converts an integer value to a resource type compatible with the resource-management functions. This macro is used in place of a string containing the name of the resource.

[MAKEINTRESOURCEW](#)

Converts an integer value to a resource type compatible with the resource-management functions. This macro is used in place of a string containing the name of the resource.

[MAKELPARAM](#)

Creates a value for use as an lParam parameter in a message. The macro concatenates the specified values.

[MAKELRESULT](#)

Creates a value for use as a return value from a window procedure. The macro concatenates the specified values.

[MAKEWPARAM](#)

Creates a value for use as a wParam parameter in a message. The macro concatenates the specified values.

[MapDialogRect](#)

Converts the specified dialog box units to screen units (pixels).

[MapVirtualKeyA](#)

Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.

[MapVirtualKeyExA](#)

Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.

[MapVirtualKeyExW](#)

Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.

[MapVirtualKeyW](#)

Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.

[MapWindowPoints](#)

The MapWindowPoints function converts (maps) a set of points from a coordinate space relative to one window to a coordinate space relative to another window.

[MenuItemFromPoint](#)

Determines which menu item, if any, is at the specified location.

[MessageBeep](#)

Plays a waveform sound. The waveform sound for each sound type is identified by an entry in the registry.

[MessageBox](#)

Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.

[MessageBoxA](#)

Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.

[MessageBoxExA](#)

Creates, displays, and operates a message box.

[MessageBoxExW](#)

Creates, displays, and operates a message box.

[MessageBoxIndirectA](#)

Creates, displays, and operates a message box. The message box contains application-defined message text and title, any icon, and any combination of predefined push buttons.

[MessageBoxIndirectW](#)

Creates, displays, and operates a message box. The message box contains application-defined message text and title, any icon, and any combination of predefined push buttons.

[MessageBoxW](#)

Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.

[ModifyMenuA](#)

Changes an existing menu item.

[ModifyMenuW](#)

Changes an existing menu item.

[MonitorFromPoint](#)

The MonitorFromPoint function retrieves a handle to the display monitor that contains a specified point.

[MonitorFromRect](#)

The MonitorFromRect function retrieves a handle to the display monitor that has the largest area of intersection with a specified rectangle.

[MonitorFromWindow](#)

The MonitorFromWindow function retrieves a handle to the display monitor that has the largest area of intersection with the bounding rectangle of a specified window.

[mouse_event](#)

The mouse_event function synthesizes mouse motion and button clicks.

[MoveWindow](#)

Changes the position and dimensions of the specified window.

[MsgWaitForMultipleObjects](#)

Waits until one or all of the specified objects are in the signaled state or the time-out interval elapses. The objects can include input event objects.

[MsgWaitForMultipleObjectsEx](#)

Waits until one or all of the specified objects are in the signaled state, an I/O completion routine or asynchronous procedure call (APC) is queued to the thread, or the time-out interval elapses. The array of objects can include input event objects.

[NEXTRAWINPUTBLOCK](#)

Retrieves the location of the next structure in an array of RAWINPUT structures.

[NotifyWinEvent](#)

Signals the system that a predefined event occurred. If any client applications have registered a hook function for the event, the system calls the client's hook function.

[OemKeyScan](#)

Maps OEMASCII codes 0 through 0x0FF into the OEM scan codes and shift states. The function provides information that allows a program to send OEM text to another program by simulating keyboard input.

[OemToCharA](#)

Translates a string from the OEM-defined character set into either an ANSI or a wide-character string.**Warning** Do not use.

[OemToCharBuffA](#)

Translates a specified number of characters in a string from the OEM-defined character set into either an ANSI or a wide-character string.

[OemToCharBuffW](#)

Translates a specified number of characters in a string from the OEM-defined character set into either an ANSI or a wide-character string.

[OemToCharW](#)

Translates a string from the OEM-defined character set into either an ANSI or a wide-character string.**Warning** Do not use.

[OffsetRect](#)

The OffsetRect function moves the specified rectangle by the specified offsets.

[OpenClipboard](#)

Opens the clipboard for examination and prevents other applications from modifying the clipboard content.

[OpenDesktopA](#)

Opens the specified desktop object.

[OpenDesktopW](#)

Opens the specified desktop object.

[OpenIcon](#)

Restores a minimized (iconic) window to its previous size and position; it then activates the window.

[OpenInputDesktop](#)

Opens the desktop that receives user input.

[OpenWindowStationA](#)

Opens the specified window station.

[OpenWindowStationW](#)

Opens the specified window station.

PackTouchHitTestingProximityEvaluation

Returns the proximity evaluation score and the adjusted touch-point coordinates as a packed value for the WM_TOUCHHITTESTING callback.

PaintDesktop

The PaintDesktop function fills the clipping region in the specified device context with the desktop pattern or wallpaper. The function is provided primarily for shell desktops.

PeekMessageA

Dispatches incoming nonqueued messages, checks the thread message queue for a posted message, and retrieves the message (if any exist).

PeekMessageW

Dispatches incoming nonqueued messages, checks the thread message queue for a posted message, and retrieves the message (if any exist).

PhysicalToLogicalPoint

Converts the physical coordinates of a point in a window to logical coordinates.

PhysicalToLogicalPointForPerMonitorDPI

Converts a point in a window from physical coordinates into logical coordinates, regardless of the dots per inch (dpi) awareness of the caller.

POINTSTOPOINT

The POINTSTOPOINT macro copies the contents of a POINTS structure into a POINT structure.

POINTTOPONTS

The POINTTOPONTS macro converts a POINT structure to a POINTS structure.

PostMessageA

Places (posts) a message in the message queue associated with the thread that created the specified window and returns without waiting for the thread to process the message.

PostMessageW

Places (posts) a message in the message queue associated with the thread that created the specified window and returns without waiting for the thread to process the message.

PostQuitMessage

Indicates to the system that a thread has made a request to terminate (quit). It is typically used in response to a WM_DESTROY message.

PostThreadMessageA

Posts a message to the message queue of the specified thread. It returns without waiting for the thread to process the message.

[PostThreadMessageW](#)

Posts a message to the message queue of the specified thread. It returns without waiting for the thread to process the message.

[PrintWindow](#)

The PrintWindow function copies a visual window into the specified device context (DC), typically a printer DC.

[PrivateExtractIconsA](#)

Creates an array of handles to icons that are extracted from a specified file.

[PrivateExtractIconsW](#)

Creates an array of handles to icons that are extracted from a specified file.

[PtInRect](#)

The PtInRect function determines whether the specified point lies within the specified rectangle.

[QueryDisplayConfig](#)

The QueryDisplayConfig function retrieves information about all possible display paths for all display devices, or views, in the current setting.

[RealChildWindowFromPoint](#)

Retrieves a handle to the child window at the specified point. The search is restricted to immediate child windows; grandchildren and deeper descendant windows are not searched.

[RealGetWindowClassA](#)

Retrieves a string that specifies the window type.

[RealGetWindowClassW](#)

Retrieves a string that specifies the window type.

[RedrawWindow](#)

The RedrawWindow function updates the specified rectangle or region in a window's client area.

[RegisterClassA](#)

Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.

[RegisterClassExA](#)

Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.

[RegisterClassExW](#)

Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.

[RegisterClassW](#)

Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.

[RegisterClipboardFormatA](#)

Registers a new clipboard format. This format can then be used as a valid clipboard format.

[RegisterClipboardFormatW](#)

Registers a new clipboard format. This format can then be used as a valid clipboard format.

[RegisterDeviceNotificationA](#)

Registers the device or type of device for which a window will receive notifications.

[RegisterDeviceNotificationW](#)

Registers the device or type of device for which a window will receive notifications.

[RegisterHotKey](#)

Defines a system-wide hot key.

[RegisterPointerDeviceNotifications](#)

Registers a window to process the WM_POINTERDEVICECHANGE, WM_POINTERDEVICEINRANGE, and WM_POINTERDEVICEOUTOFRANGE pointer device notifications.

[RegisterPointerInputTarget](#)

Allows the caller to register a target window to which all pointer input of the specified type is redirected.

[RegisterPointerInputTargetEx](#)

RegisterPointerInputTargetEx may be altered or unavailable. Instead, use RegisterPointerInputTarget.

[RegisterPowerSettingNotification](#)

Registers the application to receive power setting notifications for the specific power setting event.

[RegisterRawInputDevices](#)

Registers the devices that supply the raw input data.

[RegisterShellHookWindow](#)

Registers a specified Shell window to receive certain messages for events or notifications that are useful to Shell applications.

[RegisterSuspendResumeNotification](#)

Registers to receive notification when the system is suspended or resumed. Similar to PowerRegisterSuspendResumeNotification, but operates in user mode and can take a window handle.

[RegisterTouchHitTestingWindow](#)

Registers a window to process the WM_TOUCHHITTESTING notification.

[RegisterTouchWindow](#)

Registers a window as being touch-capable.

[RegisterWindowMessageA](#)

Defines a new window message that is guaranteed to be unique throughout the system. The message value can be used when sending or posting messages.

[RegisterWindowMessageW](#)

Defines a new window message that is guaranteed to be unique throughout the system. The message value can be used when sending or posting messages.

[ReleaseCapture](#)

Releases the mouse capture from a window in the current thread and restores normal mouse input processing.

[ReleaseDC](#)

The ReleaseDC function releases a device context (DC), freeing it for use by other applications. The effect of the ReleaseDC function depends on the type of DC. It frees only common and window DCs. It has no effect on class or private DCs.

[RemoveClipboardFormatListener](#)

Removes the given window from the system-maintained clipboard format listener list.

[RemoveMenu](#)

Deletes a menu item or detaches a submenu from the specified menu.

[RemovePropA](#)

Removes an entry from the property list of the specified window. The specified character string identifies the entry to be removed.

[RemovePropW](#)

Removes an entry from the property list of the specified window. The specified character string identifies the entry to be removed.

[ReplyMessage](#)

Replies to a message sent from another thread by the SendMessage function.

[ScreenToClient](#)

The ScreenToClient function converts the screen coordinates of a specified point on the screen to client-area coordinates.

[ScrollDC](#)

The ScrollDC function scrolls a rectangle of bits horizontally and vertically.

[ScrollWindow](#)

The ScrollWindow function scrolls the contents of the specified window's client area.

[ScrollWindowEx](#)

The ScrollWindowEx function scrolls the contents of the specified window's client area.

[SendDlgItemMessageA](#)

Sends a message to the specified control in a dialog box.

[SendDlgItemMessageW](#)

Sends a message to the specified control in a dialog box.

[SendInput](#)

Synthesizes keystrokes, mouse motions, and button clicks.

[SendMessage](#)

Sends the specified message to a window or windows. The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message.

[SendMessageA](#)

Sends the specified message to a window or windows. The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message.

[SendMessageCallbackA](#)

Sends the specified message to a window or windows.

[SendMessageCallbackW](#)

Sends the specified message to a window or windows.

[SendMessageTimeoutA](#)

Sends the specified message to one or more windows.

[SendMessageTimeoutW](#)

Sends the specified message to one or more windows.

[SendMessageW](#)

Sends the specified message to a window or windows. The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message.

[SendNotifyMessageA](#)

Sends the specified message to a window or windows.

[SendNotifyMessageW](#)

Sends the specified message to a window or windows.

[SetActiveWindow](#)

Activates a window. The window must be attached to the calling thread's message queue.

[SetCapture](#)

Sets the mouse capture to the specified window belonging to the current thread.

[SetCaretBlinkTime](#)

Sets the caret blink time to the specified number of milliseconds. The blink time is the elapsed time, in milliseconds, required to invert the caret's pixels.

[SetCaretPos](#)

Moves the caret to the specified coordinates. If the window that owns the caret was created with the CS_OWNDC class style, then the specified coordinates are subject to the mapping mode of the device context associated with that window.

[SetClassLongA](#)

Replaces the specified 32-bit (long) value at the specified offset into the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.

[SetClassLongPtrA](#)

Replaces the specified value at the specified offset in the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.

[SetClassLongPtrW](#)

Replaces the specified value at the specified offset in the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.

[SetClassLongW](#)

Replaces the specified 32-bit (long) value at the specified offset into the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.

[SetClassWord](#)

Replaces the 16-bit (WORD) value at the specified offset into the extra class memory for the window class to which the specified window belongs.

[SetClipboardData](#)

Places data on the clipboard in a specified clipboard format.

[SetClipboardViewer](#)

Adds the specified window to the chain of clipboard viewers. Clipboard viewer windows receive a WM_DRAWCLIPBOARD message whenever the content of the clipboard changes. This function is used for backward compatibility with earlier versions of Windows.

[SetCoalescableTimer](#)

Creates a timer with the specified time-out value and coalescing tolerance delay.

[SetCursor](#)

Sets the cursor shape.

[SetCursorPos](#)

Moves the cursor to the specified screen coordinates.

[SetDialogControlDpiChangeBehavior](#)

Overrides the default per-monitor DPI scaling behavior of a child window in a dialog.

[SetDialogDpiChangeBehavior](#)

Dialogs in Per-Monitor v2 contexts are automatically DPI scaled. This method lets you customize their DPI change behavior.

[SetDisplayAutoRotationPreferences](#)

Sets the screen auto-rotation preferences for the current process.

[SetDisplayConfig](#)

The SetDisplayConfig function modifies the display topology, source, and target modes by exclusively enabling the specified paths in the current session.

[SetDlgItemInt](#)

Sets the text of a control in a dialog box to the string representation of a specified integer value.

[SetDlgItemTextA](#)

Sets the title or text of a control in a dialog box.

[SetDlgItemTextW](#)

Sets the title or text of a control in a dialog box.

[SetDoubleClickTime](#)

Sets the double-click time for the mouse.

[SetFocus](#)

Sets the keyboard focus to the specified window. The window must be attached to the calling thread's message queue.

[SetForegroundWindow](#)

Brings the thread that created the specified window into the foreground and activates the window.

[SetGestureConfig](#)

Configures the messages that are sent from a window for Windows Touch gestures.

[SetKeyboardState](#)

Copies an array of keyboard key states into the calling thread's keyboard input-state table. This is the same table accessed by the GetKeyboardState and GetKeyState functions. Changes made to this table do not affect keyboard input to any other thread.

[SetLastErrorEx](#)

Sets the last-error code.

[SetLayeredWindowAttributes](#)

Sets the opacity and transparency color key of a layered window.

[SetMenu](#)

Assigns a new menu to the specified window.

[SetMenuItemContextHelpId](#)

Associates a Help context identifier with a menu.

[SetMenuItemDefaultItem](#)

Sets the default menu item for the specified menu.

[SetMenuItemInfo](#)

Sets information for a specified menu.

[SetMenuItemBitmaps](#)

Associates the specified bitmap with a menu item. Whether the menu item is selected or clear, the system displays the appropriate bitmap next to the menu item.

[SetMenuItemInfoA](#)

Changes information about a menu item.

[SetMenuItemInfoW](#)

Changes information about a menu item.

[SetMessageExtraInfo](#)

Sets the extra message information for the current thread.

[SetParent](#)

Changes the parent window of the specified child window.

[SetPhysicalCursorPos](#)

Sets the position of the cursor in physical coordinates.

[SetProcessDefaultLayout](#)

Changes the default layout when windows are created with no parent or owner only for the currently running process.

[SetProcessDPIAware](#)

SetProcessDPIAware may be altered or unavailable. Instead, use SetProcessDPIAwareness.

[SetProcessDpiAwarenessContext](#)

Sets the current process to a specified dots per inch (dpi) awareness context. The DPI awareness contexts are from the `DPI_AWARENESS_CONTEXT` value.

[SetProcessRestrictionExemption](#)

Exempts the calling process from restrictions preventing desktop processes from interacting with the Windows Store app environment. This function is used by development and debugging tools.

[SetProcessWindowStation](#)

Assigns the specified window station to the calling process.

[SetPropA](#)

Adds a new entry or changes an existing entry in the property list of the specified window.

[SetPropW](#)

Adds a new entry or changes an existing entry in the property list of the specified window.

[SetRect](#)

The `SetRect` function sets the coordinates of the specified rectangle. This is equivalent to assigning the left, top, right, and bottom arguments to the appropriate members of the `RECT` structure.

[SetRectEmpty](#)

The `SetRectEmpty` function creates an empty rectangle in which all coordinates are set to zero.

[SetScrollInfo](#)

The `SetScrollInfo` function sets the parameters of a scroll bar, including the minimum and maximum scrolling positions, the page size, and the position of the scroll box (thumb). The function also redraws the scroll bar, if requested.

[SetScrollPos](#)

The `SetScrollPos` function sets the position of the scroll box (thumb) in the specified scroll bar and, if requested, redraws the scroll bar to reflect the new position of the scroll box.

[SetScrollRange](#)

The `SetScrollRange` function sets the minimum and maximum scroll box positions for the specified scroll bar.

[SetSysColors](#)

Sets the colors for the specified display elements.

[SetSystemCursor](#)

Enables an application to customize the system cursors. It replaces the contents of the system cursor specified by the id parameter with the contents of the cursor specified by the hcur parameter and then destroys hcur.

[SetThreadCursorCreationScaling](#)

Sets the DPI scale for which the cursors being created on this thread are intended. This value is taken into account when scaling the cursor for the specific monitor on which it is being shown.

[SetThreadDesktop](#)

Assigns the specified desktop to the calling thread. All subsequent operations on the desktop use the access rights granted to the desktop.

[SetThreadDpiAwarenessContext](#)

Set the DPI awareness for the current thread to the provided value.

[SetThreadDpiHostingBehavior](#)

Sets the thread's DPI_HOSTING_BEHAVIOR. This behavior allows windows created in the thread to host child windows with a different DPI_AWARENESS_CONTEXT.

[SetTimer](#)

Creates a timer with the specified time-out value.

[SetUserObjectInformation](#)

Sets information about the specified window station or desktop object.

[SetUserObjectInformationW](#)

Sets information about the specified window station or desktop object.

[SetUserObjectSecurity](#)

Sets the security of a user object. This can be, for example, a window or a DDE conversation.

[SetWindowContextHelpId](#)

Associates a Help context identifier with the specified window.

[SetWindowDisplayAffinity](#)

Stores the display affinity setting in kernel mode on the hWnd associated with the window.

[SetWindowFeedbackSetting](#)

Sets the feedback configuration for a window.

[SetWindowLongA](#)

Changes an attribute of the specified window. The function also sets the 32-bit (long) value at the specified offset into the extra window memory.

[SetWindowLongPtrA](#)

Changes an attribute of the specified window.

[SetWindowLongPtrW](#)

Changes an attribute of the specified window.

[SetWindowLongW](#)

Changes an attribute of the specified window. The function also sets the 32-bit (long) value at the specified offset into the extra window memory.

[SetWindowPlacement](#)

Sets the show state and the restored, minimized, and maximized positions of the specified window.

[SetWindowPos](#)

Changes the size, position, and Z order of a child, pop-up, or top-level window. These windows are ordered according to their appearance on the screen. The topmost window receives the highest rank and is the first window in the Z order.

[SetWindowRgn](#)

The SetWindowRgn function sets the window region of a window.

[SetWindowsHookExA](#)

Installs an application-defined hook procedure into a hook chain.

[SetWindowsHookExW](#)

Installs an application-defined hook procedure into a hook chain.

[SetWindowTextA](#)

Changes the text of the specified window's title bar (if it has one). If the specified window is a control, the text of the control is changed. However, SetWindowText cannot change the text of a control in another application.

[SetWindowTextW](#)

Changes the text of the specified window's title bar (if it has one). If the specified window is a control, the text of the control is changed. However, SetWindowText cannot change the text of a control in another application.

[SetWinEventHook](#)

Sets an event hook function for a range of events.

ShowCaret

Makes the caret visible on the screen at the caret's current position. When the caret becomes visible, it begins flashing automatically.

ShowCursor

Displays or hides the cursor.

ShowOwnedPopups

Shows or hides all pop-up windows owned by the specified window.

ShowScrollBar

The ShowScrollBar function shows or hides the specified scroll bar.

ShowWindow

Sets the specified window's show state.

ShowWindowAsync

Sets the show state of a window without waiting for the operation to complete.

ShutdownBlockReasonCreate

Indicates that the system cannot be shut down and sets a reason string to be displayed to the user if system shutdown is initiated.

ShutdownBlockReasonDestroy

Indicates that the system can be shut down and frees the reason string.

ShutdownBlockReasonQuery

Retrieves the reason string set by the ShutdownBlockReasonCreate function.

SkipPointerFrameMessages

Determines which pointer input frame generated the most recently retrieved message for the specified pointer and discards any queued (unretrieved) pointer input messages generated from the same pointer input frame.

SoundSentry

Triggers a visual signal to indicate that a sound is playing.

SubtractRect

The SubtractRect function determines the coordinates of a rectangle formed by subtracting one rectangle from another.

SwapMouseButton

Reverses or restores the meaning of the left and right mouse buttons.

[SwitchDesktop](#)

Makes the specified desktop visible and activates it. This enables the desktop to receive input from the user.

[SwitchToThisWindow](#)

Switches focus to the specified window and brings it to the foreground.

[SystemParametersInfoA](#)

Retrieves or sets the value of one of the system-wide parameters.

[SystemParametersInfoForDpi](#)

Retrieves the value of one of the system-wide parameters, taking into account the provided DPI value.

[SystemParametersInfoW](#)

Retrieves or sets the value of one of the system-wide parameters.

[TabbedTextOutA](#)

The TabbedTextOut function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.

[TabbedTextOutW](#)

The TabbedTextOut function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.

[TileWindows](#)

Tiles the specified child windows of the specified parent window.

[ToAscii](#)

Translates the specified virtual-key code and keyboard state to the corresponding character or characters.

[ToAsciiEx](#)

Translates the specified virtual-key code and keyboard state to the corresponding character or characters. The function translates the code using the input language and physical keyboard layout identified by the input locale identifier.

[TOUCH_COORD_TO_PIXEL](#)

Converts touch coordinates to pixels.

[ToUnicode](#)

Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.

[ToUnicodeEx](#)

Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.

[TrackMouseEvent](#)

Posts messages when the mouse pointer leaves a window or hovers over a window for a specified amount of time.

[TrackPopupMenu](#)

Displays a shortcut menu at the specified location and tracks the selection of items on the menu. The shortcut menu can appear anywhere on the screen.

[TrackPopupMenuEx](#)

Displays a shortcut menu at the specified location and tracks the selection of items on the shortcut menu. The shortcut menu can appear anywhere on the screen.

[TranslateAcceleratorA](#)

Processes accelerator keys for menu commands.

[TranslateAcceleratorW](#)

Processes accelerator keys for menu commands.

[TranslateMDISysAccel](#)

Processes accelerator keystrokes for window menu commands of the multiple-document interface (MDI) child windows associated with the specified MDI client window.

[TranslateMessage](#)

Translates virtual-key messages into character messages. The character messages are posted to the calling thread's message queue, to be read the next time the thread calls the GetMessage or PeekMessage function.

[UnhookWindowsHookEx](#)

Removes a hook procedure installed in a hook chain by the SetWindowsHookEx function.

[UnhookWinEvent](#)

Removes an event hook function created by a previous call to SetWinEventHook.

[UnionRect](#)

The UnionRect function creates the union of two rectangles. The union is the smallest rectangle that contains both source rectangles.

[UnloadKeyboardLayout](#)

Unloads an input locale identifier (formerly called a keyboard layout).

[UnregisterClassA](#)

Unregisters a window class, freeing the memory required for the class.

[UnregisterClassW](#)

Unregisters a window class, freeing the memory required for the class.

[UnregisterDeviceNotification](#)

Closes the specified device notification handle.

[UnregisterHotKey](#)

Frees a hot key previously registered by the calling thread.

[UnregisterPointerInputTarget](#)

Allows the caller to unregister a target window to which all pointer input of the specified type is redirected.

[UnregisterPointerInputTargetEx](#)

UnregisterPointerInputTargetEx may be altered or unavailable. Instead, use UnregisterPointerInputTarget.

[UnregisterPowerSettingNotification](#)

Unregisters the power setting notification.

[UnregisterSuspendResumeNotification](#)

Cancels a registration to receive notification when the system is suspended or resumed. Similar to PowerUnregisterSuspendResumeNotification but operates in user mode.

[UnregisterTouchWindow](#)

Registers a window as no longer being touch-capable.

[UpdateLayeredWindow](#)

Updates the position, size, shape, content, and translucency of a layered window.

[UpdateWindow](#)

The UpdateWindow function updates the client area of the specified window by sending a WM_PAINT message to the window if the window's update region is not empty.

[UserHandleGrantAccess](#)

Grants or denies access to a handle to a User object to a job that has a user-interface restriction.

[ValidateRect](#)

The ValidateRect function validates the client area within a rectangle by removing the rectangle from the update region of the specified window.

[ValidateRgn](#)

The ValidateRgn function validates the client area within a region by removing the region from the current update region of the specified window.

[VkKeyScanA](#)

Translates a character to the corresponding virtual-key code and shift state for the current keyboard.

[VkKeyScanExA](#)

Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.

[VkKeyScanExW](#)

Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.

[VkKeyScanW](#)

Translates a character to the corresponding virtual-key code and shift state for the current keyboard.

[WaitForInputIdle](#)

Waits until the specified process has finished processing its initial input and is waiting for user input with no input pending, or until the time-out interval has elapsed.

[WaitMessage](#)

Yields control to other threads when a thread has no other messages in its message queue. The [WaitMessage](#) function suspends the thread and does not return until a new message is placed in the thread's message queue.

[WindowFromDC](#)

The [WindowFromDC](#) function returns a handle to the window associated with the specified display device context (DC). Output functions that use the specified device context draw into this window.

[WindowFromPhysicalPoint](#)

Retrieves a handle to the window that contains the specified physical point.

[WindowFromPoint](#)

Retrieves a handle to the window that contains the specified point.

[WinHelpA](#)

Launches Windows Help (`Winhelp.exe`) and passes additional data that indicates the nature of the help requested by the application.

[WinHelpW](#)

Launches Windows Help (`Winhelp.exe`) and passes additional data that indicates the nature of the help requested by the application.

[wsprintfA](#)

Writes formatted data to the specified buffer.

[wsprintfW](#)

Writes formatted data to the specified buffer.

[_vwsprintfA](#)

Writes formatted data to the specified buffer using a pointer to a list of arguments.

[_vwsprintfW](#)

Writes formatted data to the specified buffer using a pointer to a list of arguments.

Callback functions

[DLGPROC](#)

Application-defined callback function used with the CreateDialog and DialogBox families of functions.

[DRAWSTATEPROC](#)

The DrawStateProc function is an application-defined callback function that renders a complex image for the DrawState function.

[EDITWORDBREAKPROCA](#)

An application-defined callback function used with the EM_SETWORDBREAKPROC message.

[EDITWORDBREAKPROCW](#)

An application-defined callback function used with the EM_SETWORDBREAKPROC message.

[GRAYSTRINGPROC](#)

The OutputProc function is an application-defined callback function used with the GrayString function.

[HOOKPROC](#)

An application-defined or library-defined callback function used with the SetWindowsHookEx function. The system calls this function after the SendMessage function is called. The hook procedure can examine the message; it cannot modify it.

[MONITORENUMPROC](#)

A MonitorEnumProc function is an application-defined callback function that is called by the EnumDisplayMonitors function.

[MSGBOXCALLBACK](#)

A callback function, which you define in your application, that processes help events for the message box.

[PROOPENUMPROCA](#)

An application-defined callback function used with the EnumProps function.

[PROOPENUMPROCEXA](#)

Application-defined callback function used with the EnumPropsEx function.

[PROENUMPROCEW](#)

Application-defined callback function used with the EnumPropsEx function.

[PROENUMPROCW](#)

An application-defined callback function used with the EnumProps function.

[SENDASYNCPROC](#)

An application-defined callback function used with the SendMessageCallback function.

[TIMERPROC](#)

An application-defined callback function that processes WM_TIMER messages. The TIMERPROC type defines a pointer to this callback function. TimerProc is a placeholder for the application-defined function name.

[WINEVENTPROC](#)

An application-defined callback (or hook) function that the system calls in response to events generated by an accessible object.

[WNDPROC](#)

A callback function, which you define in your application, that processes messages sent to a window.

Structures

[ACCEL](#)

Defines an accelerator key used in an accelerator table.

[ACCESSTIMEOUT](#)

Contains information about the time-out period associated with the accessibility features.

[ALTTABINFO](#)

Contains status information for the application-switching (ALT+TAB) window.

[ANIMATIONINFO](#)

Describes the animation effects associated with user actions.

[AUDIODESCRIPTION](#)

Contains information associated with audio descriptions. This structure is used with the SystemParametersInfo function when the SPI_GETAUDIODESCRIPTION or SPI_SETAUDIODESCRIPTION action value is specified.

[BSMINFO](#)

Contains information about a window that denied a request from BroadcastSystemMessageEx.

CBT_CREATEWNDA

Contains information passed to a WH_CBT hook procedure, CBTProc, before a window is created.

CBT_CREATEWNDW

Contains information passed to a WH_CBT hook procedure, CBTProc, before a window is created.

CBTACTIVATESTRUCT

Contains information passed to a WH_CBT hook procedure, CBTProc, before a window is activated.

CHANGEFILTERSTRUCT

Contains extended result information obtained by calling the ChangeWindowMessageFilterEx function.

CLIENTCREATESTRUCT

Contains information about the menu and first multiple-document interface (MDI) child window of an MDI client window.

COMBOBOXINFO

Contains combo box status information.

COMPAREITEMSTRUCT

Supplies the identifiers and application-supplied data for two items in a sorted, owner-drawn list box or combo box.

COPYDATASTRUCT

Contains data to be passed to another application by the WM_COPYDATA message.

CREATESTRUCTA

Defines the initialization parameters passed to the window procedure of an application. These members are identical to the parameters of the CreateWindowEx function.

CREATESTRUCTW

Defines the initialization parameters passed to the window procedure of an application. These members are identical to the parameters of the CreateWindowEx function.

CURSORINFO

Contains global cursor information.

CURSORSHAPE

Contains information about a cursor.

CWPRETSTRUCT

Defines the message parameters passed to a WH_CALLWNDPROCRET hook procedure, CallWndRetProc.

CWPSTRUCT
Defines the message parameters passed to a WH_CALLWNDPROC hook procedure, CallWndProc.
DEBUGHOOKINFO
Contains debugging information passed to a WH_DEBUG hook procedure, DebugProc.
DELETEITEMSTRUCT
Describes a deleted list box or combo box item.
DLGITEMTEMPLATE
Defines the dimensions and style of a control in a dialog box. One or more of these structures are combined with a DLGTEMPLATE structure to form a standard template for a dialog box.
DLGTEMPLATE
Defines the dimensions and style of a dialog box.
DRAWITEMSTRUCT
Provides information that the owner window uses to determine how to paint an owner-drawn control or menu item.
DRAWTEXTPARAMS
The DRAWTEXTPARAMS structure contains extended formatting options for the DrawTextEx function.
EVENTMSG
Contains information about a hardware message sent to the system message queue. This structure is used to store message information for the JournalPlaybackProc callback function.
FILTERKEYS
Contains information about the FilterKeys accessibility feature, which enables a user with disabilities to set the keyboard repeat rate (RepeatKeys), acceptance delay (SlowKeys), and bounce rate (BounceKeys).
FLASHINFO
Contains the flash status for a window and the number of times the system should flash the window.
GESTURECONFIG
Gets and sets the configuration for enabling gesture messages and the type of this configuration.
GESTUREINFO
Stores information about a gesture.
GESTURENOTIFYSTRUCT
When transmitted with WM_GESTURENOTIFY messages, passes information about a gesture.

GUILHREADINFO

Contains information about a GUI thread.

HARDWAREINPUT

Contains information about a simulated message generated by an input device other than a keyboard or mouse.

HELPINFO

Contains information about an item for which context-sensitive help has been requested.

HELPWININFOA

Contains the size and position of either a primary or secondary Help window. An application can set this information by calling the WinHelp function with the HELP_SETWINPOS value.

HELPWININFOW

Contains the size and position of either a primary or secondary Help window. An application can set this information by calling the WinHelp function with the HELP_SETWINPOS value.

HIGHCONTRASTA

Contains information about the high contrast accessibility feature.

HIGHCONTRASTW

Contains information about the high contrast accessibility feature.

ICONINFO

Contains information about an icon or a cursor.

ICONINFOEXA

Contains information about an icon or a cursor. Extends ICONINFO. Used by GetIconInfoEx.

ICONINFOEXW

Contains information about an icon or a cursor. Extends ICONINFO. Used by GetIconInfoEx.

ICONMETRICS

Contains the scalable metrics associated with icons. This structure is used with the SystemParametersInfo function when the SPI_GETICONMETRICS or SPI_SETICONMETRICS action is specified.

ICONMETRICSW

Contains the scalable metrics associated with icons. This structure is used with the SystemParametersInfo function when the SPI_GETICONMETRICS or SPI_SETICONMETRICS action is specified.

INPUT

Used by SendInput to store information for synthesizing input events such as keystrokes, mouse movement, and mouse clicks.

INPUT_INJECTION_VALUE
Contains the input injection details.
INPUT_MESSAGE_SOURCE
Contains information about the source of the input message.
INPUT_TRANSFORM
Defines the matrix that represents a transform on a message consumer.
KBDLLHOOKSTRUCT
Contains information about a low-level keyboard input event.
KEYBDINPUT
Contains information about a simulated keyboard event.
LASTINPUTINFO
Contains the time of the last input.
MDICREATESTRUCTA
Contains information about the class, title, owner, location, and size of a multiple-document interface (MDI) child window.
MDICREATESTRUCTW
Contains information about the class, title, owner, location, and size of a multiple-document interface (MDI) child window.
MDINEXTMENU
Contains information about the menu to be activated.
MEASUREITEMSTRUCT
Informs the system of the dimensions of an owner-drawn control or menu item. This allows the system to process user interaction with the control correctly.
MENUBARINFO
Contains menu bar information.
MENUGETOBJECTINFO
Contains information about the menu that the mouse cursor is on.
MENUINFO
Contains information about a menu.

MENUITEMINFOA

Contains information about a menu item.

MENUITEMINFOW

Contains information about a menu item.

MENUITEMTEMPLATE

Defines a menu item in a menu template.

MENUITEMTEMPLATEHEADER

Defines the header for a menu template. A complete menu template consists of a header and one or more menu item lists.

MINIMIZEDMETRICS

Contains the scalable metrics associated with minimized windows.

MINMAXINFO

Contains information about a window's maximized size and position and its minimum and maximum tracking size.

MONITORINFO

The MONITORINFO structure contains information about a display monitor. The GetMonitorInfo function stores information in a MONITORINFO structure or a MONITORINFOEX structure. The MONITORINFO structure is a subset of the MONITORINFOEX structure.

MONITORINFOEXA

The MONITORINFOEX structure contains information about a display monitor. The GetMonitorInfo function stores information into a MONITORINFOEX structure or a MONITORINFO structure. The MONITORINFOEX structure is a superset of the MONITORINFO structure.

MONITORINFOEXW

The MONITORINFOEX structure contains information about a display monitor. The GetMonitorInfo function stores information into a MONITORINFOEX structure or a MONITORINFO structure. The MONITORINFOEX structure is a superset of the MONITORINFO structure.

MOUSEHOOKSTRUCT

Contains information about a mouse event passed to a WH_MOUSE hook procedure, MouseProc.

MOUSEHOOKSTRUCTEX

Contains information about a mouse event passed to a WH_MOUSE hook procedure, MouseProc. This is an extension of the MOUSEHOOKSTRUCT structure that includes information about wheel movement or the use of the X button.

MOUSEINPUT

Contains information about a simulated mouse event.

[MOUSEKEYS](#)

Contains information about the MouseKeys accessibility feature.

[MOUSEMOVEPOINT](#)

Contains information about the mouse's location in screen coordinates.

[MSG](#)

Contains message information from a thread's message queue.

[MSGBOXPARAMSA](#)

Contains information used to display a message box. The MessageBoxIndirect function uses this structure.

[MSGBOXPARAMSW](#)

Contains information used to display a message box. The MessageBoxIndirect function uses this structure.

[MSLHOOKSTRUCT](#)

Contains information about a low-level mouse input event.

[MULTIKEYHELPA](#)

Specifies a keyword to search for and the keyword table to be searched by Windows Help.

[MULTIKEYHELPW](#)

Specifies a keyword to search for and the keyword table to be searched by Windows Help.

[NCCALCSIZE_PARAMS](#)

Contains information that an application can use while processing the WM_NCCALCSIZE message to calculate the size, position, and valid contents of the client area of a window.

[NMHDR](#)

Contains information about a notification message.

[NONCLIENTMETRICS](#)

Contains the scalable metrics associated with the nonclient area of a nonminimized window.

[NONCLIENTMETRICSW](#)

Contains the scalable metrics associated with the nonclient area of a nonminimized window.

[PAINTSTRUCT](#)

The PAINTSTRUCT structure contains information for an application. This information can be used to paint the client area of a window owned by that application.

[POINTER_DEVICE_CURSOR_INFO](#)

Contains cursor ID mappings for pointer devices.

[POINTER_DEVICE_INFO](#)

Contains information about a pointer device. An array of these structures is returned from the GetPointerDevices function. A single structure is returned from a call to the GetPointerDevice function.

[POINTER_DEVICE_PROPERTY](#)

Contains pointer-based device properties (Human Interface Device (HID) global items that correspond to HID usages).

[POINTER_INFO](#)

Contains basic pointer information common to all pointer types. Applications can retrieve this information using the GetPointerInfo, GetPointerFrameInfo, GetPointerInfoHistory and GetPointerFrameInfoHistory functions.

[POINTER_PEN_INFO](#)

Defines basic pen information common to all pointer types.

[POINTER_TOUCH_INFO](#)

Defines basic touch information common to all pointer types.

[POINTER_TYPE_INFO](#)

Contains information about the pointer input type.

[POWERBROADCAST_SETTING](#)

Sent with a power setting event and contains data about the specific change.

[RAWHID](#)

Describes the format of the raw input from a Human Interface Device (HID).

[RAWINPUT](#)

Contains the raw input from a device.

[RAWINPUTDEVICE](#)

Defines information for the raw input devices.

[RAWINPUTDEVICELIST](#)

Contains information about a raw input device.

[RAWINPUTHEADER](#)

Contains the header information that is part of the raw input data.

[RAWKEYBOARD](#)

Contains information about the state of the keyboard.

[RAWMOUSE](#)

Contains information about the state of the mouse.

[RID_DEVICE_INFO](#)

Defines the raw input data coming from any device.

[RID_DEVICE_INFO_HID](#)

Defines the raw input data coming from the specified Human Interface Device (HID).

[RID_DEVICE_INFO_KEYBOARD](#)

Defines the raw input data coming from the specified keyboard.

[RID_DEVICE_INFO_MOUSE](#)

Defines the raw input data coming from the specified mouse.

[SCROLLBARINFO](#)

The SCROLLBARINFO structure contains scroll bar information.

[SCROLLINFO](#)

The SCROLLINFO structure contains scroll bar parameters to be set by the SetScrollInfo function (or SBM_SETSCROLLINFO message), or retrieved by the GetScrollInfo function (or SBM_GETSCROLLINFO message).

[SERIALKEYSA](#)

Contains information about the SerialKeys accessibility feature, which interprets data from a communication aid attached to a serial port as commands causing the system to simulate keyboard and mouse input.

[SERIALKEYSW](#)

Contains information about the SerialKeys accessibility feature, which interprets data from a communication aid attached to a serial port as commands causing the system to simulate keyboard and mouse input.

[SOUNDSENTRYA](#)

Contains information about the SoundSentry accessibility feature. When the SoundSentry feature is on, the computer displays a visual indication only when a sound is generated.

[SOUNDSENTRYW](#)

Contains information about the SoundSentry accessibility feature. When the SoundSentry feature is on, the computer displays a visual indication only when a sound is generated.

[STICKYKEYS](#)

Contains information about the StickyKeys accessibility feature.

[STYLESTRUCT](#)

Contains the styles for a window.

[TITLEBARINFO](#)

Contains title bar information.

[TITLEBARINFOEX](#)

Expands on the information described in the TITLEBARINFO structure by including the coordinates of each element of the title bar.

[TOGGLEKEYS](#)

Contains information about the ToggleKeys accessibility feature.

[TOUCH_HIT_TESTING_INPUT](#)

Contains information about the touch contact area reported by the touch digitizer.

[TOUCH_HIT_TESTING_PROXIMITY_EVALUATION](#)

Contains the hit test score that indicates whether the object is the likely target of the touch contact area, relative to other objects that intersect the touch contact area.

[TOUCHINPUT](#)

Encapsulates data for touch input.

[TOUCHPREDICTIONPARAMETERS](#)

Contains hardware input details that can be used to predict touch targets and help compensate for hardware latency when processing touch and gesture input that contains distance and velocity data.

[TPMPARAMS](#)

Contains extended parameters for the TrackPopupMenuEx function.

[TRACKMOUSEEVENT](#)

Used by the TrackMouseEvent function to track when the mouse pointer leaves a window or hovers over a window for a specified amount of time.

[UPDATELAYEREDWINDOWINFO](#)

Used by UpdateLayeredWindowIndirect to provide position, size, shape, content, and translucency information for a layered window.

[USAGE_PROPERTIES](#)

Contains device properties (Human Interface Device (HID) global items that correspond to HID usages) for any type of HID input device.

[USEROBJECTFLAGS](#)

Contains information about a window station or desktop handle.

[WINDOWINFO](#)

Contains window information.

[WINDOWPLACEMENT](#)

Contains information about the placement of a window on the screen.

[WINDOWPOS](#)

Contains information about the size and position of a window.

[WNDCLASSA](#)

Contains the window class attributes that are registered by the RegisterClass function.

[WNDCLASSEXA](#)

Contains window class information.

[WNDCLASSEXW](#)

Contains window class information.

[WNDCLASSW](#)

Contains the window class attributes that are registered by the RegisterClass function.

[WTSSESSION_NOTIFICATION](#)

Provides information about the session change notification. A service receives this structure in its HandlerEx function in response to a session change event.

Enumerations

[AR_STATE](#)

Indicates the state of screen auto-rotation for the system. For example, whether auto-rotation is supported, and whether it is enabled by the user.

[DIALOG_CONTROL_DPI_CHANGE_BEHAVIORS](#)

Describes per-monitor DPI scaling behavior overrides for child windows within dialogs. The values in this enumeration are bitfields and can be combined.

[DIALOG_DPI_CHANGE_BEHAVIORS](#)

In Per Monitor v2 contexts, dialogs will automatically respond to DPI changes by resizing themselves and re-computing the positions of their child windows (here referred to as re-layouting).

FEEDBACK_TYPE
Specifies the visual feedback associated with an event.
INPUT_MESSAGE_DEVICE_TYPE
The type of device that sent the input message.
INPUT_MESSAGE_ORIGIN_ID
The ID of the input message source.
ORIENTATION_PREFERENCE
Indicates the screen orientation preference for a desktop app process.
POINTER_BUTTON_CHANGE_TYPE
Identifies a change in the state of a button associated with a pointer.
POINTER_DEVICE_CURSOR_TYPE
Identifies the pointer device cursor types.
POINTER_DEVICE_TYPE
Identifies the pointer device types.
POINTER_FEEDBACK_MODE
Identifies the visual feedback behaviors available to CreateSyntheticPointerDevice.
tagPOINTER_INPUT_TYPE
Identifies the pointer input types.

BeginPaint function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **BeginPaint** function prepares the specified window for painting and fills a [PAINTSTRUCT](#) structure with information about the painting.

Syntax

```
HDC BeginPaint(
    [in]  HWND          hWnd,
    [out] LPPAINTSTRUCT lpPaint
);
```

Parameters

[in] hWnd

Handle to the window to be repainted.

[out] lpPaint

Pointer to the [PAINTSTRUCT](#) structure that will receive painting information.

Return value

If the function succeeds, the return value is the handle to a display device context for the specified window.

If the function fails, the return value is **NULL**, indicating that no display device context is available.

Remarks

The **BeginPaint** function automatically sets the clipping region of the device context to exclude any area outside the update region. The update region is set by the [InvalidateRect](#) or [InvalidateRgn](#) function and by the system after sizing, moving, creating, scrolling, or any other operation that affects the client area. If the update region is marked for erasing, **BeginPaint** sends a [WM_ERASEBKGND](#) message to the window.

An application should not call **BeginPaint** except in response to a [WM_PAINT](#) message. Each call to **BeginPaint** must have a corresponding call to the [EndPaint](#) function.

If the caret is in the area to be painted, **BeginPaint** automatically hides the caret to prevent it from being erased.

If the window's class has a background brush, **BeginPaint** uses that brush to erase the background of the update region before returning.

DPI Virtualization

This API does not participate in DPI virtualization. The output returned is always in terms of physical pixels.

Examples

For an example, see [Drawing in the Client Area](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[EndPaint](#)

[InvalidateRect](#)

[InvalidateRgn](#)

[PAINTSTRUCT](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[ValidateRect](#)

[ValidateRgn](#)

ChangeDisplaySettingsA function (winuser.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **ChangeDisplaySettings** function changes the settings of the default display device to the specified graphics mode.

To change the settings of a specified display device, use the [ChangeDisplaySettingsEx](#) function.

Note Apps that you design to target Windows 8 and later can no longer query or set display modes that are less than 32 bits per pixel (bpp); these operations will fail. These apps have a [compatibility manifest](#) that targets Windows 8. Windows 8 still supports 8-bit and 16-bit color modes for desktop apps that were built without a Windows 8 manifest; Windows 8 emulates these modes but still runs in 32-bit color mode.

Syntax

```
LONG ChangeDisplaySettingsA(
    [in] DEVMODEA *lpDevMode,
    [in] DWORD      dwFlags
);
```

Parameters

[in] *lpDevMode*

A pointer to a [DEVMODE](#) structure that describes the new graphics mode. If *lpDevMode* is **NULL**, all the values currently in the registry will be used for the display setting. Passing **NULL** for the *lpDevMode* parameter and 0 for the *dwFlags* parameter is the easiest way to return to the default mode after a dynamic mode change.

The **dmSize** member of [DEVMODE](#) must be initialized to the size, in bytes, of the [DEVMODE](#) structure. The **dmDriverExtra** member of [DEVMODE](#) must be initialized to indicate the number of bytes of private driver data following the [DEVMODE](#) structure. In addition, you can use any or all of the following members of the [DEVMODE](#) structure.

MEMBER	MEANING
dmBitsPerPel	Bits per pixel
dmPelsWidth	Pixel width
dmPelsHeight	Pixel height
dmDisplayFlags	Mode flags
dmDisplayFrequency	Mode frequency
dmPosition	Position of the device in a multi-monitor configuration.

In addition to using one or more of the preceding **DEVMODE** members, you must also set one or more of the following values in the **dmFields** member to change the display setting.

VALUE	MEANING
DM_BITSPERPEL	Use the dmBitsPerPel value.
DM_PELSWIDTH	Use the dmPelsWidth value.
DM_PELSHEIGHT	Use the dmPelsHeight value.
DM_DISPLAYFLAGS	Use the dmDisplayFlags value.
DM_DISPLAYFREQUENCY	Use the dmDisplayFrequency value.
DM_POSITION	Use the dmPosition value.

[in] **dwFlags**

Indicates how the graphics mode should be changed. This parameter can be one of the following values.

VALUE	MEANING
0	The graphics mode for the current screen will be changed dynamically.
CDS_FULLSCREEN	The mode is temporary in nature. If you change to and from another desktop, this mode will not be reset.
CDS_GLOBAL	The settings will be saved in the global settings area so that they will affect all users on the machine. Otherwise, only the settings for the user are modified. This flag is only valid when specified with the CDS_UPDATEREGISTRY flag.
CDS_NORESET	The settings will be saved in the registry, but will not take effect. This flag is only valid when specified with the CDS_UPDATEREGISTRY flag.
CDS_RESET	The settings should be changed, even if the requested settings are the same as the current settings.
CDS_SET_PRIMARY	This device will become the primary device.
CDS_TEST	The system tests if the requested graphics mode could be set.
CDS_UPDATEREGISTRY	The graphics mode for the current screen will be changed dynamically and the graphics mode will be updated in the registry. The mode information is stored in the USER profile.

Specifying CDS_TEST allows an application to determine which graphics modes are actually valid, without

causing the system to change to that graphics mode.

If CDS_UPDATEREGISTRY is specified and it is possible to change the graphics mode dynamically, the information is stored in the registry and DISP_CHANGE_SUCCESSFUL is returned. If it is not possible to change the graphics mode dynamically, the information is stored in the registry and DISP_CHANGE_RESTART is returned.

If CDS_UPDATEREGISTRY is specified and the information could not be stored in the registry, the graphics mode is not changed and DISP_CHANGE_NOTUPDATED is returned.

Return value

The ChangeDisplaySettings function returns one of the following values.

RETURN CODE	DESCRIPTION
DISP_CHANGE_SUCCESSFUL	The settings change was successful.
DISP_CHANGE_BADDUALVIEW	The settings change was unsuccessful because the system is DualView capable.
DISP_CHANGE_BADFLAGS	An invalid set of flags was passed in.
DISP_CHANGE_BADMODE	The graphics mode is not supported.
DISP_CHANGE_BADPARAM	An invalid parameter was passed in. This can include an invalid flag or combination of flags.
DISP_CHANGE_FAILED	The display driver failed the specified graphics mode.
DISP_CHANGE_NOTUPDATED	Unable to write settings to the registry.
DISP_CHANGE_RESTART	The computer must be restarted for the graphics mode to work.

Remarks

To ensure that the **DEVMODE** structure passed to **ChangeDisplaySettings** is valid and contains only values supported by the display driver, use the **DEVMODE** returned by the **EnumDisplaySettings** function.

When the display mode is changed dynamically, the **WM_DISPLAYCHANGE** message is sent to all running applications with the following message parameters.

PARAMETERS	MEANING
wParam	New bits per pixel
LOWORD(lParam)	New pixel width

HIWORD(IParam)	New pixel height
----------------	------------------

DPI Virtualization

This API does not participate in DPI virtualization. The input given is always in terms of physical pixels, and is not related to the calling context.

NOTE

The winuser.h header defines ChangeDisplaySettings as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[ChangeDisplaySettingsEx](#)

[CreateDC](#)

[DEVMODE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

[EnumDisplaySettings](#)

[WM_DISPLAYCHANGE](#)

ChangeDisplaySettingsExA function (winuser.h)

4/21/2022 • 5 minutes to read • [Edit Online](#)

The **ChangeDisplaySettingsEx** function changes the settings of the specified display device to the specified graphics mode.

Note Apps that you design to target Windows 8 and later can no longer query or set display modes that are less than 32 bits per pixel (bpp); these operations will fail. These apps have a [compatibility manifest](#) that targets Windows 8. Windows 8 still supports 8-bit and 16-bit color modes for desktop apps that were built without a Windows 8 manifest; Windows 8 emulates these modes but still runs in 32-bit color mode.

Syntax

```
LONG ChangeDisplaySettingsExA(
    [in] LPCSTR lpszDeviceName,
    [in] DEVMODEA *lpDevMode,
    [in] HWND     hwnd,
    [in] DWORD    dwFlags,
    [in] LPVOID   lParam
);
```

Parameters

[in] lpszDeviceName

A pointer to a null-terminated string that specifies the display device whose graphics mode will change. Only display device names as returned by [EnumDisplayDevices](#) are valid. See [EnumDisplayDevices](#) for further information on the names associated with these display devices.

The *lpszDeviceName* parameter can be **NULL**. A **NULL** value specifies the default display device. The default device can be determined by calling [EnumDisplayDevices](#) and checking for the **DISPLAY_DEVICE_PRIMARY_DEVICE** flag.

[in] lpDevMode

A pointer to a [DEVMODE](#) structure that describes the new graphics mode. If *lpDevMode* is **NULL**, all the values currently in the registry will be used for the display setting. Passing **NULL** for the *lpDevMode* parameter and 0 for the *dwFlags* parameter is the easiest way to return to the default mode after a dynamic mode change.

The **dmSize** member must be initialized to the size, in bytes, of the [DEVMODE](#) structure. The **dmDriverExtra** member must be initialized to indicate the number of bytes of private driver data following the [DEVMODE](#) structure. In addition, you can use any of the following members of the [DEVMODE](#) structure.

MEMBER	MEANING
dmBitsPerPel	Bits per pixel
dmPelsWidth	Pixel width

dmPelsHeight	Pixel height
dmDisplayFlags	Mode flags
dmDisplayFrequency	Mode frequency
dmPosition	Position of the device in a multi-monitor configuration.

In addition to using one or more of the preceding **DEVMODE** members, you must also set one or more of the following values in the **dmFields** member to change the display settings.

VALUE	MEANING
DM_BITSPERPEL	Use the dmBitsPerPel value.
DM_PELSWIDTH	Use the dmPelsWidth value.
DM_PELSHEIGHT	Use the dmPelsHeight value.
DM_DISPLAYFLAGS	Use the dmDisplayFlags value.
DM_DISPLAYFREQUENCY	Use the dmDisplayFrequency value.
DM_POSITION	Use the dmPosition value.

hwnd

Reserved; must be **NULL**.

[in] dwflags

Indicates how the graphics mode should be changed. This parameter can be one of the following values.

VALUE	MEANING
0	The graphics mode for the current screen will be changed dynamically.
CDS_FULLSCREEN	The mode is temporary in nature. If you change to and from another desktop, this mode will not be reset.
CDS_GLOBAL	The settings will be saved in the global settings area so that they will affect all users on the machine. Otherwise, only the settings for the user are modified. This flag is only valid when specified with the CDS_UPDATEREGISTRY flag.
CDS_NORESET	The settings will be saved in the registry, but will not take effect. This flag is only valid when specified with the CDS_UPDATEREGISTRY flag.
CDS_RESET	The settings should be changed, even if the requested settings are the same as the current settings.

CDS_SET_PRIMARY	This device will become the primary device.
CDS_TEST	The system tests if the requested graphics mode could be set.
CDS_UPDATEREGISTRY	The graphics mode for the current screen will be changed dynamically and the graphics mode will be updated in the registry. The mode information is stored in the USER profile.
CDS_VIDEOPARAMETERS	When set, the <i>lParam</i> parameter is a pointer to a VIDEOPARAMETERS structure.
CDS_ENABLE_UNSAFE_MODES	Enables settings changes to unsafe graphics modes.
CDS_DISABLE_UNSAFE_MODES	Disables settings changes to unsafe graphics modes.

Specifying CDS_TEST allows an application to determine which graphics modes are actually valid, without causing the system to change to them.

If CDS_UPDATEREGISTRY is specified and it is possible to change the graphics mode dynamically, the information is stored in the registry and DISP_CHANGE_SUCCESSFUL is returned. If it is not possible to change the graphics mode dynamically, the information is stored in the registry and DISP_CHANGE_RESTART is returned.

If CDS_UPDATEREGISTRY is specified and the information could not be stored in the registry, the graphics mode is not changed and DISP_CHANGE_NOTUPDATED is returned.

[in] lParam

If *dwFlags* is CDS_VIDEOPARAMETERS, *lParam* is a pointer to a [VIDEOPARAMETERS](#) structure. Otherwise *lParam* must be NULL.

Return value

The ChangeDisplaySettingsEx function returns one of the following values.

RETURN CODE	DESCRIPTION
DISP_CHANGE_SUCCESSFUL	The settings change was successful.
DISP_CHANGE_BADDUALVIEW	The settings change was unsuccessful because the system is DualView capable.
DISP_CHANGE_BADFLAGS	An invalid set of flags was passed in.
DISP_CHANGE_BADMODE	The graphics mode is not supported.

DISP_CHANGE_BADPARAM	An invalid parameter was passed in. This can include an invalid flag or combination of flags.
DISP_CHANGE_FAILED	The display driver failed the specified graphics mode.
DISP_CHANGE_NOTUPDATED	Unable to write settings to the registry.
DISP_CHANGE_RESTART	The computer must be restarted for the graphics mode to work.

Remarks

To ensure that the [DEVMODE](#) structure passed to [ChangeDisplaySettingsEx](#) is valid and contains only values supported by the display driver, use the [DEVMODE](#) returned by the [EnumDisplaySettings](#) function.

When adding a display monitor to a multiple-monitor system programmatically, set [DEVMODE.dmFields](#) to [DM_POSITION](#) and specify a position (in [DEVMODE.dmPosition](#)) for the monitor you are adding that is adjacent to at least one pixel of the display area of an existing monitor. To detach the monitor, set [DEVMODE.dmFields](#) to [DM_POSITION](#) but set [DEVMODE.dmPelsWidth](#) and [DEVMODE.dmPelsHeight](#) to zero. For more information, see [Multiple Display Monitors](#).

When the display mode is changed dynamically, the [WM_DISPLAYCHANGE](#) message is sent to all running applications with the following message parameters.

PARAMETERS	MEANING
wParam	New bits per pixel
LOWORD(lParam)	New pixel width
HIGHWORD(lParam)	New pixel height

To change the settings for more than one display at the same time, first call [ChangeDisplaySettingsEx](#) for each device individually to update the registry without applying the changes. Then call [ChangeDisplaySettingsEx](#) once more, with a [NULL](#) device, to apply the changes. For example, to change the settings for two displays, do the following:

```
ChangeDisplaySettingsEx (lpszDeviceName1, lpDevMode1, NULL, (CDS_UPDATEREGISTRY | CDS_NORESET), NULL);
ChangeDisplaySettingsEx (lpszDeviceName2, lpDevMode2, NULL, (CDS_UPDATEREGISTRY | CDS_NORESET), NULL);
ChangeDisplaySettingsEx (NULL, NULL, NULL, 0, NULL);
```

DPI Virtualization

This API does not participate in DPI virtualization. The input given is always in terms of physical pixels, and is not related to the calling context.

NOTE

The winuser.h header defines ChangeDisplaySettingsEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[CreateDC](#)

[DEVMODE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

[EnumDisplaySettings](#)

[VIDEOPARAMETERS](#)

[WM_DISPLAYCHANGE](#)

ChangeDisplaySettingsExW function (winuser.h)

4/21/2022 • 5 minutes to read • [Edit Online](#)

The **ChangeDisplaySettingsEx** function changes the settings of the specified display device to the specified graphics mode.

Note Apps that you design to target Windows 8 and later can no longer query or set display modes that are less than 32 bits per pixel (bpp); these operations will fail. These apps have a [compatibility manifest](#) that targets Windows 8. Windows 8 still supports 8-bit and 16-bit color modes for desktop apps that were built without a Windows 8 manifest; Windows 8 emulates these modes but still runs in 32-bit color mode.

Syntax

```
LONG ChangeDisplaySettingsExW(
    [in] LPCWSTR lpszDeviceName,
    [in] DEVMODEW *lpDevMode,
    HWND     hwnd,
    [in] DWORD   dwFlags,
    [in] LPVOID  lParam
);
```

Parameters

[in] lpszDeviceName

A pointer to a null-terminated string that specifies the display device whose graphics mode will change. Only display device names as returned by [EnumDisplayDevices](#) are valid. See [EnumDisplayDevices](#) for further information on the names associated with these display devices.

The *lpszDeviceName* parameter can be **NULL**. A **NULL** value specifies the default display device. The default device can be determined by calling [EnumDisplayDevices](#) and checking for the **DISPLAY_DEVICE_PRIMARY_DEVICE** flag.

[in] lpDevMode

A pointer to a [DEVMODE](#) structure that describes the new graphics mode. If *lpDevMode* is **NULL**, all the values currently in the registry will be used for the display setting. Passing **NULL** for the *lpDevMode* parameter and 0 for the *dwFlags* parameter is the easiest way to return to the default mode after a dynamic mode change.

The **dmSize** member must be initialized to the size, in bytes, of the [DEVMODE](#) structure. The **dmDriverExtra** member must be initialized to indicate the number of bytes of private driver data following the [DEVMODE](#) structure. In addition, you can use any of the following members of the [DEVMODE](#) structure.

MEMBER	MEANING
dmBitsPerPel	Bits per pixel
dmPelsWidth	Pixel width

dmPelsHeight	Pixel height
dmDisplayFlags	Mode flags
dmDisplayFrequency	Mode frequency
dmPosition	Position of the device in a multi-monitor configuration.

In addition to using one or more of the preceding **DEVMODE** members, you must also set one or more of the following values in the **dmFields** member to change the display settings.

VALUE	MEANING
DM_BITSPERPEL	Use the dmBitsPerPel value.
DM_PELSWIDTH	Use the dmPelsWidth value.
DM_PELSHEIGHT	Use the dmPelsHeight value.
DM_DISPLAYFLAGS	Use the dmDisplayFlags value.
DM_DISPLAYFREQUENCY	Use the dmDisplayFrequency value.
DM_POSITION	Use the dmPosition value.

hwnd

Reserved; must be **NULL**.

[in] dwflags

Indicates how the graphics mode should be changed. This parameter can be one of the following values.

VALUE	MEANING
0	The graphics mode for the current screen will be changed dynamically.
CDS_FULLSCREEN	The mode is temporary in nature. If you change to and from another desktop, this mode will not be reset.
CDS_GLOBAL	The settings will be saved in the global settings area so that they will affect all users on the machine. Otherwise, only the settings for the user are modified. This flag is only valid when specified with the CDS_UPDATEREGISTRY flag.
CDS_NORESET	The settings will be saved in the registry, but will not take effect. This flag is only valid when specified with the CDS_UPDATEREGISTRY flag.
CDS_RESET	The settings should be changed, even if the requested settings are the same as the current settings.

CDS_SET_PRIMARY	This device will become the primary device.
CDS_TEST	The system tests if the requested graphics mode could be set.
CDS_UPDATEREGISTRY	The graphics mode for the current screen will be changed dynamically and the graphics mode will be updated in the registry. The mode information is stored in the USER profile.
CDS_VIDEOPARAMETERS	When set, the <i>lParam</i> parameter is a pointer to a VIDEOPARAMETERS structure.
CDS_ENABLE_UNSAFE_MODES	Enables settings changes to unsafe graphics modes.
CDS_DISABLE_UNSAFE_MODES	Disables settings changes to unsafe graphics modes.

Specifying CDS_TEST allows an application to determine which graphics modes are actually valid, without causing the system to change to them.

If CDS_UPDATEREGISTRY is specified and it is possible to change the graphics mode dynamically, the information is stored in the registry and DISP_CHANGE_SUCCESSFUL is returned. If it is not possible to change the graphics mode dynamically, the information is stored in the registry and DISP_CHANGE_RESTART is returned.

If CDS_UPDATEREGISTRY is specified and the information could not be stored in the registry, the graphics mode is not changed and DISP_CHANGE_NOTUPDATED is returned.

[in] lParam

If *dwFlags* is CDS_VIDEOPARAMETERS, *lParam* is a pointer to a [VIDEOPARAMETERS](#) structure. Otherwise *lParam* must be NULL.

Return value

The ChangeDisplaySettingsEx function returns one of the following values.

RETURN CODE	DESCRIPTION
DISP_CHANGE_SUCCESSFUL	The settings change was successful.
DISP_CHANGE_BADDUALVIEW	The settings change was unsuccessful because the system is DualView capable.
DISP_CHANGE_BADFLAGS	An invalid set of flags was passed in.
DISP_CHANGE_BADMODE	The graphics mode is not supported.

DISP_CHANGE_BADPARAM	An invalid parameter was passed in. This can include an invalid flag or combination of flags.
DISP_CHANGE_FAILED	The display driver failed the specified graphics mode.
DISP_CHANGE_NOTUPDATED	Unable to write settings to the registry.
DISP_CHANGE_RESTART	The computer must be restarted for the graphics mode to work.

Remarks

To ensure that the [DEVMODE](#) structure passed to [ChangeDisplaySettingsEx](#) is valid and contains only values supported by the display driver, use the [DEVMODE](#) returned by the [EnumDisplaySettings](#) function.

When adding a display monitor to a multiple-monitor system programmatically, set [DEVMODE.dmFields](#) to [DM_POSITION](#) and specify a position (in [DEVMODE.dmPosition](#)) for the monitor you are adding that is adjacent to at least one pixel of the display area of an existing monitor. To detach the monitor, set [DEVMODE.dmFields](#) to [DM_POSITION](#) but set [DEVMODE.dmPelsWidth](#) and [DEVMODE.dmPelsHeight](#) to zero. For more information, see [Multiple Display Monitors](#).

When the display mode is changed dynamically, the [WM_DISPLAYCHANGE](#) message is sent to all running applications with the following message parameters.

PARAMETERS	MEANING
wParam	New bits per pixel
LOWORD(lParam)	New pixel width
HIGHWORD(lParam)	New pixel height

To change the settings for more than one display at the same time, first call [ChangeDisplaySettingsEx](#) for each device individually to update the registry without applying the changes. Then call [ChangeDisplaySettingsEx](#) once more, with a [NULL](#) device, to apply the changes. For example, to change the settings for two displays, do the following:

```
ChangeDisplaySettingsEx (lpszDeviceName1, lpDevMode1, NULL, (CDS_UPDATEREGISTRY | CDS_NORESET), NULL);
ChangeDisplaySettingsEx (lpszDeviceName2, lpDevMode2, NULL, (CDS_UPDATEREGISTRY | CDS_NORESET), NULL);
ChangeDisplaySettingsEx (NULL, NULL, NULL, 0, NULL);
```

DPI Virtualization

This API does not participate in DPI virtualization. The input given is always in terms of physical pixels, and is not related to the calling context.

NOTE

The winuser.h header defines ChangeDisplaySettingsEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[CreateDC](#)

[DEVMODE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

[EnumDisplaySettings](#)

[VIDEOPARAMETERS](#)

[WM_DISPLAYCHANGE](#)

ChangeDisplaySettingsW function (winuser.h)

4/21/2022 • 4 minutes to read • [Edit Online](#)

The **ChangeDisplaySettings** function changes the settings of the default display device to the specified graphics mode.

To change the settings of a specified display device, use the [ChangeDisplaySettingsEx](#) function.

Note Apps that you design to target Windows 8 and later can no longer query or set display modes that are less than 32 bits per pixel (bpp); these operations will fail. These apps have a [compatibility manifest](#) that targets Windows 8. Windows 8 still supports 8-bit and 16-bit color modes for desktop apps that were built without a Windows 8 manifest; Windows 8 emulates these modes but still runs in 32-bit color mode.

Syntax

```
LONG ChangeDisplaySettingsW(
    [in] DEVMODEW *lpDevMode,
    [in] DWORD      dwFlags
);
```

Parameters

[in] *lpDevMode*

A pointer to a [DEVMODE](#) structure that describes the new graphics mode. If *lpDevMode* is **NULL**, all the values currently in the registry will be used for the display setting. Passing **NULL** for the *lpDevMode* parameter and 0 for the *dwFlags* parameter is the easiest way to return to the default mode after a dynamic mode change.

The **dmSize** member of [DEVMODE](#) must be initialized to the size, in bytes, of the [DEVMODE](#) structure. The **dmDriverExtra** member of [DEVMODE](#) must be initialized to indicate the number of bytes of private driver data following the [DEVMODE](#) structure. In addition, you can use any or all of the following members of the [DEVMODE](#) structure.

MEMBER	MEANING
dmBitsPerPel	Bits per pixel
dmPelsWidth	Pixel width
dmPelsHeight	Pixel height
dmDisplayFlags	Mode flags
dmDisplayFrequency	Mode frequency
dmPosition	Position of the device in a multi-monitor configuration.

In addition to using one or more of the preceding **DEVMODE** members, you must also set one or more of the following values in the **dmFields** member to change the display setting.

VALUE	MEANING
DM_BITSPERPEL	Use the dmBitsPerPel value.
DM_PELSWIDTH	Use the dmPelsWidth value.
DM_PELSHEIGHT	Use the dmPelsHeight value.
DM_DISPLAYFLAGS	Use the dmDisplayFlags value.
DM_DISPLAYFREQUENCY	Use the dmDisplayFrequency value.
DM_POSITION	Use the dmPosition value.

[in] **dwFlags**

Indicates how the graphics mode should be changed. This parameter can be one of the following values.

VALUE	MEANING
0	The graphics mode for the current screen will be changed dynamically.
CDS_FULLSCREEN	The mode is temporary in nature. If you change to and from another desktop, this mode will not be reset.
CDS_GLOBAL	The settings will be saved in the global settings area so that they will affect all users on the machine. Otherwise, only the settings for the user are modified. This flag is only valid when specified with the CDS_UPDATEREGISTRY flag.
CDS_NORESET	The settings will be saved in the registry, but will not take effect. This flag is only valid when specified with the CDS_UPDATEREGISTRY flag.
CDS_RESET	The settings should be changed, even if the requested settings are the same as the current settings.
CDS_SET_PRIMARY	This device will become the primary device.
CDS_TEST	The system tests if the requested graphics mode could be set.
CDS_UPDATEREGISTRY	The graphics mode for the current screen will be changed dynamically and the graphics mode will be updated in the registry. The mode information is stored in the USER profile.

Specifying CDS_TEST allows an application to determine which graphics modes are actually valid, without

causing the system to change to that graphics mode.

If CDS_UPDATEREGISTRY is specified and it is possible to change the graphics mode dynamically, the information is stored in the registry and DISP_CHANGE_SUCCESSFUL is returned. If it is not possible to change the graphics mode dynamically, the information is stored in the registry and DISP_CHANGE_RESTART is returned.

If CDS_UPDATEREGISTRY is specified and the information could not be stored in the registry, the graphics mode is not changed and DISP_CHANGE_NOTUPDATED is returned.

Return value

The ChangeDisplaySettings function returns one of the following values.

RETURN CODE	DESCRIPTION
DISP_CHANGE_SUCCESSFUL	The settings change was successful.
DISP_CHANGE_BADDUALVIEW	The settings change was unsuccessful because the system is DualView capable.
DISP_CHANGE_BADFLAGS	An invalid set of flags was passed in.
DISP_CHANGE_BADMODE	The graphics mode is not supported.
DISP_CHANGE_BADPARAM	An invalid parameter was passed in. This can include an invalid flag or combination of flags.
DISP_CHANGE_FAILED	The display driver failed the specified graphics mode.
DISP_CHANGE_NOTUPDATED	Unable to write settings to the registry.
DISP_CHANGE_RESTART	The computer must be restarted for the graphics mode to work.

Remarks

To ensure that the **DEVMODE** structure passed to **ChangeDisplaySettings** is valid and contains only values supported by the display driver, use the **DEVMODE** returned by the **EnumDisplaySettings** function.

When the display mode is changed dynamically, the **WM_DISPLAYCHANGE** message is sent to all running applications with the following message parameters.

PARAMETERS	MEANING
wParam	New bits per pixel
LOWORD(lParam)	New pixel width

HIWORD(IParam)	New pixel height
----------------	------------------

DPI Virtualization

This API does not participate in DPI virtualization. The input given is always in terms of physical pixels, and is not related to the calling context.

NOTE

The winuser.h header defines ChangeDisplaySettings as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[ChangeDisplaySettingsEx](#)

[CreateDC](#)

[DEVMODE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

[EnumDisplaySettings](#)

[WM_DISPLAYCHANGE](#)

ClientToScreen function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ClientToScreen** function converts the client-area coordinates of a specified point to screen coordinates.

Syntax

```
BOOL ClientToScreen(
    [in]      HWND      hWnd,
    [in, out] LPPOINT  lpPoint
);
```

Parameters

[in] hWnd

A handle to the window whose client area is used for the conversion.

[in, out] lpPoint

A pointer to a [POINT](#) structure that contains the client coordinates to be converted. The new screen coordinates are copied into this structure if the function succeeds.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The **ClientToScreen** function replaces the client-area coordinates in the [POINT](#) structure with the screen coordinates. The screen coordinates are relative to the upper-left corner of the screen. Note, a screen-coordinate point that is above the window's client area has a negative y-coordinate. Similarly, a screen coordinate to the left of a client area has a negative x-coordinate.

All coordinates are device coordinates.

Examples

For an example, see "Drawing Lines with the Mouse" in [Using Mouse Input](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[MapWindowPoints](#)

[POINT](#)

[ScreenToClient](#)

CopyRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **CopyRect** function copies the coordinates of one rectangle to another.

Syntax

```
BOOL CopyRect(
    [out] LPRECT     lprcDst,
    [in]  const RECT *lprcSrc
);
```

Parameters

[out] lprcDst

Pointer to the [RECT](#) structure that receives the logical coordinates of the source rectangle.

[in] lprcSrc

Pointer to the [RECT](#) structure whose coordinates are to be copied in logical units.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Examples

For an example, see [Using Rectangles](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib

DLL	User32.dll
-----	------------

See also

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

[SetRect](#)

[SetRectEmpty](#)

DrawAnimatedRects function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

Animates the caption of a window to indicate the opening of an icon or the minimizing or maximizing of a window.

Syntax

```
BOOL DrawAnimatedRects(
    [in] HWND      hwnd,
    [in] int       idAni,
    const RECT *lprcFrom,
    const RECT *lprcTo
);
```

Parameters

[in] hwnd

A handle to the window whose caption should be animated on the screen. The animation will be clipped to the parent of this window.

[in] idAni

The type of animation. This must be IDANI_CAPTION. With the IDANI_CAPTION animation type, the window caption will animate from the position specified by *lprcFrom* to the position specified by *lprcTo*. The effect is similar to minimizing or maximizing a window.

lprcFrom

A pointer to a [RECT](#) structure specifying the location and size of the icon or minimized window. Coordinates are relative to the clipping window *hwnd*.

lprcTo

A pointer to a [RECT](#) structure specifying the location and size of the restored window. Coordinates are relative to the clipping window *hwnd*.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

DrawCaption function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DrawCaption** function draws a window caption.

Syntax

```
BOOL DrawCaption(
    [in] HWND      hwnd,
    [in] HDC       hdc,
    [in] const RECT *lprect,
    [in] UINT      flags
);
```

Parameters

[in] **hwnd**

A handle to a window that supplies text and an icon for the window caption.

[in] **hdc**

A handle to a device context. The function draws the window caption into this device context.

[in] **lprect**

A pointer to a [RECT](#) structure that specifies the bounding rectangle for the window caption in logical coordinates.

[in] **flags**

The drawing options. This parameter can be zero or more of the following values.

VALUE	MEANING
DC_ACTIVE	The function uses the colors that denote an active caption.
DC_BUTTONS	If set, the function draws the buttons in the caption bar (to minimize, restore, or close an application).
DC_GRADIENT	When this flag is set, the function uses COLOR_GRADIENTACTIVECAPTION (if the DC_ACTIVE flag was set) or COLOR_GRADIENTINACTIVECAPTION for the title-bar color. If this flag is not set, the function uses COLOR_ACTIVECAPTION or COLOR_INACTIVECAPTION for both colors.
DC_ICON	The function draws the icon when drawing the caption text.

DC_INBUTTON	The function draws the caption as a button.
DC_SMALLCAP	The function draws a small caption, using the current small caption font.
DC_TEXT	The function draws the caption text when drawing the caption.

If DC_SMALLCAP is specified, the function draws a normal window caption.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

DrawEdge function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DrawEdge** function draws one or more edges of rectangle.

Syntax

```
BOOL DrawEdge(
    [in]      HDC      hdc,
    [in, out] LPRECT qrc,
    [in]      UINT     edge,
    [in]      UINT     grffFlags
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in, out] `qrc`

A pointer to a [RECT](#) structure that contains the logical coordinates of the rectangle.

[in] `edge`

The type of inner and outer edges to draw. This parameter must be a combination of one inner-border flag and one outer-border flag. The inner-border flags are as follows.

VALUE	MEANING
<code>BDR_RAISEDINNER</code>	Raised inner edge.
<code>BDR_SUNKENINNER</code>	Sunken inner edge.

The outer-border flags are as follows.

VALUE	MEANING
<code>BDR_RAISEDOUTER</code>	Raised outer edge.
<code>BDR_SUNKENOUTER</code>	Sunken outer edge.

Alternatively, the `edge` parameter can specify one of the following flags.

VALUE	MEANING
EDGE_BUMP	Combination of BDR_RAISEDOUTER and BDR_SUNKENINNER.
EDGEETCHED	Combination of BDR_SUNKENOUTER and BDR_RAISEDINNER.
EDGE_RAISED	Combination of BDR_RAISEDOUTER and BDR_RAISEDINNER.
EDGE_SUNKEN	Combination of BDR_SUNKENOUTER and BDR_SUNKENINNER.

[in] grfFlags

The type of border. This parameter can be a combination of the following values.

VALUE	MEANING
BF_ADJUST	If this flag is passed, shrink the rectangle pointed to by the <i>qrc</i> parameter to exclude the edges that were drawn. If this flag is not passed, then do not change the rectangle pointed to by the <i>qrc</i> parameter.
BF_BOTTOM	Bottom of border rectangle.
BF_BOTTOMLEFT	Bottom and left side of border rectangle.
BF_BOTTOMRIGHT	Bottom and right side of border rectangle.
BF_DIAGONAL	Diagonal border.
BF_DIAGONAL_ENDBOTTOMLEFT	Diagonal border. The end point is the lower-left corner of the rectangle; the origin is top-right corner.
BF_DIAGONAL_ENDBOTTOMRIGHT	Diagonal border. The end point is the lower-right corner of the rectangle; the origin is top-left corner.
BF_DIAGONAL_ENDTOPLEFT	Diagonal border. The end point is the top-left corner of the rectangle; the origin is lower-right corner.
BF_DIAGONAL_ENDTOPRIGHT	Diagonal border. The end point is the top-right corner of the rectangle; the origin is lower-left corner.
BF_FLAT	Flat border.

BF_LEFT	Left side of border rectangle.
BF_MIDDLE	Interior of rectangle to be filled.
BF_MONO	One-dimensional border.
BF_RECT	Entire border rectangle.
BF_RIGHT	Right side of border rectangle.
BF_SOFT	Soft buttons instead of tiles.
BF_TOP	Top of border rectangle.
BF_TOPLEFT	Top and left side of border rectangle.
BF_TOPRIGHT	Top and right side of border rectangle.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

DrawFocusRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DrawFocusRect** function draws a rectangle in the style used to indicate that the rectangle has the focus.

Syntax

```
BOOL DrawFocusRect(
    [in] HDC      hdc,
    [in] const RECT *lprc
);
```

Parameters

[in] `hDC`

A handle to the device context.

[in] `lprc`

A pointer to a [RECT](#) structure that specifies the logical coordinates of the rectangle.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

DrawFocusRect works only in MM_TEXT mode.

Because **DrawFocusRect** is an XOR function, calling it a second time with the same rectangle removes the rectangle from the screen.

This function draws a rectangle that cannot be scrolled. To scroll an area containing a rectangle drawn by this function, call **DrawFocusRect** to remove the rectangle from the screen, scroll the area, and then call **DrawFocusRect** again to draw the rectangle in the new position.

Windows XP: The focus rectangle can now be thicker than 1 pixel, so it is more visible for high-resolution, high-density displays and accessibility needs. This is handled by the SPI_SETFOCUSBORDERWIDTH and SPI_SETFOCUSBORDERHEIGHT in [SystemParametersInfo](#).

Examples

For an example, see "Creating an Owner-Drawn List Box" in [Using List Boxes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[FrameRect](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

DrawFrameControl function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DrawFrameControl** function draws a frame control of the specified type and style.

Syntax

```
BOOL DrawFrameControl(
    [in] HDC     unnamedParam1,
    [in] LPRECT  unnamedParam2,
    [in] UINT    unnamedParam3,
    [in] UINT    unnamedParam4
);
```

Parameters

[in] `unnamedParam1`

A handle to the device context of the window in which to draw the control.

[in] `unnamedParam2`

A pointer to a [RECT](#) structure that contains the logical coordinates of the bounding rectangle for frame control.

[in] `unnamedParam3`

The type of frame control to draw. This parameter can be one of the following values.

VALUE	MEANING
<code>DFC_BUTTON</code>	Standard button
<code>DFC_CAPTION</code>	Title bar
<code>DFC_MENU</code>	Menu bar
<code>DFC_POPUPMENU</code>	Popup menu item.
<code>DFC_SCROLL</code>	Scroll bar

[in] `unnamedParam4`

The initial state of the frame control. If *uType* is `DFC_BUTTON`, *uState* can be one of the following values.

VALUE	MEANING
-------	---------

DFCS_BUTTON3STATE	Three-state button
DFCS_BUTTONCHECK	Check box
DFCS_BUTTONPUSH	Push button
DFCS_BUTTONRADIO	Radio button
DFCS_BUTTONRADIOIMAGE	Image for radio button (nonsquare needs image)
DFCS_BUTTONRADIOMASK	Mask for radio button (nonsquare needs mask)

If *uType* is DFC_CAPTION, *uState* can be one of the following values.

VALUE	MEANING
DFCS_CAPTIONCLOSE	Close button
DFCS_CAPTIONHELP	Help button
DFCS_CAPTIONMAX	Maximize button
DFCS_CAPTIONMIN	Minimize button
DFCS_CAPTIONRESTORE	Restore button

If *uType* is DFC_MENU, *uState* can be one of the following values.

VALUE	MEANING
DFCS_MENUARROW	Submenu arrow
DFCS_MENUARROWRIGHT	Submenu arrow pointing left. This is used for the right-to-left cascading menus used with right-to-left languages such as Arabic or Hebrew.

DFCS_MENUBULLET	Bullet
DFCS_MENUCHECK	Check mark

If *uType* is DFC_SCROLL, *uState* can be one of the following values.

VALUE	MEANING
DFCS_SCROLLCOMBOBOX	Combo box scroll bar
DFCS_SCROLLDOWN	Down arrow of scroll bar
DFCS_SCROLLLEFT	Left arrow of scroll bar
DFCS_SCROLLRIGHT	Right arrow of scroll bar
DFCS_SCROLLSIZEGRIP	Size grip in lower-right corner of window
DFCS_SCROLLSIZEGRIPRIGHT	Size grip in lower-left corner of window. This is used with right-to-left languages such as Arabic or Hebrew.
DFCS_SCROLLUP	Up arrow of scroll bar

The following style can be used to adjust the bounding rectangle of the push button.

VALUE	MEANING
DFCS_ADJUSTRECT	Bounding rectangle is adjusted to exclude the surrounding edge of the push button.

One or more of the following values can be used to set the state of the control to be drawn.

VALUE	MEANING
DFCS_CHECKED	Button is checked.
DFCS_FLAT	Button has a flat border.

DFCS_HOT	Button is hot-tracked.
DFCS_INACTIVE	Button is inactive (grayed).
DFCS_MONO	Button has a monochrome border.
DFCS_PUSHED	Button is pushed.
DFCS_TRANSPARENT	The background remains untouched. This flag can only be combined with DFCS_MENUARROWUP or DFCS_MENUARROWDOWN.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

If *uType* is either DFC_MENU or DFC_BUTTON and *uState* is not DFCS_BUTTONPUSH, the frame control is a black-on-white mask (that is, a black frame control on a white background). In such cases, the application must pass a handle to a bitmap memory device control. The application can then use the associated bitmap as the *hbmMask* parameter to the [MaskBlt](#) function, or it can use the device context as a parameter to the [BitBlt](#) function using ROPs such as SRCAND and SRCINVERT.

DPI Virtualization

This API does not participate in DPI virtualization. The input given is always in terms of physical pixels, and is not related to the calling context.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

DrawStateA function (winuser.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **DrawState** function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.

Syntax

```
BOOL DrawStateA(
    [in] HDC         hdc,
    [in] HBRUSH      hbrFore,
    [in] DRAWSTATEPROC qfnCallBack,
    [in] LPARAM      lData,
    [in] WPARAM      wData,
    [in] int          x,
    [in] int          y,
    [in] int          cx,
    [in] int          cy,
    [in] UINT         uFlags
);
```

Parameters

[in] `hdc`

A handle to the device context to draw in.

[in] `hbrFore`

A handle to the brush used to draw the image, if the state specified by the *fuFlags* parameter is DSS_MONO. This parameter is ignored for other states.

[in] `qfnCallBack`

A pointer to an application-defined callback function used to render the image. This parameter is required if the image type in *fuFlags* is DST_COMPLEX. It is optional and can be **NULL** if the image type is DST_TEXT. For all other image types, this parameter is ignored. For more information about the callback function, see the [DrawStateProc](#) function.

[in] `lData`

Information about the image. The meaning of this parameter depends on the image type.

[in] `wData`

Information about the image. The meaning of this parameter depends on the image type. It is, however, zero extended for use with the [DrawStateProc](#) function.

[in] `x`

The horizontal location, in device units, at which to draw the image.

[in] `y`

The vertical location, in device units, at which to draw the image.

[in] cx

The width of the image, in device units. This parameter is required if the image type is DST_COMPLEX. Otherwise, it can be zero to calculate the width of the image.

[in] cy

The height of the image, in device units. This parameter is required if the image type is DST_COMPLEX. Otherwise, it can be zero to calculate the height of the image.

[in] uFlags

The image type and state. This parameter can be one of the following type values.

VALUE (TYPE)	MEANING
DST_BITMAP	The image is a bitmap. The <i>lData</i> parameter is the bitmap handle. Note that the bitmap cannot already be selected into an existing device context.
DST_COMPLEX	The image is application defined. To render the image, DrawState calls the callback function specified by the <i>lpOutputFunc</i> parameter.
DST_ICON	The image is an icon. The <i>lData</i> parameter is the icon handle.
DST_PREFIXTEXT	The image is text that may contain an accelerator mnemonic. DrawState interprets the ampersand (&) prefix character as a directive to underscore the character that follows. The <i>lData</i> parameter is a pointer to the string, and the <i>wData</i> parameter specifies the length. If <i>wData</i> is zero, the string must be null-terminated.
DST_TEXT	The image is text. The <i>lData</i> parameter is a pointer to the string, and the <i>wData</i> parameter specifies the length. If <i>wData</i> is zero, the string must be null-terminated.

This parameter can also be one of the following state values.

VALUE (STATE)	MEANING
DSS_DISABLED	Embosses the image.
DSS_HIDEPREFIX	Ignores the ampersand (&) prefix character in the text, thus the letter that follows will not be underlined. This must be used with DST_PREFIXTEXT.
DSS_MONO	Draws the image using the brush specified by the <i>hbr</i> parameter.
DSS_NORMAL	Draws the image without any modification.

DSS_PREFIXONLY	Draws only the underline at the position of the letter after the ampersand (&) prefix character. No text in the string is drawn. This must be used with DST_PREFIXTEXT.
DSS_RIGHT	Aligns the text to the right.
DSS_UNION	Dithers the image.

For all states except DSS_NORMAL, the image is converted to monochrome before the visual effect is applied.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

NOTE

The winuser.h header defines DrawState as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[DrawStateProc](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

DRAWSTATEPROC callback function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DrawStateProc** function is an application-defined callback function that renders a complex image for the [DrawState](#) function. The **DRAWSTATEPROC** type defines a pointer to this callback function. **DrawStateProc** is a placeholder for the application-defined function name.

Syntax

```
DRAWSTATEPROC Drawstateproc;

BOOL Drawstateproc(
    [in] HDC hdc,
    [in] LPARAM lData,
    [in] WPARAM wData,
    [in] int cx,
    [in] int cy
)
{...}
```

Parameters

[in] **hdc**

A handle to the device context to draw in. The device context is a memory device context with a bitmap selected, the dimensions of which are at least as great as those specified by the *cx* and *cy* parameters.

[in] **lData**

Specifies information about the image, which the application passed to [DrawState](#).

[in] **wData**

Specifies information about the image, which the application passed to [DrawState](#).

[in] **cx**

The image width, in device units, as specified by the call to [DrawState](#).

[in] **cy**

The image height, in device units, as specified by the call to [DrawState](#).

Return value

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

[DrawState](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

DrawStateW function (winuser.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **DrawState** function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.

Syntax

```
BOOL DrawStateW(
    [in] HDC         hdc,
    [in] HBRUSH      hbrFore,
    [in] DRAWSTATEPROC qfnCallBack,
    [in] LPARAM      lData,
    [in] WPARAM      wData,
    [in] int          x,
    [in] int          y,
    [in] int          cx,
    [in] int          cy,
    [in] UINT         uFlags
);
```

Parameters

[in] `hdc`

A handle to the device context to draw in.

[in] `hbrFore`

A handle to the brush used to draw the image, if the state specified by the *fuFlags* parameter is DSS_MONO. This parameter is ignored for other states.

[in] `qfnCallBack`

A pointer to an application-defined callback function used to render the image. This parameter is required if the image type in *fuFlags* is DST_COMPLEX. It is optional and can be **NULL** if the image type is DST_TEXT. For all other image types, this parameter is ignored. For more information about the callback function, see the [DrawStateProc](#) function.

[in] `lData`

Information about the image. The meaning of this parameter depends on the image type.

[in] `wData`

Information about the image. The meaning of this parameter depends on the image type. It is, however, zero extended for use with the [DrawStateProc](#) function.

[in] `x`

The horizontal location, in device units, at which to draw the image.

[in] `y`

The vertical location, in device units, at which to draw the image.

[in] cx

The width of the image, in device units. This parameter is required if the image type is DST_COMPLEX. Otherwise, it can be zero to calculate the width of the image.

[in] cy

The height of the image, in device units. This parameter is required if the image type is DST_COMPLEX. Otherwise, it can be zero to calculate the height of the image.

[in] uFlags

The image type and state. This parameter can be one of the following type values.

VALUE (TYPE)	MEANING
DST_BITMAP	The image is a bitmap. The <i>lData</i> parameter is the bitmap handle. Note that the bitmap cannot already be selected into an existing device context.
DST_COMPLEX	The image is application defined. To render the image, DrawState calls the callback function specified by the <i>lpOutputFunc</i> parameter.
DST_ICON	The image is an icon. The <i>lData</i> parameter is the icon handle.
DST_PREFIXTEXT	The image is text that may contain an accelerator mnemonic. DrawState interprets the ampersand (&) prefix character as a directive to underscore the character that follows. The <i>lData</i> parameter is a pointer to the string, and the <i>wData</i> parameter specifies the length. If <i>wData</i> is zero, the string must be null-terminated.
DST_TEXT	The image is text. The <i>lData</i> parameter is a pointer to the string, and the <i>wData</i> parameter specifies the length. If <i>wData</i> is zero, the string must be null-terminated.

This parameter can also be one of the following state values.

VALUE (STATE)	MEANING
DSS_DISABLED	Embosses the image.
DSS_HIDEPREFIX	Ignores the ampersand (&) prefix character in the text, thus the letter that follows will not be underlined. This must be used with DST_PREFIXTEXT.
DSS_MONO	Draws the image using the brush specified by the <i>hbr</i> parameter.
DSS_NORMAL	Draws the image without any modification.

DSS_PREFIXONLY	Draws only the underline at the position of the letter after the ampersand (&) prefix character. No text in the string is drawn. This must be used with DST_PREFIXTEXT.
DSS_RIGHT	Aligns the text to the right.
DSS_UNION	Dithers the image.

For all states except DSS_NORMAL, the image is converted to monochrome before the visual effect is applied.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

NOTE

The winuser.h header defines DrawState as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[DrawStateProc](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

DrawText function (winuser.h)

4/21/2022 • 5 minutes to read • [Edit Online](#)

The **DrawText** function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

To specify additional formatting options, use the [DrawTextEx](#) function.

Syntax

```
int DrawText(
    [in]      HDC      hdc,
    [in, out] LPCTSTR  lpchText,
    [in]      int       cchText,
    [in, out] LPRECT   lprc,
    [in]      UINT      format
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in, out] `lpchText`

A pointer to the string that specifies the text to be drawn. If the *nCount* parameter is -1, the string must be null-terminated.

If *uFormat* includes DT_MODIFYSTRING, the function could add up to four additional characters to this string. The buffer containing the string should be large enough to accommodate these extra characters.

[in] `cchText`

The length, in characters, of the string. If *nCount* is -1, then the *lpchText* parameter is assumed to be a pointer to a null-terminated string and **DrawText** computes the character count automatically.

[in, out] `lprc`

A pointer to a [RECT](#) structure that contains the rectangle (in logical coordinates) in which the text is to be formatted.

[in] `format`

The method of formatting the text. This parameter can be one or more of the following values.

VALUE	MEANING
DT_BOTTOM	Justifies the text to the bottom of the rectangle. This value is used only with the DT_SINGLELINE value.

DT_CALCRECT	Determines the width and height of the rectangle. If there are multiple lines of text, DrawText uses the width of the rectangle pointed to by the <i>lpRect</i> parameter and extends the base of the rectangle to bound the last line of text. If the largest word is wider than the rectangle, the width is expanded. If the text is less than the width of the rectangle, the width is reduced. If there is only one line of text, DrawText modifies the right side of the rectangle so that it bounds the last character in the line. In either case, DrawText returns the height of the formatted text but does not draw the text.
DT_CENTER	Centers text horizontally in the rectangle.
DT_EDITCONTROL	Duplicates the text-displaying characteristics of a multiline edit control. Specifically, the average character width is calculated in the same manner as for an edit control, and the function does not display a partially visible last line.
DT_END_ELLIPSIS	<p>For displayed text, if the end of a string does not fit in the rectangle, it is truncated and ellipses are added. If a word that is not at the end of the string goes beyond the limits of the rectangle, it is truncated without ellipses.</p> <p>The string is not modified unless the DT_MODIFYSTRING flag is specified.</p> <p>Compare with DT_PATH_ELLIPSIS and DT_WORD_ELLIPSIS.</p>
DT_EXPANDTABS	Expands tab characters. The default number of characters per tab is eight. The DT_WORD_ELLIPSIS, DT_PATH_ELLIPSIS, and DT_END_ELLIPSIS values cannot be used with the DT_EXPANDTABS value.
DT_EXTERNALLEADING	Includes the font external leading in line height. Normally, external leading is not included in the height of a line of text.
DT_HIDEPREFIX	<p>Ignores the ampersand (&) prefix character in the text. The letter that follows will not be underlined, but other mnemonic-prefix characters are still processed.</p> <p>Example:</p> <p>input string: "A&bc&d"</p> <p>normal: "Abc&d"</p> <p>DT_HIDEPREFIX: "Abc&d"</p> <p>Compare with DT_NOPREFIX and DT_PREFIXONLY.</p>
DT_INTERNAL	Uses the system font to calculate text metrics.
DT_LEFT	Aligns text to the left.

DT_MODIFYSTRING	Modifies the specified string to match the displayed text. This value has no effect unless DT_END_ELLIPSIS or DT_PATH_ELLIPSIS is specified.
DT_NOCLIP	Draws without clipping. DrawText is somewhat faster when DT_NOCLIP is used.
DT_NOFULLWIDTHCHARBREAK	Prevents a line break at a DBCS (double-wide character string), so that the line breaking rule is equivalent to SBCS strings. For example, this can be used in Korean windows, for more readability of icon labels. This value has no effect unless DT_WORDBREAK is specified.
DT_NOPREFIX	<p>Turns off processing of prefix characters. Normally, DrawText interprets the mnemonic-prefix character & as a directive to underscore the character that follows, and the mnemonic-prefix characters && as a directive to print a single &. By specifying DT_NOPREFIX, this processing is turned off. For example,</p> <p>Example:</p> <p>input string: "A&bc&&d"</p> <p>normal: "<u>A</u>bc&<u>d</u>"</p> <p>DT_NOPREFIX: "A&bc&&d"</p> <p>Compare with DT_HIDEPREFIX and DT_PREFIXONLY.</p>
DT_PATH_ELLIPSIS	<p>For displayed text, replaces characters in the middle of the string with ellipses so that the result fits in the specified rectangle. If the string contains backslash (\) characters, DT_PATH_ELLIPSIS preserves as much as possible of the text after the last backslash.</p> <p>The string is not modified unless the DT MODIFYSTRING flag is specified.</p> <p>Compare with DT_END_ELLIPSIS and DT_WORD_ELLIPSIS.</p>
DT_PREFIXONLY	<p>Draws only an underline at the position of the character following the ampersand (&) prefix character. Does not draw any other characters in the string. For example,</p> <p>Example:</p> <p>input string: "A&bc&&d"\n</p> <p>normal: "<u>A</u>bc&<u>d</u>"</p> <p>DT_PREFIXONLY: " _ "</p> <p>Compare with DT_HIDEPREFIX and DT_NOPREFIX.</p>
DT_RIGHT	Aligns text to the right.
DT_RTLREADING	Layout in right-to-left reading order for bidirectional text when the font selected into the <i>hdc</i> is a Hebrew or Arabic font. The default reading order for all text is left-to-right.

DT_SINGLELINE	Displays text on a single line only. Carriage returns and line feeds do not break the line.
DT_TABSTOP	Sets tab stops. Bits 15-8 (high-order byte of the low-order word) of the <i>uFormat</i> parameter specify the number of characters for each tab. The default number of characters per tab is eight. The DT_CALCRECT, DT_EXTERNALLEADING, DT_INTERNAL, DT_NOCLIP, and DT_NOPREFIX values cannot be used with the DT_TABSTOP value.
DT_TOP	Justifies the text to the top of the rectangle.
DT_VCENTER	Centers text vertically. This value is used only with the DT_SINGLELINE value.
DT_WORDBREAK	Breaks words. Lines are automatically broken between words if a word would extend past the edge of the rectangle specified by the <i>lpRect</i> parameter. A carriage return-line feed sequence also breaks the line. If this is not specified, output is on one line.
DT_WORD_ELLIPSIS	Truncates any word that does not fit in the rectangle and adds ellipses. Compare with DT_END_ELLIPSIS and DT_PATH_ELLIPSIS.

Return value

If the function succeeds, the return value is the height of the text in logical units. If DT_VCENTER or DT_BOTTOM is specified, the return value is the offset from `lpRect->top` to the bottom of the drawn text

If the function fails, the return value is zero.

Remarks

The **DrawText** function uses the device context's selected font, text color, and background color to draw the text. Unless the DT_NOCLIP format is used, **DrawText** clips the text so that it does not appear outside the specified rectangle. Note that text with significant overhang may be clipped, for example, an initial "W" in the text string or text that is in italics. All formatting is assumed to have multiple lines unless the DT_SINGLELINE format is specified.

If the selected font is too large for the specified rectangle, the **DrawText** function does not attempt to substitute a smaller font.

The text alignment mode for the device context must include the TA_LEFT, TA_TOP, and TA_NOUPDATECP flags.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-misc-l1-1-0 (introduced in Windows 8)

See also

[DrawTextEx](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GrayString](#)

[RECT](#)

[TabbedTextOut](#)

[TextOut](#)

DrawTextA function (winuser.h)

4/21/2022 • 6 minutes to read • [Edit Online](#)

The **DrawText** function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

To specify additional formatting options, use the [DrawTextEx](#) function.

Syntax

```
int DrawTextA(
    [in]      HDC      hdc,
    [in, out] LPCSTR  lpchText,
    [in]      int       cchText,
    [in, out] LPRECT   lprc,
    [in]      UINT      format
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in, out] `lpchText`

A pointer to the string that specifies the text to be drawn. If the *nCount* parameter is -1, the string must be null-terminated.

If *uFormat* includes DT_MODIFYSTRING, the function could add up to four additional characters to this string. The buffer containing the string should be large enough to accommodate these extra characters.

[in] `cchText`

The length, in characters, of the string. If *nCount* is -1, then the *lpchText* parameter is assumed to be a pointer to a null-terminated string and **DrawText** computes the character count automatically.

[in, out] `lprc`

A pointer to a [RECT](#) structure that contains the rectangle (in logical coordinates) in which the text is to be formatted.

[in] `format`

The method of formatting the text. This parameter can be one or more of the following values.

VALUE	MEANING
DT_BOTTOM	Justifies the text to the bottom of the rectangle. This value is used only with the DT_SINGLELINE value.

DT_CALCRECT	Determines the width and height of the rectangle. If there are multiple lines of text, DrawText uses the width of the rectangle pointed to by the <i>lpRect</i> parameter and extends the base of the rectangle to bound the last line of text. If the largest word is wider than the rectangle, the width is expanded. If the text is less than the width of the rectangle, the width is reduced. If there is only one line of text, DrawText modifies the right side of the rectangle so that it bounds the last character in the line. In either case, DrawText returns the height of the formatted text but does not draw the text.
DT_CENTER	Centers text horizontally in the rectangle.
DT_EDITCONTROL	Duplicates the text-displaying characteristics of a multiline edit control. Specifically, the average character width is calculated in the same manner as for an edit control, and the function does not display a partially visible last line.
DT_END_ELLIPSIS	<p>For displayed text, if the end of a string does not fit in the rectangle, it is truncated and ellipses are added. If a word that is not at the end of the string goes beyond the limits of the rectangle, it is truncated without ellipses.</p> <p>The string is not modified unless the DT_MODIFYSTRING flag is specified.</p> <p>Compare with DT_PATH_ELLIPSIS and DT_WORD_ELLIPSIS.</p>
DT_EXPANDTABS	Expands tab characters. The default number of characters per tab is eight. The DT_WORD_ELLIPSIS, DT_PATH_ELLIPSIS, and DT_END_ELLIPSIS values cannot be used with the DT_EXPANDTABS value.
DT_EXTERNALLEADING	Includes the font external leading in line height. Normally, external leading is not included in the height of a line of text.
DT_HIDEPREFIX	<p>Ignores the ampersand (&) prefix character in the text. The letter that follows will not be underlined, but other mnemonic-prefix characters are still processed.</p> <p>Example:</p> <p>input string: "A&bc&d"</p> <p>normal: "Abc&d"</p> <p>DT_HIDEPREFIX: "Abc&d"</p> <p>Compare with DT_NOPREFIX and DT_PREFIXONLY.</p>
DT_INTERNAL	Uses the system font to calculate text metrics.
DT_LEFT	Aligns text to the left.

DT_MODIFYSTRING	Modifies the specified string to match the displayed text. This value has no effect unless DT_END_ELLIPSIS or DT_PATH_ELLIPSIS is specified.
DT_NOCLIP	Draws without clipping. DrawText is somewhat faster when DT_NOCLIP is used.
DT_NOFULLWIDTHCHARBREAK	Prevents a line break at a DBCS (double-wide character string), so that the line breaking rule is equivalent to SBCS strings. For example, this can be used in Korean windows, for more readability of icon labels. This value has no effect unless DT_WORDBREAK is specified.
DT_NOPREFIX	<p>Turns off processing of prefix characters. Normally, DrawText interprets the mnemonic-prefix character & as a directive to underscore the character that follows, and the mnemonic-prefix characters && as a directive to print a single &. By specifying DT_NOPREFIX, this processing is turned off. For example,</p> <p>Example:</p> <p>input string: "A&bc&&d"</p> <p>normal: "<u>A</u>bc&<u>d</u>"</p> <p>DT_NOPREFIX: "A&bc&&d"</p> <p>Compare with DT_HIDEPREFIX and DT_PREFIXONLY.</p>
DT_PATH_ELLIPSIS	<p>For displayed text, replaces characters in the middle of the string with ellipses so that the result fits in the specified rectangle. If the string contains backslash (\) characters, DT_PATH_ELLIPSIS preserves as much as possible of the text after the last backslash.</p> <p>The string is not modified unless the DT MODIFYSTRING flag is specified.</p> <p>Compare with DT_END_ELLIPSIS and DT_WORD_ELLIPSIS.</p>
DT_PREFIXONLY	<p>Draws only an underline at the position of the character following the ampersand (&) prefix character. Does not draw any other characters in the string. For example,</p> <p>Example:</p> <p>input string: "A&bc&&d"\n</p> <p>normal: "<u>A</u>bc&<u>d</u>"</p> <p>DT_PREFIXONLY: " _ "</p> <p>Compare with DT_HIDEPREFIX and DT_NOPREFIX.</p>
DT_RIGHT	Aligns text to the right.
DT_RTLREADING	Layout in right-to-left reading order for bidirectional text when the font selected into the <i>hdc</i> is a Hebrew or Arabic font. The default reading order for all text is left-to-right.

DT_SINGLELINE	Displays text on a single line only. Carriage returns and line feeds do not break the line.
DT_TABSTOP	Sets tab stops. Bits 15-8 (high-order byte of the low-order word) of the <i>uFormat</i> parameter specify the number of characters for each tab. The default number of characters per tab is eight. The DT_CALCRECT, DT_EXTERNALLEADING, DT_INTERNAL, DT_NOCLIP, and DT_NOPREFIX values cannot be used with the DT_TABSTOP value.
DT_TOP	Justifies the text to the top of the rectangle.
DT_VCENTER	Centers text vertically. This value is used only with the DT_SINGLELINE value.
DT_WORDBREAK	Breaks words. Lines are automatically broken between words if a word would extend past the edge of the rectangle specified by the <i>lpRect</i> parameter. A carriage return-line feed sequence also breaks the line. If this is not specified, output is on one line.
DT_WORD_ELLIPSIS	Truncates any word that does not fit in the rectangle and adds ellipses. Compare with DT_END_ELLIPSIS and DT_PATH_ELLIPSIS.

Return value

If the function succeeds, the return value is the height of the text in logical units. If DT_VCENTER or DT_BOTTOM is specified, the return value is the offset from `lpRect->top` to the bottom of the drawn text

If the function fails, the return value is zero.

Remarks

The **DrawText** function uses the device context's selected font, text color, and background color to draw the text. Unless the DT_NOCLIP format is used, **DrawText** clips the text so that it does not appear outside the specified rectangle. Note that text with significant overhang may be clipped, for example, an initial "W" in the text string or text that is in italics. All formatting is assumed to have multiple lines unless the DT_SINGLELINE format is specified.

If the selected font is too large for the specified rectangle, the **DrawText** function does not attempt to substitute a smaller font.

The text alignment mode for the device context must include the TA_LEFT, TA_TOP, and TA_NOUPDATECP flags.

NOTE

The winuser.h header defines **DrawText** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-misc-l1-1-0 (introduced in Windows 8)

See also

[DrawTextEx](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GrayString](#)

[RECT](#)

[TabbedTextOut](#)

[TextOut](#)

DrawTextExA function (winuser.h)

4/21/2022 • 5 minutes to read • [Edit Online](#)

The **DrawTextEx** function draws formatted text in the specified rectangle.

Syntax

```
int DrawTextExA(
    [in]      HDC          hdc,
    [in, out] LPSTR       lpchText,
    [in]      int          cchText,
    [in, out] LPRECT      lprc,
    [in]      UINT         format,
    [in]      LPDRAWTEXTPARAMS  lpdtlp
);
```

Parameters

[in] *hdc*

A handle to the device context in which to draw.

[in, out] *lpchText*

A pointer to the string that contains the text to draw. If the *cchText* parameter is -1, the string must be null-terminated.

If *dwDTFormat* includes DT MODIFYSTRING, the function could add up to four additional characters to this string. The buffer containing the string should be large enough to accommodate these extra characters.

[in] *cchText*

The [length of the string](#) pointed to by *lpchText*. If *cchText* is -1, then the *lpchText* parameter is assumed to be a pointer to a null-terminated string and **DrawTextEx** computes the character count automatically.

[in, out] *lprc*

A pointer to a [RECT](#) structure that contains the rectangle, in logical coordinates, in which the text is to be formatted.

[in] *format*

The formatting options. This parameter can be one or more of the following values.

VALUE	MEANING
DT_BOTTOM	Justifies the text to the bottom of the rectangle. This value is used only with the DT SINGLELINE value.

DT_CALCRECT	Determines the width and height of the rectangle. If there are multiple lines of text, DrawTextEx uses the width of the rectangle pointed to by the <i>lprc</i> parameter and extends the base of the rectangle to bound the last line of text. If there is only one line of text, DrawTextEx modifies the right side of the rectangle so that it bounds the last character in the line. In either case, DrawTextEx returns the height of the formatted text, but does not draw the text.
DT_CENTER	Centers text horizontally in the rectangle.
DT_EDITCONTROL	Duplicates the text-displaying characteristics of a multiline edit control. Specifically, the average character width is calculated in the same manner as for an edit control, and the function does not display a partially visible last line.
DT_END_ELLIPSIS	For displayed text, replaces the end of a string with ellipses so that the result fits in the specified rectangle. Any word (not at the end of the string) that goes beyond the limits of the rectangle is truncated without ellipses. The string is not modified unless the DT_MODIFYSTRING flag is specified. Compare with DT_PATH_ELLIPSIS and DT_WORD_ELLIPSIS.
DT_EXPANDTABS	Expands tab characters. The default number of characters per tab is eight.
DT_EXTERNALLEADING	Includes the font external leading in line height. Normally, external leading is not included in the height of a line of text.
DT_HIDEPRFX	Ignores the ampersand (&) prefix character in the text. The letter that follows will not be underlined, but other mnemonic-prefix characters are still processed. Example: input string: "A&bc&&d" normal: "Abc&d" DT_HIDEPRFX: "Abc&d" Compare with DT_NOPREFIX and DT_PREFIXONLY.
DT_INTERNAL	Uses the system font to calculate text metrics.
DT_LEFT	Aligns text to the left.
DT_MODIFYSTRING	Modifies the specified string to match the displayed text. This value has no effect unless DT_END_ELLIPSIS or DT_PATH_ELLIPSIS is specified.
DT_NOCLIP	Draws without clipping. DrawTextEx is somewhat faster when DT_NOCLIP is used.

DT_NOFULLWIDTHCHARBREAK	Prevents a line break at a DBCS (double-wide character string), so that the line-breaking rule is equivalent to SBCS strings. For example, this can be used in Korean windows, for more readability of icon labels. This value has no effect unless DT_WORDBREAK is specified.
DT_NOPREFIX	Turns off processing of prefix characters. Normally, <code>DrawTextEx</code> interprets the ampersand (&) mnemonic-prefix character as a directive to underscore the character that follows, and the double-ampersand (&&) mnemonic-prefix characters as a directive to print a single ampersand. By specifying DT_NOPREFIX, this processing is turned off. Compare with DT_HIDEPREFIX and DT_PREFIXONLY
DT_PATH_ELLIPSIS	For displayed text, replaces characters in the middle of the string with ellipses so that the result fits in the specified rectangle. If the string contains backslash (\) characters, DT_PATH_ELLIPSIS preserves as much as possible of the text after the last backslash. The string is not modified unless the DT_MODIFYSTRING flag is specified. Compare with DT_END_ELLIPSIS and DT_WORD_ELLIPSIS.
DT_PREFIXONLY	Draws only an underline at the position of the character following the ampersand (&) prefix character. Does not draw any character in the string. Example: input string: "A&bc&&d" normal: "Ab <u>c</u> &d" PREFIXONLY: " _ " Compare with DT_NOPREFIX and DT_HIDEPREFIX.
DT_RIGHT	Aligns text to the right.
DT_RTLREADING	Layout in right-to-left reading order for bidirectional text when the font selected into the <i>hdc</i> is a Hebrew or Arabic font. The default reading order for all text is left-to-right.
DT_SINGLELINE	Displays text on a single line only. Carriage returns and line feeds do not break the line.
DT_TABSTOP	Sets tab stops. The DRAWTEXTPARAMS structure pointed to by the <i>lpDTPParams</i> parameter specifies the number of average character widths per tab stop.
DT_TOP	Justifies the text to the top of the rectangle.
DT_VCENTER	Centers text vertically. This value is used only with the DT_SINGLELINE value.

DT_WORDBREAK	Breaks words. Lines are automatically broken between words if a word extends past the edge of the rectangle specified by the <code>lprc</code> parameter. A carriage return-line feed sequence also breaks the line.
DT_WORD_ELLIPSIS	Truncates any word that does not fit in the rectangle and adds ellipses. Compare with DT_END_ELLIPSIS and DT_PATH_ELLIPSIS.

[in] `lpdtp`

A pointer to a [DRAWTEXTPARAMS](#) structure that specifies additional formatting options. This parameter can be `NULL`.

Return value

If the function succeeds, the return value is the text height in logical units. If DT_VCENTER or DT_BOTTOM is specified, the return value is the offset from `lprc->top` to the bottom of the drawn text

If the function fails, the return value is zero.

Remarks

The `DrawTextEx` function supports only fonts whose escapement and orientation are both zero.

The text alignment mode for the device context must include the TA_LEFT, TA_TOP, and TA_NOUPDATECP flags.

NOTE

The winuserh header defines `DrawTextEx` as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the `UNICODE` preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuserh (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-misc-l1-2-0 (introduced in Windows 8.1)

See also

DRAWTEXTPARAMS

[DrawText](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

DrawTextExW function (winuser.h)

4/21/2022 • 5 minutes to read • [Edit Online](#)

The **DrawTextEx** function draws formatted text in the specified rectangle.

Syntax

```
int DrawTextExW(
    [in]      HDC          hdc,
    [in, out] LPWSTR      lpchText,
    [in]      int          cchText,
    [in, out] LPRECT      lprc,
    [in]      UINT         format,
    [in]      LPDRAWTEXTPARAMS  lpdt
);
```

Parameters

[in] *hdc*

A handle to the device context in which to draw.

[in, out] *lpchText*

A pointer to the string that contains the text to draw. If the *cchText* parameter is -1, the string must be null-terminated.

If *dwDTFormat* includes DT MODIFYSTRING, the function could add up to four additional characters to this string. The buffer containing the string should be large enough to accommodate these extra characters.

[in] *cchText*

The [length of the string](#) pointed to by *lpchText*. If *cchText* is -1, then the *lpchText* parameter is assumed to be a pointer to a null-terminated string and **DrawTextEx** computes the character count automatically.

[in, out] *lprc*

A pointer to a [RECT](#) structure that contains the rectangle, in logical coordinates, in which the text is to be formatted.

[in] *format*

The formatting options. This parameter can be one or more of the following values.

VALUE	MEANING
DT_BOTTOM	Justifies the text to the bottom of the rectangle. This value is used only with the DT SINGLELINE value.

DT_CALCRECT	Determines the width and height of the rectangle. If there are multiple lines of text, DrawTextEx uses the width of the rectangle pointed to by the <i>lprc</i> parameter and extends the base of the rectangle to bound the last line of text. If there is only one line of text, DrawTextEx modifies the right side of the rectangle so that it bounds the last character in the line. In either case, DrawTextEx returns the height of the formatted text, but does not draw the text.
DT_CENTER	Centers text horizontally in the rectangle.
DT_EDITCONTROL	Duplicates the text-displaying characteristics of a multiline edit control. Specifically, the average character width is calculated in the same manner as for an edit control, and the function does not display a partially visible last line.
DT_END_ELLIPSIS	For displayed text, replaces the end of a string with ellipses so that the result fits in the specified rectangle. Any word (not at the end of the string) that goes beyond the limits of the rectangle is truncated without ellipses. The string is not modified unless the DT_MODIFYSTRING flag is specified. Compare with DT_PATH_ELLIPSIS and DT_WORD_ELLIPSIS.
DT_EXPANDTABS	Expands tab characters. The default number of characters per tab is eight.
DT_EXTERNALLEADING	Includes the font external leading in line height. Normally, external leading is not included in the height of a line of text.
DT_HIDEPRFX	Ignores the ampersand (&) prefix character in the text. The letter that follows will not be underlined, but other mnemonic-prefix characters are still processed. Example: input string: "A&bc&&d" normal: "Abc&d" DT_HIDEPRFX: "Abc&d" Compare with DT_NOPREFIX and DT_PREFIXONLY.
DT_INTERNAL	Uses the system font to calculate text metrics.
DT_LEFT	Aligns text to the left.
DT_MODIFYSTRING	Modifies the specified string to match the displayed text. This value has no effect unless DT_END_ELLIPSIS or DT_PATH_ELLIPSIS is specified.
DT_NOCLIP	Draws without clipping. DrawTextEx is somewhat faster when DT_NOCLIP is used.

DT_NOFULLWIDTHCHARBREAK	Prevents a line break at a DBCS (double-wide character string), so that the line-breaking rule is equivalent to SBCS strings. For example, this can be used in Korean windows, for more readability of icon labels. This value has no effect unless DT_WORDBREAK is specified.
DT_NOPREFIX	Turns off processing of prefix characters. Normally, <code>DrawTextEx</code> interprets the ampersand (&) mnemonic-prefix character as a directive to underscore the character that follows, and the double-ampersand (&&) mnemonic-prefix characters as a directive to print a single ampersand. By specifying DT_NOPREFIX, this processing is turned off. Compare with DT_HIDEPREFIX and DT_PREFIXONLY
DT_PATH_ELLIPSIS	For displayed text, replaces characters in the middle of the string with ellipses so that the result fits in the specified rectangle. If the string contains backslash (\) characters, DT_PATH_ELLIPSIS preserves as much as possible of the text after the last backslash. The string is not modified unless the DT_MODIFYSTRING flag is specified. Compare with DT_END_ELLIPSIS and DT_WORD_ELLIPSIS.
DT_PREFIXONLY	Draws only an underline at the position of the character following the ampersand (&) prefix character. Does not draw any character in the string. Example: input string: "A&bc&&d" normal: "Ab <u>c</u> &d" PREFIXONLY: " _ " Compare with DT_NOPREFIX and DT_HIDEPREFIX.
DT_RIGHT	Aligns text to the right.
DT_RTLREADING	Layout in right-to-left reading order for bidirectional text when the font selected into the <i>hdc</i> is a Hebrew or Arabic font. The default reading order for all text is left-to-right.
DT_SINGLELINE	Displays text on a single line only. Carriage returns and line feeds do not break the line.
DT_TABSTOP	Sets tab stops. The DRAWTEXTPARAMS structure pointed to by the <i>lpDTPParams</i> parameter specifies the number of average character widths per tab stop.
DT_TOP	Justifies the text to the top of the rectangle.
DT_VCENTER	Centers text vertically. This value is used only with the DT_SINGLELINE value.

DT_WORDBREAK	Breaks words. Lines are automatically broken between words if a word extends past the edge of the rectangle specified by the <code>lprc</code> parameter. A carriage return-line feed sequence also breaks the line.
DT_WORD_ELLIPSIS	Truncates any word that does not fit in the rectangle and adds ellipses. Compare with DT_END_ELLIPSIS and DT_PATH_ELLIPSIS.

[in] `lpdtp`

A pointer to a [DRAWTEXTPARAMS](#) structure that specifies additional formatting options. This parameter can be `NULL`.

Return value

If the function succeeds, the return value is the text height in logical units. If DT_VCENTER or DT_BOTTOM is specified, the return value is the offset from `lprc->top` to the bottom of the drawn text

If the function fails, the return value is zero.

Remarks

The `DrawTextEx` function supports only fonts whose escapement and orientation are both zero.

The text alignment mode for the device context must include the TA_LEFT, TA_TOP, and TA_NOUPDATECP flags.

NOTE

The winuserh header defines `DrawTextEx` as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the `UNICODE` preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuserh (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-misc-l1-2-0 (introduced in Windows 8.1)

See also

DRAWTEXTPARAMS

[DrawText](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

DRAWTEXTPARAMS structure (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **DRAWTEXTPARAMS** structure contains extended formatting options for the [DrawTextEx](#) function.

Syntax

```
typedef struct tagDRAWTEXTPARAMS {  
    UINT cbSize;  
    int iTabLength;  
    int iLeftMargin;  
    int iRightMargin;  
    UINT uiLengthDrawn;  
} DRAWTEXTPARAMS, *LPDRAWTEXTPARAMS;
```

Members

`cbSize`

The structure size, in bytes.

`iTabLength`

The size of each tab stop, in units equal to the average character width.

`iLeftMargin`

The left margin, in units equal to the average character width.

`iRightMargin`

The right margin, in units equal to the average character width.

`uiLengthDrawn`

Receives the number of characters processed by [DrawTextEx](#), including white-space characters. The number can be the [length of the string](#) or the index of the first line that falls below the drawing area. Note that [DrawTextEx](#) always processes the entire string if the DT_NOCLIP formatting flag is specified.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

[DrawTextEx](#)

[Font and Text Structures](#)

[Fonts and Text Overview](#)

DrawTextW function (winuser.h)

4/21/2022 • 6 minutes to read • [Edit Online](#)

The **DrawText** function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

To specify additional formatting options, use the [DrawTextEx](#) function.

Syntax

```
int DrawTextW(
    [in]      HDC      hdc,
    [in, out] LPCWSTR lpchText,
    [in]      int       cchText,
    [in, out] LPRECT   lprc,
    [in]      UINT      format
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in, out] `lpchText`

A pointer to the string that specifies the text to be drawn. If the *nCount* parameter is -1, the string must be null-terminated.

If *uFormat* includes DT_MODIFYSTRING, the function could add up to four additional characters to this string. The buffer containing the string should be large enough to accommodate these extra characters.

[in] `cchText`

The length, in characters, of the string. If *nCount* is -1, then the *lpchText* parameter is assumed to be a pointer to a null-terminated string and **DrawText** computes the character count automatically.

[in, out] `lprc`

A pointer to a [RECT](#) structure that contains the rectangle (in logical coordinates) in which the text is to be formatted.

[in] `format`

The method of formatting the text. This parameter can be one or more of the following values.

VALUE	MEANING
DT_BOTTOM	Justifies the text to the bottom of the rectangle. This value is used only with the DT_SINGLELINE value.

DT_CALCRECT	Determines the width and height of the rectangle. If there are multiple lines of text, DrawText uses the width of the rectangle pointed to by the <i>lpRect</i> parameter and extends the base of the rectangle to bound the last line of text. If the largest word is wider than the rectangle, the width is expanded. If the text is less than the width of the rectangle, the width is reduced. If there is only one line of text, DrawText modifies the right side of the rectangle so that it bounds the last character in the line. In either case, DrawText returns the height of the formatted text but does not draw the text.
DT_CENTER	Centers text horizontally in the rectangle.
DT_EDITCONTROL	Duplicates the text-displaying characteristics of a multiline edit control. Specifically, the average character width is calculated in the same manner as for an edit control, and the function does not display a partially visible last line.
DT_END_ELLIPSIS	<p>For displayed text, if the end of a string does not fit in the rectangle, it is truncated and ellipses are added. If a word that is not at the end of the string goes beyond the limits of the rectangle, it is truncated without ellipses.</p> <p>The string is not modified unless the DT_MODIFYSTRING flag is specified.</p> <p>Compare with DT_PATH_ELLIPSIS and DT_WORD_ELLIPSIS.</p>
DT_EXPANDTABS	Expands tab characters. The default number of characters per tab is eight. The DT_WORD_ELLIPSIS, DT_PATH_ELLIPSIS, and DT_END_ELLIPSIS values cannot be used with the DT_EXPANDTABS value.
DT_EXTERNALLEADING	Includes the font external leading in line height. Normally, external leading is not included in the height of a line of text.
DT_HIDEPREFIX	<p>Ignores the ampersand (&) prefix character in the text. The letter that follows will not be underlined, but other mnemonic-prefix characters are still processed.</p> <p>Example:</p> <p>input string: "A&bc&d"</p> <p>normal: "Abc&d"</p> <p>DT_HIDEPREFIX: "Abc&d"</p> <p>Compare with DT_NOPREFIX and DT_PREFIXONLY.</p>
DT_INTERNAL	Uses the system font to calculate text metrics.
DT_LEFT	Aligns text to the left.

DT_MODIFYSTRING	Modifies the specified string to match the displayed text. This value has no effect unless DT_END_ELLIPSIS or DT_PATH_ELLIPSIS is specified.
DT_NOCLIP	Draws without clipping. DrawText is somewhat faster when DT_NOCLIP is used.
DT_NOFULLWIDTHCHARBREAK	Prevents a line break at a DBCS (double-wide character string), so that the line breaking rule is equivalent to SBCS strings. For example, this can be used in Korean windows, for more readability of icon labels. This value has no effect unless DT_WORDBREAK is specified.
DT_NOPREFIX	<p>Turns off processing of prefix characters. Normally, DrawText interprets the mnemonic-prefix character & as a directive to underscore the character that follows, and the mnemonic-prefix characters && as a directive to print a single &. By specifying DT_NOPREFIX, this processing is turned off. For example,</p> <p>Example:</p> <p>input string: "A&bc&&d"</p> <p>normal: "<u>A</u>bc&<u>d</u>"</p> <p>DT_NOPREFIX: "A&bc&&d"</p> <p>Compare with DT_HIDEPREFIX and DT_PREFIXONLY.</p>
DT_PATH_ELLIPSIS	<p>For displayed text, replaces characters in the middle of the string with ellipses so that the result fits in the specified rectangle. If the string contains backslash (\) characters, DT_PATH_ELLIPSIS preserves as much as possible of the text after the last backslash.</p> <p>The string is not modified unless the DT MODIFYSTRING flag is specified.</p> <p>Compare with DT_END_ELLIPSIS and DT_WORD_ELLIPSIS.</p>
DT_PREFIXONLY	<p>Draws only an underline at the position of the character following the ampersand (&) prefix character. Does not draw any other characters in the string. For example,</p> <p>Example:</p> <p>input string: "A&bc&&d"\n</p> <p>normal: "<u>A</u>bc&<u>d</u>"</p> <p>DT_PREFIXONLY: " _ "</p> <p>Compare with DT_HIDEPREFIX and DT_NOPREFIX.</p>
DT_RIGHT	Aligns text to the right.
DT_RTLREADING	Layout in right-to-left reading order for bidirectional text when the font selected into the <i>hdc</i> is a Hebrew or Arabic font. The default reading order for all text is left-to-right.

DT_SINGLELINE	Displays text on a single line only. Carriage returns and line feeds do not break the line.
DT_TABSTOP	Sets tab stops. Bits 15-8 (high-order byte of the low-order word) of the <i>uFormat</i> parameter specify the number of characters for each tab. The default number of characters per tab is eight. The DT_CALCRECT, DT_EXTERNALLEADING, DT_INTERNAL, DT_NOCLIP, and DT_NOPREFIX values cannot be used with the DT_TABSTOP value.
DT_TOP	Justifies the text to the top of the rectangle.
DT_VCENTER	Centers text vertically. This value is used only with the DT_SINGLELINE value.
DT_WORDBREAK	Breaks words. Lines are automatically broken between words if a word would extend past the edge of the rectangle specified by the <i>lpRect</i> parameter. A carriage return-line feed sequence also breaks the line. If this is not specified, output is on one line.
DT_WORD_ELLIPSIS	Truncates any word that does not fit in the rectangle and adds ellipses. Compare with DT_END_ELLIPSIS and DT_PATH_ELLIPSIS.

Return value

If the function succeeds, the return value is the height of the text in logical units. If DT_VCENTER or DT_BOTTOM is specified, the return value is the offset from `lpRect->top` to the bottom of the drawn text

If the function fails, the return value is zero.

Remarks

The **DrawText** function uses the device context's selected font, text color, and background color to draw the text. Unless the DT_NOCLIP format is used, **DrawText** clips the text so that it does not appear outside the specified rectangle. Note that text with significant overhang may be clipped, for example, an initial "W" in the text string or text that is in italics. All formatting is assumed to have multiple lines unless the DT_SINGLELINE format is specified.

If the selected font is too large for the specified rectangle, the **DrawText** function does not attempt to substitute a smaller font.

The text alignment mode for the device context must include the TA_LEFT, TA_TOP, and TA_NOUPDATECP flags.

NOTE

The winuser.h header defines **DrawText** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-misc-l1-1-0 (introduced in Windows 8)

See also

[DrawTextEx](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GrayString](#)

[RECT](#)

[TabbedTextOut](#)

[TextOut](#)

EndPaint function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EndPaint** function marks the end of painting in the specified window. This function is required for each call to the [BeginPaint](#) function, but only after painting is complete.

Syntax

```
BOOL EndPaint(
    [in] HWND         hWnd,
    [in] const PAINTSTRUCT *lpPaint
);
```

Parameters

[in] hWnd

Handle to the window that has been repainted.

[in] lpPaint

Pointer to a [PAINTSTRUCT](#) structure that contains the painting information retrieved by [BeginPaint](#).

Return value

The return value is always nonzero.

Remarks

If the caret was hidden by [BeginPaint](#), [EndPaint](#) restores the caret to the screen.

[EndPaint](#) releases the display device context that [BeginPaint](#) retrieved.

Examples

For an example, see [Drawing in the Client Area](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib

DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[BeginPaint](#)

[PAINTSTRUCT](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

EnumDisplayDevicesA function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumDisplayDevices** function lets you obtain information about the display devices in the current session.

Syntax

```
BOOL EnumDisplayDevicesA(
    [in]    LPCSTR      lpDevice,
    [in]    DWORD       iDevNum,
    [out]   PDISPLAY_DEVICEA lpDisplayDevice,
    [in]    DWORD       dwFlags
);
```

Parameters

[in] lpDevice

A pointer to the device name. If **NULL**, function returns information for the display adapter(s) on the machine, based on *iDevNum*.

For more information, see Remarks.

[in] iDevNum

An index value that specifies the display device of interest.

The operating system identifies each display device in the current session with an index value. The index values are consecutive integers, starting at 0. If the current session has three display devices, for example, they are specified by the index values 0, 1, and 2.

[out] lpDisplayDevice

A pointer to a **DISPLAY_DEVICE** structure that receives information about the display device specified by *iDevNum*.

Before calling **EnumDisplayDevices**, you must initialize the **cb** member of **DISPLAY_DEVICE** to the size, in bytes, of **DISPLAY_DEVICE**.

[in] dwFlags

Set this flag to **EDD_GET_DEVICE_INTERFACE_NAME** (0x00000001) to retrieve the device interface name for **GUID_DEVINTERFACE_MONITOR**, which is registered by the operating system on a per monitor basis. The value is placed in the **DeviceID** member of the **DISPLAY_DEVICE** structure returned in *lpDisplayDevice*. The resulting device interface name can be used with **SetupAPI functions** and serves as a link between GDI monitor devices and **SetupAPI** monitor devices.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. The function fails if *iDevNum* is greater than the largest device index.

Remarks

To query all display devices in the current session, call this function in a loop, starting with *iDevNum* set to 0, and incrementing *iDevNum* until the function fails. To select all display devices in the desktop, use only the display devices that have the DISPLAY_DEVICE_ATTACHED_TO_DESKTOP flag in the [DISPLAY_DEVICE](#) structure.

To get information on the display adapter, call [EnumDisplayDevices](#) with *lpDevice* set to **NULL**. For example, [DISPLAY_DEVICE.DeviceString](#) contains the adapter name.

To obtain information on a display monitor, first call [EnumDisplayDevices](#) with *lpDevice* set to **NULL**. Then call [EnumDisplayDevices](#) with *lpDevice* set to [DISPLAY_DEVICE.DeviceName](#) from the first call to [EnumDisplayDevices](#) and with *iDevNum* set to zero. Then [DISPLAY_DEVICE.DeviceString](#) is the monitor name.

To query all monitor devices associated with an adapter, call [EnumDisplayDevices](#) in a loop with *lpDevice* set to the adapter name, *iDevNum* set to start at 0, and *iDevNum* set to increment until the function fails. Note that [DISPLAY_DEVICE.DeviceName](#) changes with each call for monitor information, so you must save the adapter name. The function fails when there are no more monitors for the adapter.

NOTE

The winuser.h header defines [EnumDisplayDevices](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[ChangeDisplaySettings](#)

[ChangeDisplaySettingsEx](#)

[CreateDC](#)

[DEVMODE](#)

[DISPLAY_DEVICE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplaySettings](#)

EnumDisplayDevicesW function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumDisplayDevices** function lets you obtain information about the display devices in the current session.

Syntax

```
BOOL EnumDisplayDevicesW(
    [in]    LPCWSTR      lpDevice,
    [in]    DWORD        iDevNum,
    [out]   PDISPLAY_DEVICEW lpDisplayDevice,
    [in]    DWORD        dwFlags
);
```

Parameters

[in] lpDevice

A pointer to the device name. If **NULL**, function returns information for the display adapter(s) on the machine, based on *iDevNum*.

For more information, see Remarks.

[in] iDevNum

An index value that specifies the display device of interest.

The operating system identifies each display device in the current session with an index value. The index values are consecutive integers, starting at 0. If the current session has three display devices, for example, they are specified by the index values 0, 1, and 2.

[out] lpDisplayDevice

A pointer to a **DISPLAY_DEVICE** structure that receives information about the display device specified by *iDevNum*.

Before calling **EnumDisplayDevices**, you must initialize the **cb** member of **DISPLAY_DEVICE** to the size, in bytes, of **DISPLAY_DEVICE**.

[in] dwFlags

Set this flag to **EDD_GET_DEVICE_INTERFACE_NAME** (0x00000001) to retrieve the device interface name for **GUID_DEVINTERFACE_MONITOR**, which is registered by the operating system on a per monitor basis. The value is placed in the **DeviceID** member of the **DISPLAY_DEVICE** structure returned in *lpDisplayDevice*. The resulting device interface name can be used with **SetupAPI functions** and serves as a link between GDI monitor devices and **SetupAPI** monitor devices.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. The function fails if *iDevNum* is greater than the largest device index.

Remarks

To query all display devices in the current session, call this function in a loop, starting with *iDevNum* set to 0, and incrementing *iDevNum* until the function fails. To select all display devices in the desktop, use only the display devices that have the DISPLAY_DEVICE_ATTACHED_TO_DESKTOP flag in the [DISPLAY_DEVICE](#) structure.

To get information on the display adapter, call [EnumDisplayDevices](#) with *lpDevice* set to **NULL**. For example, [DISPLAY_DEVICE.DeviceString](#) contains the adapter name.

To obtain information on a display monitor, first call [EnumDisplayDevices](#) with *lpDevice* set to **NULL**. Then call [EnumDisplayDevices](#) with *lpDevice* set to [DISPLAY_DEVICE.DeviceName](#) from the first call to [EnumDisplayDevices](#) and with *iDevNum* set to zero. Then [DISPLAY_DEVICE.DeviceString](#) is the monitor name.

To query all monitor devices associated with an adapter, call [EnumDisplayDevices](#) in a loop with *lpDevice* set to the adapter name, *iDevNum* set to start at 0, and *iDevNum* set to increment until the function fails. Note that [DISPLAY_DEVICE.DeviceName](#) changes with each call for monitor information, so you must save the adapter name. The function fails when there are no more monitors for the adapter.

NOTE

The winuser.h header defines [EnumDisplayDevices](#) as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[ChangeDisplaySettings](#)

[ChangeDisplaySettingsEx](#)

[CreateDC](#)

[DEVMODE](#)

[DISPLAY_DEVICE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplaySettings](#)

EnumDisplayMonitors function (winuser.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **EnumDisplayMonitors** function enumerates display monitors (including invisible pseudo-monitors associated with the mirroring drivers) that intersect a region formed by the intersection of a specified clipping rectangle and the visible region of a device context. **EnumDisplayMonitors** calls an application-defined [MonitorEnumProc](#) callback function once for each monitor that is enumerated. Note that [GetSystemMetrics](#) (SM_CMONITORS) counts only the display monitors.

Syntax

```
BOOL EnumDisplayMonitors(
    [in] HDC             hdc,
    [in] LPCRECT         lprcClip,
    [in] MONITORENUMPROC lpfnEnum,
    [in] LPARAM          dwData
);
```

Parameters

[in] hdc

A handle to a display device context that defines the visible region of interest.

If this parameter is **NULL**, the *hdcMonitor* parameter passed to the callback function will be **NULL**, and the visible region of interest is the virtual screen that encompasses all the displays on the desktop.

[in] lprcClip

A pointer to a [RECT](#) structure that specifies a clipping rectangle. The region of interest is the intersection of the clipping rectangle with the visible region specified by *hdc*.

If *hdc* is non-**NULL**, the coordinates of the clipping rectangle are relative to the origin of the *hdc*. If *hdc* is **NULL**, the coordinates are virtual-screen coordinates.

This parameter can be **NULL** if you don't want to clip the region specified by *hdc*.

[in] lpfnEnum

A pointer to a [MonitorEnumProc](#) application-defined callback function.

[in] dwData

Application-defined data that **EnumDisplayMonitors** passes directly to the [MonitorEnumProc](#) function.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

There are two reasons to call the **EnumDisplayMonitors** function:

- You want to draw optimally into a device context that spans several display monitors, and the monitors have different color formats.
- You want to obtain a handle and position rectangle for one or more display monitors.

To determine whether all the display monitors in a system share the same color format, call [GetSystemMetrics](#) (SM_SAMEDISPLAYFORMAT).

You do not need to use the **EnumDisplayMonitors** function when a window spans display monitors that have different color formats. You can continue to paint under the assumption that the entire screen has the color properties of the primary monitor. Your windows will look fine. **EnumDisplayMonitors** just lets you make them look better.

Setting the *hdc* parameter to **NULL** lets you use the **EnumDisplayMonitors** function to obtain a handle and position rectangle for one or more display monitors. The following table shows how the four combinations of **NULL** and non-**NULL** *hdc* and *lprcClip* values affect the behavior of the **EnumDisplayMonitors** function.

<i>HDC</i>	<i>LPRCRECT</i>	ENUMDISPLAYMONITORS BEHAVIOR
NULL	NULL	Enumerates all display monitors. The callback function receives a NULL HDC.
NULL	non- NULL	Enumerates all display monitors that intersect the clipping rectangle. Use virtual screen coordinates for the clipping rectangle. The callback function receives a NULL HDC.
non- NULL	NULL	Enumerates all display monitors that intersect the visible region of the device context. The callback function receives a handle to a DC for the specific display monitor.
non- NULL	non- NULL	Enumerates all display monitors that intersect the visible region of the device context and the clipping rectangle. Use device context coordinates for the clipping rectangle. The callback function receives a handle to a DC for the specific display monitor.

Examples

To paint in response to a WM_PAINT message, using the capabilities of each monitor, you can use code like this in a window procedure:

```
case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    EnumDisplayMonitors(hdc, NULL, MyPaintEnumProc, 0);
    EndPaint(hwnd, &ps);
```

To paint the top half of a window using the capabilities of each monitor, you can use code like this:

```
GetClientRect(hwnd, &rc);
rc.bottom = (rc.bottom - rc.top) / 2;
hdc = GetDC(hwnd);
EnumDisplayMonitors(hdc, &rc, MyPaintEnumProc, 0);
ReleaseDC(hwnd, hdc);
```

To paint the entire virtual screen optimally for each display monitor, you can use code like this:

```
hdc = GetDC(NULL);
EnumDisplayMonitors(hdc, NULL, MyPaintScreenEnumProc, 0);
ReleaseDC(NULL, hdc);
```

To retrieve information about all of the display monitors, use code like this:

```
EnumDisplayMonitors(NULL, NULL, MyInfoEnumProc, 0);
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[GetSystemMetrics](#)

[MonitorEnumProc](#)

[Multiple Display Monitors Functions](#)

[Multiple Display Monitors Overview](#)

EnumDisplaySettingsA function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumDisplaySettings** function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.

Note Apps that you design to target Windows 8 and later can no longer query or set display modes that are less than 32 bits per pixel (bpp); these operations will fail. These apps have a [compatibility manifest](#) that targets Windows 8. Windows 8 still supports 8-bit and 16-bit color modes for desktop apps that were built without a Windows 8 manifest; Windows 8 emulates these modes but still runs in 32-bit color mode.

Syntax

```
BOOL EnumDisplaySettingsA(
    [in]  LPCSTR    lpszDeviceName,
    [in]  DWORD     iModeNum,
    [out] DEVMODEA *lpDevMode
);
```

Parameters

[in] lpszDeviceName

A pointer to a null-terminated string that specifies the display device about whose graphics mode the function will obtain information.

This parameter is either **NULL** or a [DISPLAY_DEVICE.DeviceName](#) returned from [EnumDisplayDevices](#). A **NULL** value specifies the current display device on the computer on which the calling thread is running.

[in] iModeNum

The type of information to be retrieved. This value can be a graphics mode index or one of the following values.

VALUE	MEANING
ENUM_CURRENT_SETTINGS	Retrieve the current settings for the display device.
ENUM_REGISTRY_SETTINGS	Retrieve the settings for the display device that are currently stored in the registry.

Graphics mode indexes start at zero. To obtain information for all of a display device's graphics modes, make a series of calls to **EnumDisplaySettings**, as follows: Set *iModeNum* to zero for the first call, and increment *iModeNum* by one for each subsequent call. Continue calling the function until the return value is zero.

When you call **EnumDisplaySettings** with *iModeNum* set to zero, the operating system initializes and caches

information about the display device. When you call **EnumDisplaySettings** with *iModeNum* set to a nonzero value, the function returns the information that was cached the last time the function was called with *iModeNum* set to zero.

[out] *lpDevMode*

A pointer to a **DEVMODE** structure into which the function stores information about the specified graphics mode. Before calling **EnumDisplaySettings**, set the **dmSize** member to `sizeof(DEVMODE)`, and set the **dmDriverExtra** member to indicate the size, in bytes, of the additional space available to receive private driver data.

The **EnumDisplaySettings** function sets values for the following five **DEVMODE** members:

- **dmBitsPerPel**
- **dmPelsWidth**
- **dmPelsHeight**
- **dmDisplayFlags**
- **dmDisplayFrequency**

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The function fails if *iModeNum* is greater than the index of the display device's last graphics mode. As noted in the description of the *iModeNum* parameter, you can use this behavior to enumerate all of a display device's graphics modes.

DPI Virtualization

This API does not participate in DPI virtualization. The output given is always in terms of physical pixels, and is not related to the calling context.

NOTE

The winuser.h header defines **EnumDisplaySettings** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-0 (introduced in Windows 8)

See also

[ChangeDisplaySettings](#)

[ChangeDisplaySettingsEx](#)

[CreateDC](#)

[CreateDesktop](#)

[DEVMODE](#)

[DISPLAY_DEVICE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

EnumDisplaySettingsExA function (winuser.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **EnumDisplaySettingsEx** function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.

This function differs from [EnumDisplaySettings](#) in that there is a *dwFlags* parameter.

Note Apps that you design to target Windows 8 and later can no longer query or set display modes that are less than 32 bits per pixel (bpp); these operations will fail. These apps have a [compatibility manifest](#) that targets Windows 8. Windows 8 still supports 8-bit and 16-bit color modes for desktop apps that were built without a Windows 8 manifest; Windows 8 emulates these modes but still runs in 32-bit color mode.

Syntax

```
BOOL EnumDisplaySettingsExA(
    [in]  LPCSTR  lpszDeviceName,
    [in]  DWORD    iModeNum,
    [out] DEVMODEA *lpDevMode,
    [in]  DWORD    dwFlags
);
```

Parameters

[in] lpszDeviceName

A pointer to a null-terminated string that specifies the display device about which graphics mode the function will obtain information.

This parameter is either **NULL** or a [DISPLAY_DEVICE.DeviceName](#) returned from [EnumDisplayDevices](#). A **NULL** value specifies the current display device on the computer that the calling thread is running on.

[in] iModeNum

Indicates the type of information to be retrieved. This value can be a graphics mode index or one of the following values.

VALUE	MEANING
ENUM_CURRENT_SETTINGS	Retrieve the current settings for the display device.
ENUM_REGISTRY_SETTINGS	Retrieve the settings for the display device that are currently stored in the registry.

Graphics mode indexes start at zero. To obtain information for all of a display device's graphics modes, make a series of calls to [EnumDisplaySettingsEx](#), as follows: Set *iModeNum* to zero for the first call, and increment

iModeNum by one for each subsequent call. Continue calling the function until the return value is zero.

When you call **EnumDisplaySettingsEx** with *iModeNum* set to zero, the operating system initializes and caches information about the display device. When you call **EnumDisplaySettingsEx** with *iModeNum* set to a nonzero value, the function returns the information that was cached the last time the function was called with *iModeNum* set to zero.

[out] **lpDevMode**

A pointer to a **DEVMODE** structure into which the function stores information about the specified graphics mode. Before calling **EnumDisplaySettingsEx**, set the **dmSize** member to **sizeof (DEVMODE)**, and set the **dmDriverExtra** member to indicate the size, in bytes, of the additional space available to receive private driver data.

The **EnumDisplaySettingsEx** function will populate the **dmFields** member of the **lpDevMode** and one or more other members of the **DEVMODE** structure. To determine which members were set by the call to **EnumDisplaySettingsEx**, inspect the *dmFields* bitmask. Some of the fields typically populated by this function include:

- **dmBitsPerPel**
- **dmPelsWidth**
- **dmPelsHeight**
- **dmDisplayFlags**
- **dmDisplayFrequency**
- **dmPosition**
- **dmDisplayOrientation**

[in] **dwFlags**

This parameter can be the following value.

VALUE	MEANING
EDS_RAWMODE	If set, the function will return all graphics modes reported by the adapter driver, regardless of monitor capabilities. Otherwise, it will only return modes that are compatible with current monitors.
EDS_ROTATEDMODE	If set, the function will return graphics modes in all orientations. Otherwise, it will only return modes that have the same orientation as the one currently set for the requested display.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The function fails if *iModeNum* is greater than the index of the display device's last graphics mode. As noted in the description of the *iModeNum* parameter, you can use this behavior to enumerate all of a display device's graphics modes.

DPI Virtualization

This API does not participate in DPI virtualization. The output given is always in terms of physical pixels, and is not related to the calling context.

NOTE

The winuser.h header defines `EnumDisplaySettingsEx` as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the `UNICODE` preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[ChangeDisplaySettings](#)

[ChangeDisplaySettingsEx](#)

[CreateDC](#)

[CreateDesktop](#)

[DEVMODE](#)

[DISPLAY_DEVICE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

[EnumDisplaySettings](#)

EnumDisplaySettingsExW function (winuser.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **EnumDisplaySettingsEx** function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.

This function differs from [EnumDisplaySettings](#) in that there is a *dwFlags* parameter.

Note Apps that you design to target Windows 8 and later can no longer query or set display modes that are less than 32 bits per pixel (bpp); these operations will fail. These apps have a [compatibility manifest](#) that targets Windows 8. Windows 8 still supports 8-bit and 16-bit color modes for desktop apps that were built without a Windows 8 manifest; Windows 8 emulates these modes but still runs in 32-bit color mode.

Syntax

```
BOOL EnumDisplaySettingsExW(
    [in]  LPCWSTR  lpszDeviceName,
    [in]  DWORD     iModeNum,
    [out] DEVMODEW *lpDevMode,
    [in]  DWORD     dwFlags
);
```

Parameters

[in] lpszDeviceName

A pointer to a null-terminated string that specifies the display device about which graphics mode the function will obtain information.

This parameter is either **NULL** or a [DISPLAY_DEVICE](#).**DeviceName** returned from [EnumDisplayDevices](#). A **NULL** value specifies the current display device on the computer that the calling thread is running on.

[in] iModeNum

Indicates the type of information to be retrieved. This value can be a graphics mode index or one of the following values.

VALUE	MEANING
ENUM_CURRENT_SETTINGS	Retrieve the current settings for the display device.
ENUM_REGISTRY_SETTINGS	Retrieve the settings for the display device that are currently stored in the registry.

Graphics mode indexes start at zero. To obtain information for all of a display device's graphics modes, make a series of calls to **EnumDisplaySettingsEx**, as follows: Set *iModeNum* to zero for the first call, and increment

iModeNum by one for each subsequent call. Continue calling the function until the return value is zero.

When you call **EnumDisplaySettingsEx** with *iModeNum* set to zero, the operating system initializes and caches information about the display device. When you call **EnumDisplaySettingsEx** with *iModeNum* set to a nonzero value, the function returns the information that was cached the last time the function was called with *iModeNum* set to zero.

[out] **lpDevMode**

A pointer to a **DEVMODE** structure into which the function stores information about the specified graphics mode. Before calling **EnumDisplaySettingsEx**, set the **dmSize** member to **sizeof (DEVMODE)**, and set the **dmDriverExtra** member to indicate the size, in bytes, of the additional space available to receive private driver data.

The **EnumDisplaySettingsEx** function will populate the **dmFields** member of the **lpDevMode** and one or more other members of the **DEVMODE** structure. To determine which members were set by the call to **EnumDisplaySettingsEx**, inspect the *dmFields* bitmask. Some of the fields typically populated by this function include:

- **dmBitsPerPel**
- **dmPelsWidth**
- **dmPelsHeight**
- **dmDisplayFlags**
- **dmDisplayFrequency**
- **dmPosition**
- **dmDisplayOrientation**

[in] **dwFlags**

This parameter can be the following value.

VALUE	MEANING
EDS_RAWMODE	If set, the function will return all graphics modes reported by the adapter driver, regardless of monitor capabilities. Otherwise, it will only return modes that are compatible with current monitors.
EDS_ROTATEDMODE	If set, the function will return graphics modes in all orientations. Otherwise, it will only return modes that have the same orientation as the one currently set for the requested display.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The function fails if *iModeNum* is greater than the index of the display device's last graphics mode. As noted in the description of the *iModeNum* parameter, you can use this behavior to enumerate all of a display device's graphics modes.

DPI Virtualization

This API does not participate in DPI virtualization. The output given is always in terms of physical pixels, and is not related to the calling context.

NOTE

The winuser.h header defines `EnumDisplaySettingsEx` as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the `UNICODE` preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[ChangeDisplaySettings](#)

[ChangeDisplaySettingsEx](#)

[CreateDC](#)

[CreateDesktop](#)

[DEVMODE](#)

[DISPLAY_DEVICE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

[EnumDisplaySettings](#)

EnumDisplaySettingsW function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EnumDisplaySettings** function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.

Note Apps that you design to target Windows 8 and later can no longer query or set display modes that are less than 32 bits per pixel (bpp); these operations will fail. These apps have a [compatibility manifest](#) that targets Windows 8. Windows 8 still supports 8-bit and 16-bit color modes for desktop apps that were built without a Windows 8 manifest; Windows 8 emulates these modes but still runs in 32-bit color mode.

Syntax

```
BOOL EnumDisplaySettingsW(
    [in]  LPCWSTR lpszDeviceName,
    [in]  DWORD    iModeNum,
    [out] DEVMODEW *lpDevMode
);
```

Parameters

[in] lpszDeviceName

A pointer to a null-terminated string that specifies the display device about whose graphics mode the function will obtain information.

This parameter is either **NULL** or a [DISPLAY_DEVICE.DeviceName](#) returned from [EnumDisplayDevices](#). A **NULL** value specifies the current display device on the computer on which the calling thread is running.

[in] iModeNum

The type of information to be retrieved. This value can be a graphics mode index or one of the following values.

VALUE	MEANING
ENUM_CURRENT_SETTINGS	Retrieve the current settings for the display device.
ENUM_REGISTRY_SETTINGS	Retrieve the settings for the display device that are currently stored in the registry.

Graphics mode indexes start at zero. To obtain information for all of a display device's graphics modes, make a series of calls to **EnumDisplaySettings**, as follows: Set *iModeNum* to zero for the first call, and increment *iModeNum* by one for each subsequent call. Continue calling the function until the return value is zero.

When you call **EnumDisplaySettings** with *iModeNum* set to zero, the operating system initializes and caches

information about the display device. When you call **EnumDisplaySettings** with *iModeNum* set to a nonzero value, the function returns the information that was cached the last time the function was called with *iModeNum* set to zero.

[out] *lpDevMode*

A pointer to a **DEVMODE** structure into which the function stores information about the specified graphics mode. Before calling **EnumDisplaySettings**, set the **dmSize** member to `sizeof(DEVMODE)`, and set the **dmDriverExtra** member to indicate the size, in bytes, of the additional space available to receive private driver data.

The **EnumDisplaySettings** function sets values for the following five **DEVMODE** members:

- **dmBitsPerPel**
- **dmPelsWidth**
- **dmPelsHeight**
- **dmDisplayFlags**
- **dmDisplayFrequency**

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The function fails if *iModeNum* is greater than the index of the display device's last graphics mode. As noted in the description of the *iModeNum* parameter, you can use this behavior to enumerate all of a display device's graphics modes.

DPI Virtualization

This API does not participate in DPI virtualization. The output given is always in terms of physical pixels, and is not related to the calling context.

NOTE

The winuser.h header defines **EnumDisplaySettings** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-0 (introduced in Windows 8)

See also

[ChangeDisplaySettings](#)

[ChangeDisplaySettingsEx](#)

[CreateDC](#)

[CreateDesktop](#)

[DEVMODE](#)

[DISPLAY_DEVICE](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[EnumDisplayDevices](#)

EqualRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **EqualRect** function determines whether the two specified rectangles are equal by comparing the coordinates of their upper-left and lower-right corners.

Syntax

```
BOOL EqualRect(
    [in] const RECT *lprc1,
    [in] const RECT *lprc2
);
```

Parameters

[in] lprc1

Pointer to a [RECT](#) structure that contains the logical coordinates of the first rectangle.

[in] lprc2

Pointer to a [RECT](#) structure that contains the logical coordinates of the second rectangle.

Return value

If the two rectangles are identical, the return value is nonzero.

If the two rectangles are not identical, the return value is zero.

Remarks

The **EqualRect** function does not treat empty rectangles as equal if their coordinates are different.

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib

DLL	User32.dll
-----	------------

See also

[IsRectEmpty](#)

[PtInRect](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

ExcludeUpdateRgn function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ExcludeUpdateRgn** function prevents drawing within invalid areas of a window by excluding an updated region in the window from a clipping region.

Syntax

```
int ExcludeUpdateRgn(
    [in] HDC  hdc,
    [in] HWND hWnd
);
```

Parameters

[in] `hDC`

Handle to the device context associated with the clipping region.

[in] `hWnd`

Handle to the window to update.

Return value

The return value specifies the complexity of the excluded region; it can be any one of the following values.

VALUE	MEANING
COMPLEXREGION	Region consists of more than one rectangle.
ERROR	An error occurred.
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

Library	User32.lib
DLL	User32.dll

See also

[BeginPaint](#)

[GetUpdateRect](#)

[GetUpdateRgn](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[UpdateWindow](#)

FillRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **FillRect** function fills a rectangle by using the specified brush. This function includes the left and top borders, but excludes the right and bottom borders of the rectangle.

Syntax

```
int FillRect(
    [in] HDC      hDC,
    [in] const RECT *lprc,
    [in] HBRUSH   hbr
);
```

Parameters

[in] `hDC`

A handle to the device context.

[in] `lprc`

A pointer to a [RECT](#) structure that contains the logical coordinates of the rectangle to be filled.

[in] `hbr`

A handle to the brush used to fill the rectangle.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The brush identified by the `hbr` parameter may be either a handle to a logical brush or a color value. If specifying a handle to a logical brush, call one of the following functions to obtain the handle: [CreateHatchBrush](#), [CreatePatternBrush](#), or [CreateSolidBrush](#). Additionally, you may retrieve a handle to one of the stock brushes by using the [GetStockObject](#) function. If specifying a color value for the `hbr` parameter, it must be one of the standard system colors (the value 1 must be added to the chosen color). For example:

```
FillRect(hdc, &rect, (HBRUSH) (COLOR_WINDOW+1));
```

For a list of all the standard system colors, see [GetSysColor](#).

When filling the specified rectangle, **FillRect** does not include the rectangle's right and bottom sides. GDI fills a rectangle up to, but not including, the right column and bottom row, regardless of the current mapping mode.

Examples

For an example, see [Using Rectangles](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-0 (introduced in Windows 8)

See also

[CreateHatchBrush](#)

[CreatePatternBrush](#)

[CreateSolidBrush](#)

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

[GetStockObject](#)

[RECT](#)

FrameRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **FrameRect** function draws a border around the specified rectangle by using the specified brush. The width and height of the border are always one logical unit.

Syntax

```
int FrameRect(
    [in] HDC      hDC,
    [in] const RECT *lprc,
    [in] HBRUSH   hbr
);
```

Parameters

[in] **hDC**

A handle to the device context in which the border is drawn.

[in] **lprc**

A pointer to a [RECT](#) structure that contains the logical coordinates of the upper-left and lower-right corners of the rectangle.

[in] **hbr**

A handle to the brush used to draw the border.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The brush identified by the *hbr* parameter must have been created by using the [CreateHatchBrush](#), [CreatePatternBrush](#), or [CreateSolidBrush](#) function, or retrieved by using the [GetStockObject](#) function.

If the **bottom** member of the [RECT](#) structure is less than the **top** member, or if the **right** member is less than the **left** member, the function does not draw the rectangle.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-1 (introduced in Windows 8.1)

See also

[CreateHatchBrush](#)

[CreatePatternBrush](#)

[CreateSolidBrush](#)

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

[GetStockObject](#)

[RECT](#)

GetDC function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetDC** function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen. You can use the returned handle in subsequent GDI functions to draw in the DC. The device context is an opaque data structure, whose values are used internally by GDI.

The [GetDCEx](#) function is an extension to **GetDC**, which gives an application more control over how and whether clipping occurs in the client area.

Syntax

```
HDC GetDC(  
    [in] HWND hWnd  
)
```

Parameters

[in] hWnd

A handle to the window whose DC is to be retrieved. If this value is **NULL**, **GetDC** retrieves the DC for the entire screen.

Return value

If the function succeeds, the return value is a handle to the DC for the specified window's client area.

If the function fails, the return value is **NULL**.

Remarks

The **GetDC** function retrieves a common, class, or private DC depending on the class style of the specified window. For class and private DCs, **GetDC** leaves the previously assigned attributes unchanged. However, for common DCs, **GetDC** assigns default attributes to the DC each time it is retrieved. For example, the default font is **System**, which is a bitmap font. Because of this, the handle to a common DC returned by **GetDC** does not tell you what font, color, or brush was used when the window was drawn. To determine the font, call [GetTextFace](#).

Note that the handle to the DC can only be used by a single thread at any one time.

After painting with a common DC, the [ReleaseDC](#) function must be called to release the DC. Class and private DCs do not have to be released. **ReleaseDC** must be called from the same thread that called **GetDC**. The number of DCs is limited only by available memory.

Examples

For an example, see [Drawing with the Mouse](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetDCEx](#)

[GetTextFace](#)

[GetWindowDC](#)

[ReleaseDC](#)

[WindowFromDC](#)

GetDCEx function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetDCEx** function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen. You can use the returned handle in subsequent GDI functions to draw in the DC. The device context is an opaque data structure, whose values are used internally by GDI.

This function is an extension to the [GetDC](#) function, which gives an application more control over how and whether clipping occurs in the client area.

Syntax

```
HDC GetDCEx(
    [in] HWND hWnd,
    [in] HRGN hrgnClip,
    [in] DWORD flags
);
```

Parameters

[in] *hWnd*

A handle to the window whose DC is to be retrieved. If this value is **NULL**, **GetDCEx** retrieves the DC for the entire screen.

[in] *hrgnClip*

A clipping region that may be combined with the visible region of the DC. If the value of *flags* is **DCX_INTERSECTRGN** or **DCX_EXCLUDERGN**, then the operating system assumes ownership of the region and will automatically delete it when it is no longer needed. In this case, the application should not use or delete the region after a successful call to **GetDCEx**.

[in] *flags*

Specifies how the DC is created. This parameter can be one or more of the following values.

VALUE	MEANING
DCX_WINDOW	Returns a DC that corresponds to the window rectangle rather than the client rectangle.
DCX_CACHE	Returns a DC from the cache, rather than the OWNDC or CLASSDC window. Essentially overrides CS_OWNDC and CS_CLASSDC .
DCX_PARENTCLIP	Uses the visible region of the parent window. The parent's WS_CLIPCHILDREN and CS_PARENTDC style bits are ignored. The origin is set to the upper-left corner of the window identified by <i>hWnd</i> .
DCX_CLIPSIBLINGS	Excludes the visible regions of all sibling windows above the window identified by <i>hWnd</i> .

DCX_CLIPCHILDREN	Excludes the visible regions of all child windows below the window identified by <i>hWnd</i> .
DCX_NORESETATTRS	This flag is ignored.
DCX_LOCKWINDOWUPDATE	Allows drawing even if there is a LockWindowUpdate call in effect that would otherwise exclude this window. Used for drawing during tracking.
DCX_EXCLUDERGN	The clipping region identified by <i>hrgnClip</i> is excluded from the visible region of the returned DC.
DCX_INTERSECTRGN	The clipping region identified by <i>hrgnClip</i> is intersected with the visible region of the returned DC.
DCX_INTERSECTUPDATE	Reserved; do not use.
DCX_VALIDATE	Reserved; do not use.

Return value

If the function succeeds, the return value is the handle to the DC for the specified window.

If the function fails, the return value is **NULL**. An invalid value for the *hWnd* parameter will cause the function to fail.

Remarks

Unless the display DC belongs to a window class, the [ReleaseDC](#) function must be called to release the DC after painting. Also, [ReleaseDC](#) must be called from the same thread that called [GetDCE](#). The number of DCs is limited only by available memory.

The function returns a handle to a DC that belongs to the window's class if CS_CLASSDC, CS_OWNDC or CS_PARENTDC was specified as a style in the [WNDCLASS](#) structure when the class was registered.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib

DLL	User32.dll
-----	------------

See also

[BeginPaint](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetWindowDC](#)

[ReleaseDC](#)

[WNDCLASS](#)

GetMonitorInfoA function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetMonitorInfo** function retrieves information about a display monitor.

Syntax

```
BOOL GetMonitorInfoA(
    [in] HMONITOR     hMonitor,
    [out] LPMONITORINFO lpmi
);
```

Parameters

[in] hMonitor

A handle to the display monitor of interest.

[out] lpmi

A pointer to a [MONITORINFO](#) or [MONITORINFOEX](#) structure that receives information about the specified display monitor.

You must set the **cbSize** member of the structure to `sizeof(MONITORINFO)` or `sizeof(MONITORINFOEX)` before calling the **GetMonitorInfo** function. Doing so lets the function determine the type of structure you are passing to it.

The [MONITORINFOEX](#) structure is a superset of the [MONITORINFO](#) structure. It has one additional member: a string that contains a name for the display monitor. Most applications have no use for a display monitor name, and so can save some bytes by using a [MONITORINFO](#) structure.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

NOTE

The winuser.h header defines **GetMonitorInfo** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[MONITORINFO](#)

[MONITORINFOEX](#)

[Multiple Display Monitors Functions](#)

[Multiple Display Monitors Overview](#)

GetMonitorInfoW function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetMonitorInfo** function retrieves information about a display monitor.

Syntax

```
BOOL GetMonitorInfoW(
    [in] HMONITOR     hMonitor,
    [out] LPMONITORINFO lpmi
);
```

Parameters

[in] hMonitor

A handle to the display monitor of interest.

[out] lpmi

A pointer to a [MONITORINFO](#) or [MONITORINFOEX](#) structure that receives information about the specified display monitor.

You must set the **cbSize** member of the structure to `sizeof(MONITORINFO)` or `sizeof(MONITORINFOEX)` before calling the **GetMonitorInfo** function. Doing so lets the function determine the type of structure you are passing to it.

The [MONITORINFOEX](#) structure is a superset of the [MONITORINFO](#) structure. It has one additional member: a string that contains a name for the display monitor. Most applications have no use for a display monitor name, and so can save some bytes by using a [MONITORINFO](#) structure.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

NOTE

The winuser.h header defines **GetMonitorInfo** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[MONITORINFO](#)

[MONITORINFOEX](#)

[Multiple Display Monitors Functions](#)

[Multiple Display Monitors Overview](#)

GetSysColorBrush function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetSysColorBrush** function retrieves a handle identifying a logical brush that corresponds to the specified color index.

Syntax

```
HBRUSH GetSysColorBrush(  
    [in] int nIndex  
>);
```

Parameters

[in] *nIndex*

A color index. This value corresponds to the color used to paint one of the window elements. See [GetSysColor](#) for system color index values.

Return value

The return value identifies a logical brush if the *nIndex* parameter is supported by the current platform. Otherwise, it returns **NULL**.

Remarks

A brush is a bitmap that the system uses to paint the interiors of filled shapes. An application can retrieve the current system colors by calling the [GetSysColor](#) function. An application can set the current system colors by calling the [SetSysColors](#) function.

An application must not register a window class for a window using a system brush. To register a window class with a system color, see the documentation of the **hbrBackground** member of the [WNDCLASS](#) or [WNDCLASSEX](#) structures.

System color brushes track changes in system colors. In other words, when the user changes a system color, the associated system color brush automatically changes to the new color.

To paint with a system color brush, an application should use **GetSysColorBrush** (*nIndex*) instead of **CreateSolidBrush** ([GetSysColor](#) (*nIndex*)), because **GetSysColorBrush** returns a cached brush instead of allocating a new one.

System color brushes are owned by the system so you don't need to destroy them. Although you don't need to delete the logical brush that **GetSysColorBrush** returns, no harm occurs by calling [DeleteObject](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-1 (introduced in Windows 8.1)

See also

[Brush Functions](#)

[Brushes Overview](#)

[CreateSolidBrush](#)

[GetSysColor](#)

[SetSysColors](#)

[WNDCLASS](#)

[WNDCLASSEX](#)

GetTabbedTextExtentA function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTabbedTextExtent** function computes the width and height of a character string. If the string contains one or more tab characters, the width of the string is based upon the specified tab stops. The **GetTabbedTextExtent** function uses the currently selected font to compute the dimensions of the string.

Syntax

```
DWORD GetTabbedTextExtentA(
    [in] HDC      hdc,
    [in] LPCSTR   lpString,
    [in] int      chCount,
    [in] int      nTabPositions,
    [in] const INT *lpnTabStopPositions
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpString`

A pointer to a character string.

[in] `chCount`

The length of the text string. For the ANSI function it is a BYTE count and for the Unicode function it is a WORD count. Note that for the ANSI function, characters in SBCS code pages take one byte each, while most characters in DBCS code pages take two bytes; for the Unicode function, most currently defined Unicode characters (those in the Basic Multilingual Plane (BMP)) are one WORD while Unicode surrogates are two WORDs.

[in] `nTabPositions`

The number of tab-stop positions in the array pointed to by the `lpnTabStopPositions` parameter.

[in] `lpnTabStopPositions`

A pointer to an array containing the tab-stop positions, in device units. The tab stops must be sorted in increasing order; the smallest x-value should be the first item in the array.

Return value

If the function succeeds, the return value is the dimensions of the string in logical units. The height is in the high-order word and the width is in the low-order word.

If the function fails, the return value is 0. **GetTabbedTextExtent** will fail if `hDC` is invalid and if `nTabPositions` is less than 0.

Remarks

The current clipping region does not affect the width and height returned by the **GetTabbedTextExtent** function.

Because some devices do not place characters in regular cell arrays (that is, they kern the characters), the sum of the extents of the characters in a string may not be equal to the extent of the string.

If the *nTabPositions* parameter is zero and the *lpnTabStopPositions* parameter is **NULL**, tabs are expanded to eight times the average character width.

If *nTabPositions* is 1, the tab stops are separated by the distance specified by the first value in the array to which *lpnTabStopPositions* points.

NOTE

The winuser.h header defines GetTabbedTextExtent as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[TabbedTextOut](#)

GetTabbedTextExtentW function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetTabbedTextExtent** function computes the width and height of a character string. If the string contains one or more tab characters, the width of the string is based upon the specified tab stops. The **GetTabbedTextExtent** function uses the currently selected font to compute the dimensions of the string.

Syntax

```
DWORD GetTabbedTextExtentW(
    [in] HDC      hdc,
    [in] LPCWSTR  lpString,
    [in] int       chCount,
    [in] int       nTabPositions,
    [in] const INT *lpnTabStopPositions
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `lpString`

A pointer to a character string.

[in] `chCount`

The length of the text string. For the ANSI function it is a BYTE count and for the Unicode function it is a WORD count. Note that for the ANSI function, characters in SBCS code pages take one byte each, while most characters in DBCS code pages take two bytes; for the Unicode function, most currently defined Unicode characters (those in the Basic Multilingual Plane (BMP)) are one WORD while Unicode surrogates are two WORDs.

[in] `nTabPositions`

The number of tab-stop positions in the array pointed to by the `lpnTabStopPositions` parameter.

[in] `lpnTabStopPositions`

A pointer to an array containing the tab-stop positions, in device units. The tab stops must be sorted in increasing order; the smallest x-value should be the first item in the array.

Return value

If the function succeeds, the return value is the dimensions of the string in logical units. The height is in the high-order word and the width is in the low-order word.

If the function fails, the return value is 0. **GetTabbedTextExtent** will fail if `hDC` is invalid and if `nTabPositions` is less than 0.

Remarks

The current clipping region does not affect the width and height returned by the **GetTabbedTextExtent** function.

Because some devices do not place characters in regular cell arrays (that is, they kern the characters), the sum of the extents of the characters in a string may not be equal to the extent of the string.

If the *nTabPositions* parameter is zero and the *lpnTabStopPositions* parameter is **NULL**, tabs are expanded to eight times the average character width.

If *nTabPositions* is 1, the tab stops are separated by the distance specified by the first value in the array to which *lpnTabStopPositions* points.

NOTE

The winuser.h header defines GetTabbedTextExtent as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[TabbedTextOut](#)

GetUpdateRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetUpdateRect** function retrieves the coordinates of the smallest rectangle that completely encloses the update region of the specified window. **GetUpdateRect** retrieves the rectangle in logical coordinates. If there is no update region, **GetUpdateRect** retrieves an empty rectangle (sets all coordinates to zero).

Syntax

```
BOOL GetUpdateRect(
    [in]    HWND    hWnd,
    [out]   LPRECT  lpRect,
    [in]    BOOL    bErase
);
```

Parameters

[in] hWnd

Handle to the window whose update region is to be retrieved.

[out] lpRect

Pointer to the **RECT** structure that receives the coordinates, in device units, of the enclosing rectangle.

An application can set this parameter to **NULL** to determine whether an update region exists for the window. If this parameter is **NULL**, **GetUpdateRect** returns nonzero if an update region exists, and zero if one does not. This provides a simple and efficient means of determining whether a **WM_PAINT** message resulted from an invalid area.

[in] bErase

Specifies whether the background in the update region is to be erased. If this parameter is **TRUE** and the update region is not empty, **GetUpdateRect** sends a **WM_ERASEBKGND** message to the specified window to erase the background.

Return value

If the update region is not empty, the return value is nonzero.

If there is no update region, the return value is zero.

Remarks

The update rectangle retrieved by the **BeginPaint** function is identical to that retrieved by **GetUpdateRect**.

BeginPaint automatically validates the update region, so any call to **GetUpdateRect** made immediately after the call to **BeginPaint** retrieves an empty update region.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[BeginPaint](#)

[GetUpdateRgn](#)

[InvalidateRect](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

[UpdateWindow](#)

[ValidateRect](#)

GetUpdateRgn function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetUpdateRgn** function retrieves the update region of a window by copying it into the specified region. The coordinates of the update region are relative to the upper-left corner of the window (that is, they are client coordinates).

Syntax

```
int GetUpdateRgn(
    [in] HWND hWnd,
    [in] HRGN hRgn,
    [in] BOOL bErase
);
```

Parameters

[in] `hWnd`

Handle to the window with an update region that is to be retrieved.

[in] `hRgn`

Handle to the region to receive the update region.

[in] `bErase`

Specifies whether the window background should be erased and whether nonclient areas of child windows should be drawn. If this parameter is **FALSE**, no drawing is done.

Return value

The return value indicates the complexity of the resulting region; it can be one of the following values.

VALUE	MEANING
COMPLEXREGION	Region consists of more than one rectangle.
ERROR	An error occurred.
NULLRGN	Region is empty.
SIMPLEREGION	Region is a single rectangle.

Remarks

The **BeginPaint** function automatically validates the update region, so any call to **GetUpdateRgn** made immediately after the call to **BeginPaint** retrieves an empty update region.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

GetWindowDC function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetWindowDC** function retrieves the device context (DC) for the entire window, including title bar, menus, and scroll bars. A window device context permits painting anywhere in a window, because the origin of the device context is the upper-left corner of the window instead of the client area.

GetWindowDC assigns default attributes to the window device context each time it retrieves the device context. Previous attributes are lost.

Syntax

```
HDC GetWindowDC(
    [in] HWND hWnd
);
```

Parameters

[in] *hWnd*

A handle to the window with a device context that is to be retrieved. If this value is **NULL**, **GetWindowDC** retrieves the device context for the entire screen.

If this parameter is **NULL**, **GetWindowDC** retrieves the device context for the primary display monitor. To get the device context for other display monitors, use the [EnumDisplayMonitors](#) and [CreateDC](#) functions.

Return value

If the function succeeds, the return value is a handle to a device context for the specified window.

If the function fails, the return value is **NULL**, indicating an error or an invalid *hWnd* parameter.

Remarks

GetWindowDC is intended for special painting effects within a window's nonclient area. Painting in nonclient areas of any window is not recommended.

The [GetSystemMetrics](#) function can be used to retrieve the dimensions of various parts of the nonclient area, such as the title bar, menu, and scroll bars.

The [GetDC](#) function can be used to retrieve a device context for the entire screen.

After painting is complete, the [ReleaseDC](#) function must be called to release the device context. Not releasing the window device context has serious effects on painting requested by applications.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[BeginPaint](#)

[GetDC](#)

[GetSystemMetrics](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[ReleaseDC](#)

GetWindowRgn function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetWindowRgn** function obtains a copy of the window region of a window. The window region of a window is set by calling the [SetWindowRgn](#) function. The window region determines the area within the window where the system permits drawing. The system does not display any portion of a window that lies outside of the window region.

Syntax

```
int GetWindowRgn(
    [in] HWND hWnd,
    [in] HRGN hRgn
);
```

Parameters

[in] hWnd

Handle to the window whose window region is to be obtained.

[in] hRgn

Handle to the region which will be modified to represent the window region.

Return value

The return value specifies the type of the region that the function obtains. It can be one of the following values.

RETURN CODE	DESCRIPTION
NULLREGION	The region is empty.
SIMPLEREGION	The region is a single rectangle.
COMPLEXREGION	The region is more than one rectangle.
ERROR	The specified window does not have a region, or an error occurred while attempting to return the region.

Remarks

The coordinates of a window's window region are relative to the upper-left corner of the window, not the client area of the window.

To set the window region of a window, call the [SetWindowRgn](#) function.

Examples

The following code shows how you pass in the handle of an existing region.

```
HRGN hrgn = CreateRectRgn(0,0,0,0);
int regionType = GetWindowRgn(hwnd, hrgn);
if (regionType != ERROR)
{
/* hrgn contains window region */
}
DeleteObject(hrgn); /* finished with region */
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[SetWindowRgn](#)

GetWindowRgnBox function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **GetWindowRgnBox** function retrieves the dimensions of the tightest bounding rectangle for the window region of a window.

Syntax

```
int GetWindowRgnBox(
    [in]  HWND   hWnd,
    [out] LPRECT lprc
);
```

Parameters

[in] `hWnd`

Handle to the window.

[out] `lprc`

Pointer to a [RECT](#) structure that receives the rectangle dimensions, in device units relative to the upper-left corner of the window.

Return value

The return value specifies the type of the region that the function obtains. It can be one of the following values.

VALUE	MEANING
COMPLEXREGION	The region is more than one rectangle.
ERROR	The specified window does not have a region, or an error occurred while attempting to return the region.
NULLREGION	The region is empty.
SIMPLEREGION	The region is a single rectangle.

Remarks

The window region determines the area within the window where the system permits drawing. The system does not display any portion of a window that lies outside of the window region. The coordinates of a window's window region are relative to the upper-left corner of the window, not the client area of the window.

To set the window region of a window, call the [SetWindowRgn](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[GetClipBox](#)

[GetWindowRgn](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

[SetWindowRgn](#)

GrayStringA function (winuser.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **GrayString** function draws gray text at the specified location. The function draws the text by copying it into a memory bitmap, graying the bitmap, and then copying the bitmap to the screen. The function grays the text regardless of the selected brush and background. **GrayString** uses the font currently selected for the specified device context.

If the *lpOutputFunc* parameter is **NULL**, GDI uses the [TextOut](#) function, and the *lpData* parameter is assumed to be a pointer to the character string to be output. If the characters to be output cannot be handled by [TextOut](#) (for example, the string is stored as a bitmap), the application must supply its own output function.

Syntax

```
BOOL GrayStringA(
    [in] HDC           hdc,
    [in] HBRUSH        hBrush,
    [in] GRAYSTRINGPROC lpOutputFunc,
    [in] LPARAM        lpData,
    [in] int            nCount,
    [in] int            X,
    [in] int            Y,
    [in] int            nWidth,
    [in] int            nHeight
);
```

Parameters

[in] hdc

A handle to the device context.

[in] hBrush

A handle to the brush to be used for graying. If this parameter is **NULL**, the text is grayed with the same brush that was used to draw window text.

[in] lpOutputFunc

A pointer to the application-defined function that will draw the string, or, if [TextOut](#) is to be used to draw the string, it is a **NULL** pointer. For details, see the [OutputProc](#) callback function.

[in] lpData

A pointer to data to be passed to the output function. If the *lpOutputFunc* parameter is **NULL**, *lpData* must be a pointer to the string to be output.

[in] nCount

The number of characters to be output. If the *nCount* parameter is zero, **GrayString** calculates the length of the string (assuming *lpData* is a pointer to the string). If *nCount* is 1 and the function pointed to by *lpOutputFunc* returns **FALSE**, the image is shown but not grayed.

[in] x

The device x-coordinate of the starting position of the rectangle that encloses the string.

[in] *y*

The device y-coordinate of the starting position of the rectangle that encloses the string.

[in] *nWidth*

The width, in device units, of the rectangle that encloses the string. If this parameter is zero, **GrayString** calculates the width of the area, assuming *lpData* is a pointer to the string.

[in] *nHeight*

The height, in device units, of the rectangle that encloses the string. If this parameter is zero, **GrayString** calculates the height of the area, assuming *lpData* is a pointer to the string.

Return value

If the string is drawn, the return value is nonzero.

If either the [TextOut](#) function or the application-defined output function returned zero, or there was insufficient memory to create a memory bitmap for graying, the return value is zero.

Remarks

Without calling **GrayString**, an application can draw grayed strings on devices that support a solid gray color. The system color COLOR_GRAYTEXT is the solid-gray system color used to draw disabled text. The application can call the [GetSysColor](#) function to retrieve the color value of COLOR_GRAYTEXT. If the color is other than zero (black), the application can call the [SetTextColor](#) function to set the text color to the color value and then draw the string directly. If the retrieved color is black, the application must call **GrayString** to gray the text.

NOTE

The winuser.h header defines **GrayString** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[DrawText](#)

[GetSysColor](#)

[OutputProc](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[SetTextColor](#)

[TabbedTextOut](#)

[TextOut](#)

GRAYSTRINGPROC callback function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **OutputProc** function is an application-defined callback function used with the [GrayString](#) function. It is used to draw a string. The **GRAYSTRINGPROC** type defines a pointer to this callback function. **OutputProc** is a placeholder for the application-defined or library-defined function name.

Syntax

```
GRAYSTRINGPROC Graystringproc;  
  
BOOL Graystringproc(  
    HDC unnamedParam1,  
    LPARAM unnamedParam2,  
    int unnamedParam3  
)  
{...}
```

Parameters

unnamedParam1

A handle to a device context with a bitmap of at least the width and height specified by the *nWidth* and *nHeight* parameters passed to [GrayString](#).

unnamedParam2

A pointer to the string to be drawn.

unnamedParam3

The length, in characters, of the string.

Return value

If it succeeds, the callback function should return TRUE.

If the function fails, the return value is FALSE.

Remarks

The callback function must draw an image relative to the coordinates (0,0).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header

winuser.h (include Windows.h)

See also

[GrayString](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

GrayStringW function (winuser.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **GrayString** function draws gray text at the specified location. The function draws the text by copying it into a memory bitmap, graying the bitmap, and then copying the bitmap to the screen. The function grays the text regardless of the selected brush and background. **GrayString** uses the font currently selected for the specified device context.

If the *lpOutputFunc* parameter is **NULL**, GDI uses the [TextOut](#) function, and the *lpData* parameter is assumed to be a pointer to the character string to be output. If the characters to be output cannot be handled by [TextOut](#) (for example, the string is stored as a bitmap), the application must supply its own output function.

Syntax

```
BOOL GrayStringW(
    [in] HDC           hdc,
    [in] HBRUSH        hBrush,
    [in] GRAYSTRINGPROC lpOutputFunc,
    [in] LPARAM        lpData,
    [in] int            nCount,
    [in] int            X,
    [in] int            Y,
    [in] int            nWidth,
    [in] int            nHeight
);
```

Parameters

[in] hdc

A handle to the device context.

[in] hBrush

A handle to the brush to be used for graying. If this parameter is **NULL**, the text is grayed with the same brush that was used to draw window text.

[in] lpOutputFunc

A pointer to the application-defined function that will draw the string, or, if [TextOut](#) is to be used to draw the string, it is a **NULL** pointer. For details, see the [OutputProc](#) callback function.

[in] lpData

A pointer to data to be passed to the output function. If the *lpOutputFunc* parameter is **NULL**, *lpData* must be a pointer to the string to be output.

[in] nCount

The number of characters to be output. If the *nCount* parameter is zero, **GrayString** calculates the length of the string (assuming *lpData* is a pointer to the string). If *nCount* is 1 and the function pointed to by *lpOutputFunc* returns **FALSE**, the image is shown but not grayed.

[in] x

The device x-coordinate of the starting position of the rectangle that encloses the string.

[in] *y*

The device y-coordinate of the starting position of the rectangle that encloses the string.

[in] *nWidth*

The width, in device units, of the rectangle that encloses the string. If this parameter is zero, **GrayString** calculates the width of the area, assuming *lpData* is a pointer to the string.

[in] *nHeight*

The height, in device units, of the rectangle that encloses the string. If this parameter is zero, **GrayString** calculates the height of the area, assuming *lpData* is a pointer to the string.

Return value

If the string is drawn, the return value is nonzero.

If either the [TextOut](#) function or the application-defined output function returned zero, or there was insufficient memory to create a memory bitmap for graying, the return value is zero.

Remarks

Without calling **GrayString**, an application can draw grayed strings on devices that support a solid gray color. The system color COLOR_GRAYTEXT is the solid-gray system color used to draw disabled text. The application can call the [GetSysColor](#) function to retrieve the color value of COLOR_GRAYTEXT. If the color is other than zero (black), the application can call the [SetTextColor](#) function to set the text color to the color value and then draw the string directly. If the retrieved color is black, the application must call **GrayString** to gray the text.

NOTE

The winuser.h header defines **GrayString** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[DrawText](#)

[GetSysColor](#)

[OutputProc](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[SetTextColor](#)

[TabbedTextOut](#)

[TextOut](#)

InflateRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **InflateRect** function increases or decreases the width and height of the specified rectangle. The **InflateRect** function adds *-dx* units to the left end and *dx* to the right end of the rectangle and *-dy* units to the top and *dy* to the bottom. The *dx* and *dy* parameters are signed values; positive values increase the width and height, and negative values decrease them.

Syntax

```
BOOL InflateRect(
    [in, out] LPRECT lprc,
    [in]      int     dx,
    [in]      int     dy
);
```

Parameters

[in, out] *lprc*

A pointer to the [RECT](#) structure that increases or decreases in size.

[in] *dx*

The amount to increase or decrease the rectangle width. This parameter must be negative to decrease the width.

[in] *dy*

The amount to increase or decrease the rectangle height. This parameter must be negative to decrease the height.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[IntersectRect](#)

[OffsetRect](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

[UnionRect](#)

IntersectRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **IntersectRect** function calculates the intersection of two source rectangles and places the coordinates of the intersection rectangle into the destination rectangle. If the source rectangles do not intersect, an empty rectangle (in which all coordinates are set to zero) is placed into the destination rectangle.

Syntax

```
BOOL IntersectRect(
    [out] LPRECT     lprcDst,
    [in]  const RECT *lprcSrc1,
    [in]  const RECT *lprcSrc2
);
```

Parameters

[out] *lprcDst*

A pointer to the **RECT** structure that is to receive the intersection of the rectangles pointed to by the *lprcSrc1* and *lprcSrc2* parameters. This parameter cannot be **NULL**.

[in] *lprcSrc1*

A pointer to the **RECT** structure that contains the first source rectangle.

[in] *lprcSrc2*

A pointer to the **RECT** structure that contains the second source rectangle.

Return value

If the rectangles intersect, the return value is nonzero.

If the rectangles do not intersect, the return value is zero.

Remarks

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Examples

For an example, see [Using Rectangles](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[InflateRect](#)

[OffsetRect](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

[UnionRect](#)

InvalidateRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **InvalidateRect** function adds a rectangle to the specified window's update region. The update region represents the portion of the window's client area that must be redrawn.

Syntax

```
BOOL InvalidateRect(
    [in] HWND      hWnd,
    [in] const RECT *lpRect,
    [in] BOOL       bErase
);
```

Parameters

[in] hWnd

A handle to the window whose update region has changed. If this parameter is **NULL**, the system invalidates and redraws all windows, not just the windows for this application, and sends the [WM_ERASEBKGND](#) and [WM_NCPAINT](#) messages before the function returns. Setting this parameter to **NULL** is not recommended.

[in] lpRect

A pointer to a [RECT](#) structure that contains the client coordinates of the rectangle to be added to the update region. If this parameter is **NULL**, the entire client area is added to the update region.

[in] bErase

Specifies whether the background within the update region is to be erased when the update region is processed. If this parameter is **TRUE**, the background is erased when the [BeginPaint](#) function is called. If this parameter is **FALSE**, the background remains unchanged.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The invalidated areas accumulate in the update region until the region is processed when the next [WM_PAINT](#) message occurs or until the region is validated by using the [ValidateRect](#) or [ValidateRgn](#) function.

The system sends a [WM_PAINT](#) message to a window whenever its update region is not empty and there are no other messages in the application queue for that window.

If the *bErase* parameter is **TRUE** for any part of the update region, the background is erased in the entire region, not just in the specified part.

Examples

For an example, see [Invalidating the Client Area](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[BeginPaint](#)

[InvalidateRgn](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

[ValidateRect](#)

[ValidateRgn](#)

[WM_ERASEBKGND](#)

[WM_NCPAINT](#)

[WM_PAINT](#)

InvalidateRgn function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **InvalidateRgn** function invalidates the client area within the specified region by adding it to the current update region of a window. The invalidated region, along with all other areas in the update region, is marked for painting when the next [WM_PAINT](#) message occurs.

Syntax

```
BOOL InvalidateRgn(
    [in] HWND hWnd,
    [in] HRGN hRgn,
    [in] BOOL bErase
);
```

Parameters

[in] hWnd

A handle to the window with an update region that is to be modified.

[in] hRgn

A handle to the region to be added to the update region. The region is assumed to have client coordinates. If this parameter is **NULL**, the entire client area is added to the update region.

[in] bErase

Specifies whether the background within the update region should be erased when the update region is processed. If this parameter is **TRUE**, the background is erased when the [BeginPaint](#) function is called. If the parameter is **FALSE**, the background remains unchanged.

Return value

The return value is always nonzero.

Remarks

Invalidated areas accumulate in the update region until the next [WM_PAINT](#) message is processed or until the region is validated by using the [ValidateRect](#) or [ValidateRgn](#) function.

The system sends a [WM_PAINT](#) message to a window whenever its update region is not empty and there are no other messages in the application queue for that window.

The specified region must have been created by using one of the region functions.

If the *bErase* parameter is **TRUE** for any part of the update region, the background in the entire region is erased, not just in the specified part.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[BeginPaint](#)

[InvalidateRect](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[ValidateRect](#)

[ValidateRgn](#)

[WM_PAINT](#)

InvertRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **InvertRect** function inverts a rectangle in a window by performing a logical NOT operation on the color values for each pixel in the rectangle's interior.

Syntax

```
BOOL InvertRect(
    [in] HDC      hDC,
    [in] const RECT *lprc
);
```

Parameters

[in] `hDC`

A handle to the device context.

[in] `lprc`

A pointer to a [RECT](#) structure that contains the logical coordinates of the rectangle to be inverted.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

On monochrome screens, **InvertRect** makes white pixels black and black pixels white. On color screens, the inversion depends on how colors are generated for the screen. Calling **InvertRect** twice for the same rectangle restores the display to its previous colors.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

API set	ext-ms-win-ntuser-gui-l1-1-1 (introduced in Windows 8.1)
---------	--

See also

[FillRect](#)

[Filled Shape Functions](#)

[Filled Shapes Overview](#)

[RECT](#)

IsRectEmpty function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **IsRectEmpty** function determines whether the specified rectangle is empty. An empty rectangle is one that has no area; that is, the coordinate of the right side is less than or equal to the coordinate of the left side, or the coordinate of the bottom side is less than or equal to the coordinate of the top side.

Syntax

```
BOOL IsRectEmpty(  
    [in] const RECT *lprc  
) ;
```

Parameters

[in] lprc

Pointer to a [RECT](#) structure that contains the logical coordinates of the rectangle.

Return value

If the rectangle is empty, the return value is nonzero.

If the rectangle is not empty, the return value is zero.

Remarks

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Examples

For an example, see [Using Rectangles](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[EqualRect](#)

[PtInRect](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

LoadBitmapA function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

[**LoadBitmap** is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Instead, use [LoadImage](#) and [DrawFrameControl](#).]

The **LoadBitmap** function loads the specified bitmap resource from a module's executable file.

Syntax

```
HBITMAP LoadBitmapA(
    [in] HINSTANCE hInstance,
    [in] LPCSTR     lpBitmapName
);
```

Parameters

[in] hInstance

A handle to the instance of the module whose executable file contains the bitmap to be loaded.

[in] lpBitmapName

A pointer to a null-terminated string that contains the name of the bitmap resource to be loaded. Alternatively, this parameter can consist of the resource identifier in the low-order word and zero in the high-order word. The [MAKEINTRESOURCE](#) macro can be used to create this value.

Return value

If the function succeeds, the return value is the handle to the specified bitmap.

If the function fails, the return value is **NULL**.

Remarks

If the bitmap pointed to by the *lpBitmapName* parameter does not exist or there is insufficient memory to load the bitmap, the function fails.

LoadBitmap creates a compatible bitmap of the display, which cannot be selected to a printer. To load a bitmap that you can select to a printer, call [LoadImage](#) and specify **LR_CREATEDIBSECTION** to create a DIB section. A DIB section can be selected to any device.

An application can use the **LoadBitmap** function to access predefined bitmaps. To do so, the application must set the *hInstance* parameter to **NULL** and the *lpBitmapName* parameter to one of the following values.

BITMAP NAME	BITMAP NAME
OBM_BTNCORNERS	OBM_OLD_RESTORE
OBM_BTSIZE	OBM_OLD_RGARROW

OBM_CHECK	OBM_OLD_UPARROW
OBM_CHECKBOXES	OBM_OLD_ZOOM
OBM_CLOSE	OBM_REDUCE
OBM_COMBO	OBM_REDUCED
OBM_DNARROW	OBM_RESTORE
OBM_DNARROWD	OBM_RESTORED
OBM_DNARROWI	OBM_RGARROW
OBM_LFARROW	OBM_RGARROWD
OBM_LFARROWD	OBM_RGARROWI
OBM_LFARROWI	OBM_SIZE
OBM_MNARROW	OBM_UPARROW
OBM_OLD_CLOSE	OBM_UPARROWD
OBM_OLD_DNARROW	OBM_UPARROWI
OBM_OLD_LFARROW	OBM_ZOOM
OBM_OLD_REDUCE	OBM_ZOOMD

Bitmap names that begin with OBM_OLD represent bitmaps used by 16-bit versions of Windows earlier than 3.0.

For an application to use any of the OBM_ constants, the constant OEMRESOURCE must be defined before the Windows.h header file is included.

The application must call the [DeleteObject](#) function to delete each bitmap handle returned by the [LoadBitmap](#) function.

Examples

For an example, see Example of Menu-Item Bitmaps in [Using Menus](#).

NOTE

The winuser.h header defines LoadBitmap as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-1 (introduced in Windows 8.1)

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateBitmap](#)

[DeleteObject](#)

[DrawFrameControl](#)

[LoadCursor](#)

[LoadIcon](#)

[LoadImage](#)

[MAKEINTRESOURCE](#)

LoadBitmapW function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

[**LoadBitmap** is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Instead, use [LoadImage](#) and [DrawFrameControl](#).]

The **LoadBitmap** function loads the specified bitmap resource from a module's executable file.

Syntax

```
HBITMAP LoadBitmapW(
    [in] HINSTANCE hInstance,
    [in] LPCWSTR   lpBitmapName
);
```

Parameters

[in] hInstance

A handle to the instance of the module whose executable file contains the bitmap to be loaded.

[in] lpBitmapName

A pointer to a null-terminated string that contains the name of the bitmap resource to be loaded. Alternatively, this parameter can consist of the resource identifier in the low-order word and zero in the high-order word. The [MAKEINTRESOURCE](#) macro can be used to create this value.

Return value

If the function succeeds, the return value is the handle to the specified bitmap.

If the function fails, the return value is **NULL**.

Remarks

If the bitmap pointed to by the *lpBitmapName* parameter does not exist or there is insufficient memory to load the bitmap, the function fails.

LoadBitmap creates a compatible bitmap of the display, which cannot be selected to a printer. To load a bitmap that you can select to a printer, call [LoadImage](#) and specify **LR_CREATEDIBSECTION** to create a DIB section. A DIB section can be selected to any device.

An application can use the **LoadBitmap** function to access predefined bitmaps. To do so, the application must set the *hInstance* parameter to **NULL** and the *lpBitmapName* parameter to one of the following values.

BITMAP NAME	BITMAP NAME
OBM_BTNCORNERS	OBM_OLD_RESTORE
OBM_BTSIZE	OBM_OLD_RGARROW

OBM_CHECK	OBM_OLD_UPARROW
OBM_CHECKBOXES	OBM_OLD_ZOOM
OBM_CLOSE	OBM_REDUCE
OBM_COMBO	OBM_REDUCED
OBM_DNARROW	OBM_RESTORE
OBM_DNARROWD	OBM_RESTORED
OBM_DNARROWI	OBM_RGARROW
OBM_LFARROW	OBM_RGARROWD
OBM_LFARROWD	OBM_RGARROWI
OBM_LFARROWI	OBM_SIZE
OBM_MNARROW	OBM_UPARROW
OBM_OLD_CLOSE	OBM_UPARROWD
OBM_OLD_DNARROW	OBM_UPARROWI
OBM_OLD_LFARROW	OBM_ZOOM
OBM_OLD_REDUCE	OBM_ZOOMD

Bitmap names that begin with OBM_OLD represent bitmaps used by 16-bit versions of Windows earlier than 3.0.

For an application to use any of the OBM_ constants, the constant OEMRESOURCE must be defined before the Windows.h header file is included.

The application must call the [DeleteObject](#) function to delete each bitmap handle returned by the [LoadBitmap](#) function.

Examples

For an example, see Example of Menu-Item Bitmaps in [Using Menus](#).

NOTE

The winuser.h header defines LoadBitmap as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-1 (introduced in Windows 8.1)

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateBitmap](#)

[DeleteObject](#)

[DrawFrameControl](#)

[LoadCursor](#)

[LoadIcon](#)

[LoadImage](#)

[MAKEINTRESOURCE](#)

LockWindowUpdate function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **LockWindowUpdate** function disables or enables drawing in the specified window. Only one window can be locked at a time.

Syntax

```
BOOL LockWindowUpdate(  
    [in] HWND hWndLock  
);
```

Parameters

[in] *hWndLock*

The window in which drawing will be disabled. If this parameter is **NULL**, drawing in the locked window is enabled.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero, indicating that an error occurred or another window was already locked.

Remarks

The purpose of the **LockWindowUpdate** function is to permit drag/drop feedback to be drawn over a window without interference from the window itself. The intent is that the window is locked when feedback is drawn and unlocked when feedback is complete. **LockWindowUpdate** is not intended for general-purpose suppression of window redraw. Use the [WM_SETREDRAW](#) message to disable redrawing of a particular window.

If an application with a locked window (or any locked child windows) calls the [GetDC](#), [GetDCEx](#), or [BeginPaint](#) function, the called function returns a device context with a visible region that is empty. This will occur until the application unlocks the window by calling **LockWindowUpdate**, specifying a value of **NULL** for *hWndLock*.

If an application attempts to draw within a locked window, the system records the extent of the attempted operation in a bounding rectangle. When the window is unlocked, the system invalidates the area within this bounding rectangle, forcing an eventual [WM_PAINT](#) message to be sent to the previously locked window and its child windows. If no drawing has occurred while the window updates were locked, no area is invalidated.

LockWindowUpdate does not make the specified window invisible and does not clear the **WS_VISIBLE** style bit.

A locked window cannot be moved.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

[BeginPaint](#)

[GetDC](#)

[GetDCEx](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[WM_PAINT](#)

MapWindowPoints function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MapWindowPoints** function converts (maps) a set of points from a coordinate space relative to one window to a coordinate space relative to another window.

Syntax

```
int MapWindowPoints(
    [in]      HWND      hWndFrom,
    [in]      HWND      hWndTo,
    [in, out] LPPOINT  lpPoints,
    [in]      UINT      cPoints
);
```

Parameters

[in] *hWndFrom*

A handle to the window from which points are converted. If this parameter is **NULL** or **HWND_DESKTOP**, the points are presumed to be in screen coordinates.

[in] *hWndTo*

A handle to the window to which points are converted. If this parameter is **NULL** or **HWND_DESKTOP**, the points are converted to screen coordinates.

[in, out] *lpPoints*

A pointer to an array of **POINT** structures that contain the set of points to be converted. The points are in device units. This parameter can also point to a **RECT** structure, in which case the *cPoints* parameter should be set to 2.

[in] *cPoints*

The number of **POINT** structures in the array pointed to by the *lpPoints* parameter.

Return value

If the function succeeds, the low-order word of the return value is the number of pixels added to the horizontal coordinate of each source point in order to compute the horizontal coordinate of each destination point. (In addition to that, if precisely one of *hWndFrom* and *hWndTo* is mirrored, then each resulting horizontal coordinate is multiplied by -1.) The high-order word is the number of pixels added to the vertical coordinate of each source point in order to compute the vertical coordinate of each destination point.

If the function fails, the return value is zero. Call **SetLastError** prior to calling this method to differentiate an error return value from a legitimate "0" return value.

Remarks

If *hWndFrom* or *hWndTo* (or both) are mirrored windows (that is, have **WS_EX_LAYOUTRTL** extended style) and precisely two points are passed in *lpPoints*, **MapWindowPoints** will interpret those two points as a **RECT** and possibly automatically swap the left and right fields of that rectangle to ensure that left is not greater than

right. If any number of points other than 2 is passed in *lpPoints*, then **MapWindowPoints** will correctly map the coordinates of each of those points separately, so if you pass in a pointer to an array of more than one rectangle in *lpPoints*, the new rectangles may get their left field greater than right. Thus, to guarantee the correct transformation of rectangle coordinates, you must call **MapWindowPoints** with one **RECT** pointer at a time, as shown in the following example:

```
RECT rc[10];

for(int i = 0; i < (sizeof(rc)/sizeof(rc[0])); i++)
{
    MapWindowPoints(hWnd1, hWnd2, (LPPOINT)(&rc[i]), (sizeof(RECT)/sizeof(POINT)) );
}
```

Also, if you need to map precisely two independent points and don't want the **RECT** logic applied to them by **MapWindowPoints**, to guarantee the correct result you must call **MapWindowPoints** with one **POINT** pointer at a time, as shown in the following example:

```
POINT pt[2];

MapWindowPoints(hWnd1, hWnd2, &pt[0], 1);
MapWindowPoints(hWnd1, hWnd2, &pt[1], 1);
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

[ClientToScreen](#)

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[POINT](#)

RECT

ScreenToClient

MONITORENUMPROC callback function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

A **MonitorEnumProc** function is an application-defined callback function that is called by the [EnumDisplayMonitors](#) function.

A value of type **MONITORENUMPROC** is a pointer to a **MonitorEnumProc** function.

Syntax

```
MONITORENUMPROC Monitorenumproc;

BOOL Monitorenumproc(
    HMONITOR unnamedParam1,
    HDC unnamedParam2,
    LPRECT unnamedParam3,
    LPARAM unnamedParam4
)
{...}
```

Parameters

unnamedParam1

A handle to the display monitor. This value will always be non-**NULL**.

unnamedParam2

A handle to a device context.

The device context has color attributes that are appropriate for the display monitor identified by *hMonitor*. The clipping area of the device context is set to the intersection of the visible region of the device context identified by the *hdc* parameter of [EnumDisplayMonitors](#), the rectangle pointed to by the *lprcClip* parameter of [EnumDisplayMonitors](#), and the display monitor rectangle.

This value is **NULL** if the *hdc* parameter of [EnumDisplayMonitors](#) was **NULL**.

unnamedParam3

A pointer to a [RECT](#) structure.

If *hdcMonitor* is non-**NULL**, this rectangle is the intersection of the clipping area of the device context identified by *hdcMonitor* and the display monitor rectangle. The rectangle coordinates are device-context coordinates.

If *hdcMonitor* is **NULL**, this rectangle is the display monitor rectangle. The rectangle coordinates are virtual-screen coordinates.

unnamedParam4

Application-defined data that [EnumDisplayMonitors](#) passes directly to the enumeration function.

Return value

To continue the enumeration, return **TRUE**.

To stop the enumeration, return FALSE.

Remarks

You can use the [EnumDisplayMonitors](#) function to enumerate the set of display monitors that intersect the visible region of a specified device context and, optionally, a clipping rectangle. To do this, set the *hdc* parameter to a non-NULL value, and set the *lprcClip* parameter as needed.

You can also use the [EnumDisplayMonitors](#) function to enumerate one or more of the display monitors on the desktop, without supplying a device context. To do this, set the *hdc* parameter of [EnumDisplayMonitors](#) to NULL and set the *lprcClip* parameter as needed.

In all cases, [EnumDisplayMonitors](#) calls a specified **MonitorEnumProc** function once for each display monitor in the calculated enumeration set. The **MonitorEnumProc** function always receives a handle to the display monitor.

If the *hdc* parameter of [EnumDisplayMonitors](#) is non-NULL, the **MonitorEnumProc** function also receives a handle to a device context whose color format is appropriate for the display monitor. You can then paint into the device context in a manner that is optimal for the display monitor.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

[EnumDisplayMonitors](#)

[Multiple Display Monitors Functions](#)

[Multiple Display Monitors Overview](#)

MonitorFromPoint function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MonitorFromPoint** function retrieves a handle to the display monitor that contains a specified point.

Syntax

```
HMONITOR MonitorFromPoint(  
    [in] POINT pt,  
    [in] DWORD dwFlags  
)
```

Parameters

[in] *pt*

A **POINT** structure that specifies the point of interest in virtual-screen coordinates.

[in] *dwFlags*

Determines the function's return value if the point is not contained within any display monitor.

This parameter can be one of the following values.

VALUE	MEANING
MONITOR_DEFAULTTONEAREST	Returns a handle to the display monitor that is nearest to the point.
MONITOR_DEFAULTTONULL	Returns NULL .
MONITOR_DEFAULTTOPRIMARY	Returns a handle to the primary display monitor.

Return value

If the point is contained by a display monitor, the return value is an **HMONITOR** handle to that display monitor.

If the point is not contained by a display monitor, the return value depends on the value of *dwFlags*.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[MonitorFromRect](#)

[MonitorFromWindow](#)

[Multiple Display Monitors Functions](#)

[Multiple Display Monitors Overview](#)

MonitorFromRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MonitorFromRect** function retrieves a handle to the display monitor that has the largest area of intersection with a specified rectangle.

Syntax

```
HMONITOR MonitorFromRect(
    [in] LPCRECT lprc,
    [in] DWORD    dwFlags
);
```

Parameters

[in] *lprc*

A pointer to a [RECT](#) structure that specifies the rectangle of interest in virtual-screen coordinates.

[in] *dwFlags*

Determines the function's return value if the rectangle does not intersect any display monitor.

This parameter can be one of the following values.

VALUE	MEANING
MONITOR_DEFAULTTONEAREST	Returns a handle to the display monitor that is nearest to the rectangle.
MONITOR_DEFAULTTONULL	Returns NULL .
MONITOR_DEFAULTTOPRIMARY	Returns a handle to the primary display monitor.

Return value

If the rectangle intersects one or more display monitor rectangles, the return value is an **HMONITOR** handle to the display monitor that has the largest area of intersection with the rectangle.

If the rectangle does not intersect a display monitor, the return value depends on the value of *dwFlags*.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[MonitorFromPoint](#)

[MonitorFromWindow](#)

[Multiple Display Monitors Functions](#)

[Multiple Display Monitors Overview](#)

MonitorFromWindow function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MonitorFromWindow** function retrieves a handle to the display monitor that has the largest area of intersection with the bounding rectangle of a specified window.

Syntax

```
HMONITOR MonitorFromWindow(
    [in] HWND hwnd,
    [in] DWORD dwFlags
);
```

Parameters

[in] `hwnd`

A handle to the window of interest.

[in] `dwFlags`

Determines the function's return value if the window does not intersect any display monitor.

This parameter can be one of the following values.

VALUE	MEANING
<code>MONITOR_DEFAULTTONEAREST</code>	Returns a handle to the display monitor that is nearest to the window.
<code>MONITOR_DEFAULTTONULL</code>	Returns <code>NULL</code> .
<code>MONITOR_DEFAULTTOPRIMARY</code>	Returns a handle to the primary display monitor.

Return value

If the window intersects one or more display monitor rectangles, the return value is an **HMONITOR** handle to the display monitor that has the largest area of intersection with the window.

If the window does not intersect a display monitor, the return value depends on the value of *dwFlags*.

Remarks

If the window is currently minimized, **MonitorFromWindow** uses the rectangle of the window before it was minimized.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-sysparams-ext-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

[MonitorFromPoint](#)

[MonitorFromRect](#)

[Multiple Display Monitors Functions](#)

[Multiple Display Monitors Overview](#)

MONITORINFO structure (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MONITORINFO** structure contains information about a display monitor.

The **GetMonitorInfo** function stores information in a **MONITORINFO** structure or a **MONITORINFOEX** structure.

The **MONITORINFO** structure is a subset of the **MONITORINFOEX** structure. The **MONITORINFOEX** structure adds a string member to contain a name for the display monitor.

Syntax

```
typedef struct tagMONITORINFO {  
    DWORD cbSize;  
    RECT rcMonitor;  
    RECT rcWork;  
    DWORD dwFlags;  
} MONITORINFO, *LPMONITORINFO;
```

Members

cbSize

The size of the structure, in bytes.

Set this member to `sizeof (MONITORINFO)` before calling the **GetMonitorInfo** function. Doing so lets the function determine the type of structure you are passing to it.

rcMonitor

A **RECT** structure that specifies the display monitor rectangle, expressed in virtual-screen coordinates. Note that if the monitor is not the primary display monitor, some of the rectangle's coordinates may be negative values.

rcWork

A **RECT** structure that specifies the work area rectangle of the display monitor, expressed in virtual-screen coordinates. Note that if the monitor is not the primary display monitor, some of the rectangle's coordinates may be negative values.

dwFlags

A set of flags that represent attributes of the display monitor.

The following flag is defined.

VALUE	MEANING
MONITORINFOF_PRIMARY	This is the primary display monitor.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

[GetMonitorInfo](#)

[MONITORINFOEX](#)

[Multiple Display Monitors Overview](#)

[Multiple Display Monitors Structures](#)

MONITORINFOEXA structure (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MONITORINFOEX** structure contains information about a display monitor.

The [GetMonitorInfo](#) function stores information into a **MONITORINFOEX** structure or a [MONITORINFO](#) structure.

The **MONITORINFOEX** structure is a superset of the [MONITORINFO](#) structure. The **MONITORINFOEX** structure adds a string member to contain a name for the display monitor.

Syntax

```
typedef struct tagMONITORINFOEXA : tagMONITORINFO {  
    CHAR szDevice[CCHDEVICENAME];  
} MONITORINFOEXA, *LPMONITORINFOEXA;
```

Inheritance

The **MONITORINFOEXA** structure implements [tagMONITORINFO](#).

Members

szDevice

A string that specifies the device name of the monitor being used. Most applications have no use for a display monitor name, and so can save some bytes by using a [MONITORINFO](#) structure.

Remarks

NOTE

The winuser.h header defines **MONITORINFOEX** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

MONITORINFOEXW structure (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **MONITORINFOEX** structure contains information about a display monitor.

The [GetMonitorInfo](#) function stores information into a **MONITORINFOEX** structure or a [MONITORINFO](#) structure.

The **MONITORINFOEX** structure is a superset of the [MONITORINFO](#) structure. The **MONITORINFOEX** structure adds a string member to contain a name for the display monitor.

Syntax

```
typedef struct tagMONITORINFOEXW : tagMONITORINFO {  
    WCHAR szDevice[CCHDEVICENAME];  
} MONITORINFOEXW, *LPMONITORINFOEXW;
```

Inheritance

The **MONITORINFOEXW** structure implements [tagMONITORINFO](#).

Members

szDevice

A string that specifies the device name of the monitor being used. Most applications have no use for a display monitor name, and so can save some bytes by using a [MONITORINFO](#) structure.

Remarks

NOTE

The winuser.h header defines **MONITORINFOEX** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

[GetMonitorInfo](#)

[MONITORINFO](#)

[Multiple Display Monitors Overview](#)

[Multiple Display Monitors Structures](#)

OffsetRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **OffsetRect** function moves the specified rectangle by the specified offsets.

Syntax

```
BOOL OffsetRect(
    [in, out] LPRECT lprc,
    [in]      int     dx,
    [in]      int     dy
);
```

Parameters

[in, out] **lprc**

Pointer to a [RECT](#) structure that contains the logical coordinates of the rectangle to be moved.

[in] **dx**

Specifies the amount to move the rectangle left or right. This parameter must be a negative value to move the rectangle to the left.

[in] **dy**

Specifies the amount to move the rectangle up or down. This parameter must be a negative value to move the rectangle up.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Examples

For an example, see [Using Rectangles](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[InflateRect](#)

[IntersectRect](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

[UnionRect](#)

PaintDesktop function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PaintDesktop** function fills the clipping region in the specified device context with the desktop pattern or wallpaper. The function is provided primarily for shell desktops.

Syntax

```
BOOL PaintDesktop(  
    [in] HDC hdc  
>;
```

Parameters

[in] hdc

Handle to the device context.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

PAINTSTRUCT structure (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PAINTSTRUCT** structure contains information for an application. This information can be used to paint the client area of a window owned by that application.

Syntax

```
typedef struct tagPAINTSTRUCT {  
    HDC    hdc;  
    BOOL   fErase;  
    RECT   rcPaint;  
    BOOL   fRestore;  
    BOOL   fIncUpdate;  
    BYTE   rgbReserved[32];  
} PAINTSTRUCT, *PPAINTSTRUCT, *NPPAINTSTRUCT, *LPPAINTSTRUCT;
```

Members

hdc

A handle to the display DC to be used for painting.

fErase

Indicates whether the background must be erased. This value is nonzero if the application should erase the background. The application is responsible for erasing the background if a window class is created without a background brush. For more information, see the description of the **hbrBackground** member of the [WNDCLASS](#) structure.

rcPaint

A [RECT](#) structure that specifies the upper left and lower right corners of the rectangle in which the painting is requested, in device units relative to the upper-left corner of the client area.

fRestore

Reserved; used internally by the system.

fIncUpdate

Reserved; used internally by the system.

rgbReserved

Reserved; used internally by the system.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

[BeginPaint](#)

[Painting and Drawing Overview](#)

[Painting and Drawing Structures](#)

[RECT](#)

[WNDCLASS](#)

POINTSTOPOINT macro (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **POINTSTOPOINT** macro copies the contents of a [POINTS](#) structure into a [POINT](#) structure.

Syntax

```
void POINTSTOPOINT(
    pt,
    pts
);
```

Parameters

`pt`

The [POINT](#) structure to receive the contents of the [POINTS](#) structure.

`pts`

The [POINTS](#) structure to copy.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

[MAKEPOINTS](#)

[POINTTOPONTS](#)

[Rectangle Macros](#)

[Rectangles Overview](#)

POINTTOPPOINTS macro (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **POINTTOPPOINTS** macro converts a [POINT](#) structure to a [POINTS](#) structure.

Syntax

```
void POINTTOPPOINTS(
    pt
);
```

Parameters

pt

The [POINT](#) structure to convert.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

[MAKEPOINTS](#)

[POINT](#)

[POINTS](#)

[Rectangle Macros](#)

[Rectangles Overview](#)

PtInRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **PtInRect** function determines whether the specified point lies within the specified rectangle. A point is within a rectangle if it lies on the left or top side or is within all four sides. A point on the right or bottom side is considered outside the rectangle.

Syntax

```
BOOL PtInRect(
    [in] const RECT *lprc,
    [in] POINT      pt
);
```

Parameters

[in] lprc

A pointer to a [RECT](#) structure that contains the specified rectangle.

[in] pt

A [POINT](#) structure that contains the specified point.

Return value

If the specified point lies within the rectangle, the return value is nonzero.

If the specified point does not lie within the rectangle, the return value is zero.

Remarks

The rectangle must be normalized before **PtInRect** is called. That is, **lprc.right** must be greater than **lprc.left** and **lprc.bottom** must be greater than **lprc.top**. If the rectangle is not normalized, a point is never considered inside of the rectangle.

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Examples

For an example, see [Using Rectangles](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[EqualRect](#)

[IsRectEmpty](#)

[POINT](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

RedrawWindow function (winuser.h)

4/21/2022 • 3 minutes to read • [Edit Online](#)

The **RedrawWindow** function updates the specified rectangle or region in a window's client area.

Syntax

```
BOOL RedrawWindow(
    [in] HWND      hWnd,
    [in] const RECT *lprcUpdate,
    [in] HRGN      hrgnUpdate,
    [in] UINT      flags
);
```

Parameters

[in] *hWnd*

A handle to the window to be redrawn. If this parameter is **NULL**, the desktop window is updated.

[in] *lprcUpdate*

A pointer to a [RECT](#) structure containing the coordinates, in device units, of the update rectangle. This parameter is ignored if the *hrgnUpdate* parameter identifies a region.

[in] *hrgnUpdate*

A handle to the update region. If both the *hrgnUpdate* and *lprcUpdate* parameters are **NULL**, the entire client area is added to the update region.

[in] *flags*

One or more redraw flags. This parameter can be used to invalidate or validate a window, control repainting, and control which windows are affected by **RedrawWindow**.

The following flags are used to invalidate the window.

FLAG (INVALIDATION)	DESCRIPTION
RDW_ERASE	Causes the window to receive a WM_ERASEBKGND message when the window is repainted. The RDW_INVALIDATE flag must also be specified; otherwise, RDW_ERASE has no effect.
RDW_FRAME	Causes any part of the nonclient area of the window that intersects the update region to receive a WM_NCPAINT message. The RDW_INVALIDATE flag must also be specified; otherwise, RDW_FRAME has no effect. The WM_NCPAINT message is typically not sent during the execution of RedrawWindow unless either RDW_UPDATENOW or RDW_ERASENOW is specified.
RDW_INTERNALPAINT	Causes a WM_PAINT message to be posted to the window regardless of whether any portion of the window is invalid.

RDW_INVALIDATE	Invalidates <i>lprcUpdate</i> or <i>hrgnUpdate</i> (only one may be non-NULL). If both are NULL , the entire window is invalidated.
-----------------------	--

The following flags are used to validate the window.

FLAG (VALIDATION)	DESCRIPTION
RDW_NOERASE	Suppresses any pending WM_ERASEBKGND messages.
RDW_NOFRAME	Suppresses any pending WM_NCPAINT messages. This flag must be used with RDW_VALIDATE and is typically used with RDW_NOCHILDREN. RDW_NOFRAME should be used with care, as it could cause parts of a window to be painted improperly.
RDW_NOINTERNALPAINT	Suppresses any pending internal WM_PAINT messages. This flag does not affect WM_PAINT messages resulting from a non-NULL update area.
RDW_VALIDATE	Validates <i>lprcUpdate</i> or <i>hrgnUpdate</i> (only one may be non-NULL). If both are NULL , the entire window is validated. This flag does not affect internal WM_PAINT messages.

The following flags control when repainting occurs. **RedrawWindow** will not repaint unless one of these flags is specified.

FLAG	DESCRIPTION
RDW_ERASENOW	Causes the affected windows (as specified by the RDW_ALLCHILDREN and RDW_NOCHILDREN flags) to receive WM_NCPAINT and WM_ERASEBKGND messages, if necessary, before the function returns. WM_PAINT messages are received at the ordinary time.
RDW_UPDATENOW	Causes the affected windows (as specified by the RDW_ALLCHILDREN and RDW_NOCHILDREN flags) to receive WM_NCPAINT , WM_ERASEBKGND , and WM_PAINT messages, if necessary, before the function returns.

By default, the windows affected by **RedrawWindow** depend on whether the specified window has the WS_CLIPCHILDREN style. Child windows that are not the WS_CLIPCHILDREN style are unaffected; non-WS_CLIPCHILDREN windows are recursively validated or invalidated until a WS_CLIPCHILDREN window is encountered. The following flags control which windows are affected by the **RedrawWindow** function.

FLAG	DESCRIPTION
RDW_ALLCHILDREN	Includes child windows, if any, in the repainting operation.

RDW_NOCHILDREN	Excludes child windows, if any, from the repainting operation.
-----------------------	--

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

When **RedrawWindow** is used to invalidate part of the desktop window, the desktop window does not receive a [WM_PAINT](#) message. To repaint the desktop, an application uses the RDW_ERASE flag to generate a [WM_ERASEBKGND](#) message.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[GetUpdateRect](#)

[GetUpdateRgn](#)

[InvalidateRect](#)

[InvalidateRgn](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

[UpdateWindow](#)

ReleaseDC function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ReleaseDC** function releases a device context (DC), freeing it for use by other applications. The effect of the **ReleaseDC** function depends on the type of DC. It frees only common and window DCs. It has no effect on class or private DCs.

Syntax

```
int ReleaseDC(
    [in] HWND hWnd,
    [in] HDC   hdc
);
```

Parameters

[in] hWnd

A handle to the window whose DC is to be released.

[in] hdc

A handle to the DC to be released.

Return value

The return value indicates whether the DC was released. If the DC was released, the return value is 1.

If the DC was not released, the return value is zero.

Remarks

The application must call the **ReleaseDC** function for each call to the [GetWindowDC](#) function and for each call to the [GetDC](#) function that retrieves a common DC.

An application cannot use the **ReleaseDC** function to release a DC that was created by calling the [CreateDC](#) function; instead, it must use the [DeleteDC](#) function. **ReleaseDC** must be called from the same thread that called [GetDC](#).

Examples

For an example, see [Scaling an Image](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[CreateDC](#)

[DeleteDC](#)

[Device Context Functions](#)

[Device Contexts Overview](#)

[GetDC](#)

[GetWindowDC](#)

ScreenToClient function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ScreenToClient** function converts the screen coordinates of a specified point on the screen to client-area coordinates.

Syntax

```
BOOL ScreenToClient(
    [in] HWND     hWnd,
    LPPOINT lpPoint
);
```

Parameters

[in] **hWnd**

A handle to the window whose client area will be used for the conversion.

lpPoint

A pointer to a [POINT](#) structure that specifies the screen coordinates to be converted.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The function uses the window identified by the *hWnd* parameter and the screen coordinates given in the [POINT](#) structure to compute client coordinates. It then replaces the screen coordinates with the client coordinates. The new coordinates are relative to the upper-left corner of the specified window's client area.

The **ScreenToClient** function assumes the specified point is in screen coordinates.

All coordinates are in device units.

Do not use **ScreenToClient** when in a mirroring situation, that is, when changing from left-to-right layout to right-to-left layout. Instead, use [MapWindowPoints](#). For more information, see "Window Layout and Mirroring" in [Window Features](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

[ClientToScreen](#)

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[MapWindowPoints](#)

[POINT](#)

SetRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetRect** function sets the coordinates of the specified rectangle. This is equivalent to assigning the left, top, right, and bottom arguments to the appropriate members of the **RECT** structure.

Syntax

```
BOOL SetRect(
    [out] LPRECT lprc,
    [in]   int     xLeft,
    [in]   int     yTop,
    [in]   int     xRight,
    [in]   int     yBottom
);
```

Parameters

[out] lprc

Pointer to the **RECT** structure that contains the rectangle to be set.

[in] xLeft

Specifies the x-coordinate of the rectangle's upper-left corner.

[in] yTop

Specifies the y-coordinate of the rectangle's upper-left corner.

[in] xRight

Specifies the x-coordinate of the rectangle's lower-right corner.

[in] yBottom

Specifies the y-coordinate of the rectangle's lower-right corner.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Examples

For an example, see [Using Rectangles](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[CopyRect](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

[SetRectEmpty](#)

SetRectEmpty function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetRectEmpty** function creates an empty rectangle in which all coordinates are set to zero.

Syntax

```
BOOL SetRectEmpty(  
    [out] LPRECT lprc  
>);
```

Parameters

[out] lprc

Pointer to the [RECT](#) structure that contains the coordinates of the rectangle.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Examples

For an example, see [Using Rectangles](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[CopyRect](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

[SetRect](#)

SetWindowRgn function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SetWindowRgn** function sets the window region of a window. The window region determines the area within the window where the system permits drawing. The system does not display any portion of a window that lies outside of the window region.

Syntax

```
int SetWindowRgn(
    [in] HWND hWnd,
    [in] HRGN hRgn,
    [in] BOOL bRedraw
);
```

Parameters

[in] *hWnd*

A handle to the window whose window region is to be set.

[in] *hRgn*

A handle to a region. The function sets the window region of the window to this region.

If *hRgn* is **NULL**, the function sets the window region to **NULL**.

[in] *bRedraw*

Specifies whether the system redraws the window after setting the window region. If *bRedraw* is **TRUE**, the system does so; otherwise, it does not.

Typically, you set *bRedraw* to **TRUE** if the window is visible.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

When this function is called, the system sends the [WM_WINDOWPOSCHANGING](#) and [WM_WINDOWPOSCHANGED](#) messages to the window.

The coordinates of a window's window region are relative to the upper-left corner of the window, not the client area of the window.

Note If the window layout is right-to-left (RTL), the coordinates are relative to the upper-right corner of the window. See [Window Layout and Mirroring](#).

After a successful call to [SetWindowRgn](#), the system owns the region specified by the region handle *hRgn*. The system does not make a copy of the region. Thus, you should not make any further function calls with this region handle. In particular, do not delete this region handle. The system deletes the region handle when it no longer needed.

To obtain the window region of a window, call the [GetWindowRgn](#) function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[GetWindowRgn](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[WM_WINDOWPOSCHANGING](#)

SubtractRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **SubtractRect** function determines the coordinates of a rectangle formed by subtracting one rectangle from another.

Syntax

```
BOOL SubtractRect(
    [out] LPRECT     lprcDst,
    [in]  const RECT *lprcSrc1,
    [in]  const RECT *lprcSrc2
);
```

Parameters

[out] *lprcDst*

A pointer to a **RECT** structure that receives the coordinates of the rectangle determined by subtracting the rectangle pointed to by *lprcSrc2* from the rectangle pointed to by *lprcSrc1*.

[in] *lprcSrc1*

A pointer to a **RECT** structure from which the function subtracts the rectangle pointed to by *lprcSrc2*.

[in] *lprcSrc2*

A pointer to a **RECT** structure that the function subtracts from the rectangle pointed to by *lprcSrc1*.

Return value

If the resulting rectangle is empty, the return value is zero.

If the resulting rectangle is not empty, the return value is nonzero.

Remarks

The function only subtracts the rectangle specified by *lprcSrc2* from the rectangle specified by *lprcSrc1* when the rectangles intersect completely in either the x- or y-direction. For example, if **lprcSrc1* has the coordinates (10,10,100,100) and **lprcSrc2* has the coordinates (50,50,150,150), the function sets the coordinates of the rectangle pointed to by *lprcDst* to (10,10,100,100). If **lprcSrc1* has the coordinates (10,10,100,100) and **lprcSrc2* has the coordinates (50,10,150,150), however, the function sets the coordinates of the rectangle pointed to by *lprcDst* to (10,10,50,100). In other words, the resulting rectangle is the bounding box of the geometric difference.

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[IntersectRect](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

[UnionRect](#)

TabbedTextOutA function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **TabbedTextOut** function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.

Syntax

```
LONG TabbedTextOutA(
    [in] HDC      hdc,
    [in] int      x,
    [in] int      y,
    [in] LPCSTR   lpString,
    [in] int      chCount,
    [in] int      nTabPositions,
    [in] const INT *lpnTabStopPositions,
    [in] int      nTabOrigin
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The x-coordinate of the starting point of the string, in logical units.

[in] `y`

The y-coordinate of the starting point of the string, in logical units.

[in] `lpString`

A pointer to the character string to draw. The string does not need to be zero-terminated, since `nCount` specifies the length of the string.

[in] `chCount`

The [length of the string](#) pointed to by `lpString`.

[in] `nTabPositions`

The number of values in the array of tab-stop positions.

[in] `lpnTabStopPositions`

A pointer to an array containing the tab-stop positions, in logical units. The tab stops must be sorted in increasing order; the smallest x-value should be the first item in the array.

[in] `nTabOrigin`

The x-coordinate of the starting position from which tabs are expanded, in logical units.

Return value

If the function succeeds, the return value is the dimensions, in logical units, of the string. The height is in the high-order word and the width is in the low-order word.

If the function fails, the return value is zero.

Remarks

If the *nTabPositions* parameter is zero and the *lpnTabStopPositions* parameter is **NULL**, tabs are expanded to eight times the average character width.

If *nTabPositions* is 1, the tab stops are separated by the distance specified by the first value in the *lpnTabStopPositions* array.

If the *lpnTabStopPositions* array contains more than one value, a tab stop is set for each value in the array, up to the number specified by *nTabPositions*.

The *nTabOrigin* parameter allows an application to call the **TabbedTextOut** function several times for a single line. If the application calls **TabbedTextOut** more than once with the *nTabOrigin* set to the same value each time, the function expands all tabs relative to the position specified by *nTabOrigin*.

By default, the current position is not used or updated by the **TabbedTextOut** function. If an application needs to update the current position when it calls **TabbedTextOut**, the application can call the **SetTextAlign** function with the *wFlags* parameter set to **TA_UPDATECP**. When this flag is set, the system ignores the *X* and *Y* parameters on subsequent calls to the **TabbedTextOut** function, using the current position instead.

Note For Windows Vista and later, **TabbedTextOut** ignores text alignment when it draws text.

NOTE

The winuser.h header defines **TabbedTextOut** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

API set

ext-ms-win-ntuser-misc-l1-5-1 (introduced in Windows 10, version 10.0.14393)

See also

[DrawText](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTabbedTextExtent](#)

[GrayString](#)

[SelectObject](#)

[SetBkColor](#)

[SetTextAlign](#)

[SetTextColor](#)

[TextOut](#)

TabbedTextOutW function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **TabbedTextOut** function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.

Syntax

```
LONG TabbedTextOutW(
    [in] HDC      hdc,
    [in] int      x,
    [in] int      y,
    [in] LPCWSTR  lpString,
    [in] int      chCount,
    [in] int      nTabPositions,
    [in] const INT *lpnTabStopPositions,
    [in] int      nTabOrigin
);
```

Parameters

[in] `hdc`

A handle to the device context.

[in] `x`

The x-coordinate of the starting point of the string, in logical units.

[in] `y`

The y-coordinate of the starting point of the string, in logical units.

[in] `lpString`

A pointer to the character string to draw. The string does not need to be zero-terminated, since `nCount` specifies the length of the string.

[in] `chCount`

The [length of the string](#) pointed to by `lpString`.

[in] `nTabPositions`

The number of values in the array of tab-stop positions.

[in] `lpnTabStopPositions`

A pointer to an array containing the tab-stop positions, in logical units. The tab stops must be sorted in increasing order; the smallest x-value should be the first item in the array.

[in] `nTabOrigin`

The x-coordinate of the starting position from which tabs are expanded, in logical units.

Return value

If the function succeeds, the return value is the dimensions, in logical units, of the string. The height is in the high-order word and the width is in the low-order word.

If the function fails, the return value is zero.

Remarks

If the *nTabPositions* parameter is zero and the *lpnTabStopPositions* parameter is **NULL**, tabs are expanded to eight times the average character width.

If *nTabPositions* is 1, the tab stops are separated by the distance specified by the first value in the *lpnTabStopPositions* array.

If the *lpnTabStopPositions* array contains more than one value, a tab stop is set for each value in the array, up to the number specified by *nTabPositions*.

The *nTabOrigin* parameter allows an application to call the **TabbedTextOut** function several times for a single line. If the application calls **TabbedTextOut** more than once with the *nTabOrigin* set to the same value each time, the function expands all tabs relative to the position specified by *nTabOrigin*.

By default, the current position is not used or updated by the **TabbedTextOut** function. If an application needs to update the current position when it calls **TabbedTextOut**, the application can call the **SetTextAlign** function with the *wFlags* parameter set to **TA_UPDATECP**. When this flag is set, the system ignores the *X* and *Y* parameters on subsequent calls to the **TabbedTextOut** function, using the current position instead.

Note For Windows Vista and later, **TabbedTextOut** ignores text alignment when it draws text.

NOTE

The winuser.h header defines **TabbedTextOut** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

API set	ext-ms-win-ntuser-misc-l1-5-1 (introduced in Windows 10, version 10.0.14393)
---------	--

See also

[DrawText](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTabbedTextExtent](#)

[GrayString](#)

[SelectObject](#)

[SetBkColor](#)

[SetTextAlign](#)

[SetTextColor](#)

[TextOut](#)

UnionRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **UnionRect** function creates the union of two rectangles. The union is the smallest rectangle that contains both source rectangles.

Syntax

```
BOOL UnionRect(
    [out] LPRECT     lprcDst,
    [in]  const RECT *lprcSrc1,
    [in]  const RECT *lprcSrc2
);
```

Parameters

[out] *lprcDst*

A pointer to the **RECT** structure that will receive a rectangle containing the rectangles pointed to by the *lprcSrc1* and *lprcSrc2* parameters.

[in] *lprcSrc1*

A pointer to the **RECT** structure that contains the first source rectangle.

[in] *lprcSrc2*

A pointer to the **RECT** structure that contains the second source rectangle.

Return value

If the specified structure contains a nonempty rectangle, the return value is nonzero.

If the specified structure does not contain a nonempty rectangle, the return value is zero.

Remarks

The system ignores the dimensions of an empty rectangle that is, a rectangle in which all coordinates are set to zero, so that it has no height or no width.

Because applications can use rectangles for different purposes, the rectangle functions do not use an explicit unit of measure. Instead, all rectangle coordinates and dimensions are given in signed, logical values. The mapping mode and the function in which the rectangle is used determine the units of measure.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

[InflateRect](#)

[IntersectRect](#)

[OffsetRect](#)

[RECT](#)

[Rectangle Functions](#)

[Rectangles Overview](#)

UpdateWindow function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **UpdateWindow** function updates the client area of the specified window by sending a [WM_PAINT](#) message to the window if the window's update region is not empty. The function sends a **WM_PAINT** message directly to the window procedure of the specified window, bypassing the application queue. If the update region is empty, no message is sent.

Syntax

```
BOOL UpdateWindow(  
    [in] HWND hWnd  
)
```

Parameters

[in] hWnd

Handle to the window to be updated.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[ExcludeUpdateRgn](#)

[GetUpdateRect](#)

[GetUpdateRgn](#)

[InvalidateRect](#)

[InvalidateRgn](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[WM_PAINT](#)

ValidateRect function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ValidateRect** function validates the client area within a rectangle by removing the rectangle from the update region of the specified window.

Syntax

```
BOOL ValidateRect(
    [in] HWND      hWnd,
    [in] const RECT *lpRect
);
```

Parameters

[in] hWnd

Handle to the window whose update region is to be modified. If this parameter is **NULL**, the system invalidates and redraws all windows and sends the **WM_ERASEBKGND** and **WM_NCPAINT** messages to the window procedure before the function returns.

[in] lpRect

Pointer to a [RECT](#) structure that contains the client coordinates of the rectangle to be removed from the update region. If this parameter is **NULL**, the entire client area is removed.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The [BeginPaint](#) function automatically validates the entire client area. Neither the **ValidateRect** nor [ValidateRgn](#) function should be called if a portion of the update region must be validated before the next [WM_PAINT](#) message is generated.

The system continues to generate [WM_PAINT](#) messages until the current update region is validated.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[BeginPaint](#)

[InvalidateRect](#)

[InvalidateRgn](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

[ValidateRgn](#)

[WM_PAINT](#)

ValidateRgn function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **ValidateRgn** function validates the client area within a region by removing the region from the current update region of the specified window.

Syntax

```
BOOL ValidateRgn(  
    [in] HWND hWnd,  
    [in] HRGN hRgn  
)
```

Parameters

[in] hWnd

Handle to the window whose update region is to be modified.

[in] hRgn

Handle to a region that defines the area to be removed from the update region. If this parameter is **NULL**, the entire client area is removed.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

The specified region must have been created by a region function. The region coordinates are assumed to be client coordinates.

The [BeginPaint](#) function automatically validates the entire client area. Neither the [ValidateRect](#) nor [ValidateRgn](#) function should be called if a portion of the update region must be validated before the next [WM_PAINT](#) message is generated.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-0 (introduced in Windows 8)

See also

[BeginPaint](#)

[ExcludeUpdateRgn](#)

[InvalidateRect](#)

[InvalidateRgn](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[ValidateRect](#)

[WM_PAINT](#)

WindowFromDC function (winuser.h)

4/21/2022 • 2 minutes to read • [Edit Online](#)

The **WindowFromDC** function returns a handle to the window associated with the specified display device context (DC). Output functions that use the specified device context draw into this window.

Syntax

```
HWND WindowFromDC(  
    [in] HDC hDC  
)
```

Parameters

[in] hDC

Handle to the device context from which a handle to the associated window is to be retrieved.

Return value

The return value is a handle to the window associated with the specified DC. If no window is associated with the specified DC, the return value is **NULL**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-draw-l1-1-1 (introduced in Windows 8.1)

See also

[GetDC](#)

[GetDCEx](#)

[GetWindowDC](#)

[Painting and Drawing Functions](#)

Painting and Drawing Overview

xpsprint.h header

4/21/2022 • 2 minutes to read • [Edit Online](#)

This header is used by Windows GDI. For more information, see:

- [Windows GDI](#)

xpsprint.h contains the following programming interfaces:

Interfaces

[IXpsPrintJob](#)

Provides access to a print job that is currently in progress.

[IXpsPrintJobStream](#)

A write-only stream interface into which an application writes print job data.

Functions

[StartXpsPrintJob](#)

Starts printing an XPS document stream to a printer.

[StartXpsPrintJob1](#)

Creates a print job for sending XPS document content to a printer.

Structures

[XPS_JOB_STATUS](#)

Contains a snapshot of job status.

Enumerations

[XPS_JOB_COMPLETION](#)

Indicates the completion status of a print job.